

# **Coding with Jiangly**

2.7

2023-12-31

# Contents

Introduction . . . . .	28
gray . . . . .	31
brute force . . . . .	31
1: Erase First or Second Letter . . . . .	31
2: Easy As ABC . . . . .	32
3: Don't Try to Count . . . . .	33
4: Grandma Capa Knits a Scarf . . . . .	34
5: Two Vessels . . . . .	35
6: Download More RAM . . . . .	36
7: Cirno's Perfect Bitmasks Classroom . . . . .	37
8: Pizza Separation . . . . .	38
9: Beautiful Divisors . . . . .	39
10: Fancy Coins . . . . .	40
11: ACM ICPC . . . . .	41
12: Local Extrema . . . . .	42
13: Another Permutation Problem . . . . .	43
14: Fibonacchairsis . . . . .	44
15: Desorting . . . . .	45
16: Longest Divisors Interval . . . . .	46
17: Balanced Round . . . . .	47
18: Rudolph and Tic-Tac-Toe . . . . .	48
19: Trust Nobody . . . . .	49
20: Gold Rush . . . . .	50
constructive algorithms . . . . .	51
21: Binary Imbalance . . . . .	51
22: Sorting with Twos . . . . .	52
23: Chips on the Board . . . . .	53
24: MEXanized Array . . . . .	54
25: Make It Zero . . . . .	55
26: XOR Palindromes . . . . .	56

27: green_gold_dog, array and permutation . . . . .	57
28: Madoka and the Elegant Gift . . . . .	58
29: Patchouli's Magical Talisman . . . . .	59
30: Difference of GCDs . . . . .	60
31: Two Binary Strings . . . . .	61
32: Prime Deletion . . . . .	62
33: Swap and Reverse . . . . .	63
34: Increasing and Decreasing . . . . .	65
35: Sequence Game . . . . .	66
36: Not a Substring . . . . .	67
37: Yet Another Permutation Problem . . . . .	68
38: Olya and Game with Arrays . . . . .	69
39: United We Stand . . . . .	70
40: Escalator Conversations . . . . .	72
greedy . . . . .	73
41: Training Before the Olympiad . . . . .	73
42: Unnatural Language Processing . . . . .	74
43: Three Activities . . . . .	75
44: Quests . . . . .	77
45: Begginer's Zelda . . . . .	78
46: Removal of Unattractive Pairs . . . . .	79
47: Chip and Ribbon . . . . .	80
48: Line Trip . . . . .	81
49: StORage room . . . . .	82
50: Halloumi Boxes . . . . .	83
51: Getting Points . . . . .	84
52: Cover in Water . . . . .	85
53: Yarik and Array . . . . .	87
54: Points and Minimum Distance . . . . .	88
55: Haunted House . . . . .	89
56: Simple Design . . . . .	90
57: Gamer Hemose . . . . .	91
58: Make Them Equal . . . . .	92
59: Helmets in Night Light . . . . .	93
60: How Much Does Daytona Cost? . . . . .	94
implementation . . . . .	95
61: 2023 . . . . .	95
62: Can I Square? . . . . .	96

63: Not Quite Latin Square . . . . .	97
64: Odd One Out . . . . .	97
65: Distinct Buttons . . . . .	98
66: Problem-solving Log . . . . .	99
67: Rating Increase . . . . .	100
68: YetanotherbrokenKeoard . . . . .	101
69: Rook . . . . .	103
70: 250 Thousand Tons of TNT . . . . .	104
71: Milica and String . . . . .	105
72: Perfect Square . . . . .	106
73: 1D Eraser . . . . .	107
74: Target Practice . . . . .	108
75: Short Sort . . . . .	109
76: Divine Array . . . . .	110
77: Two Subsequences . . . . .	111
78: The Corridor or There and Back Again . . . . .	112
79: Life of a Flower . . . . .	113
80: Game . . . . .	115
math . . . . .	116
81: Two Divisors . . . . .	116
82: Least Product . . . . .	116
83: Make Almost Equal With Mod . . . . .	118
84: Preparing for the Contest . . . . .	119
85: Constructive Problems . . . . .	120
86: Laura and Operations . . . . .	121
87: Game with Integers . . . . .	122
88: Treasure Chest . . . . .	123
89: Deja Vu . . . . .	124
90: Hemose Shopping . . . . .	125
91: Special Numbers . . . . .	126
92: Consecutive Sum Riddle . . . . .	127
93: Three Threadlets . . . . .	128
94: Fear of the Dark . . . . .	129
95: Sum of Three . . . . .	130
96: Joyboard . . . . .	131
97: Goals of Victory . . . . .	132
98: Vasilije in Cacak . . . . .	133
99: Aleksa and Stack . . . . .	134

100: Good Kid . . . . .	134
misc . . . . .	135
101: LOL Lovers . . . . .	135
102: Swap and Delete . . . . .	136
103: Increasing Sequence . . . . .	138
104: Rigged! . . . . .	138
105: Friendly Arrays . . . . .	140
106: Luntik and Subsequences . . . . .	141
107: Luntik and Concerts . . . . .	142
109: Era . . . . .	143
110: A.M. Deviation . . . . .	144
111: Coloring Rectangles . . . . .	145
112: Mathematical Addition . . . . .	146
113: Plus Minus Permutation . . . . .	146
114: Non-coprime Split . . . . .	147
115: GCD Arrays . . . . .	148
116: Fun with Even Subarrays . . . . .	149
117: Min Max Swap . . . . .	150
118: Power Walking . . . . .	151
119: Hard Way . . . . .	152
120: Madoka and Math Dad . . . . .	153
121: Direction Change . . . . .	154
122: Difference Operations . . . . .	155
123: Ambitious Kid . . . . .	156
124: MEX Repetition . . . . .	157
125: Channel . . . . .	158
126: Gift Carpet . . . . .	159
127: Boxes Packing . . . . .	160
128: The Modcrab . . . . .	161
129: Hungry Student Problem . . . . .	162
130: Find Extra One . . . . .	163
sortings . . . . .	164
131: Building an Aquarium . . . . .	164
132: 2D Traveling . . . . .	165
133: Social Distance . . . . .	166
134: Split Sort . . . . .	167
135: Flower City Fence . . . . .	169
136: Destroyer . . . . .	170

137: Assembly via Minimums . . . . .	171
138: Monsters . . . . .	172
139: Parity Sort . . . . .	173
140: Ten Words of Wisdom . . . . .	174
141: To My Critics . . . . .	175
142: The Man who became a God . . . . .	175
143: Vika and the Bridge . . . . .	177
144: Rudolf and the Another Competition . . . . .	178
145: Lamps . . . . .	180
146: Restore the Weather . . . . .	181
147: Counting Orders . . . . .	182
148: Helpful Maths . . . . .	184
149: Sort with Step . . . . .	184
150: Karina and Array . . . . .	185
green . . . . .	187
brute force . . . . .	187
151: Good Triples . . . . .	187
152: Chtholly's request . . . . .	188
153: Binary String Copying . . . . .	189
154: Vampiric Powers, anyone? . . . . .	190
155: Tracking Segments . . . . .	192
156: Ice Sculptures . . . . .	193
157: Martian Dollar . . . . .	194
158: The Text Splitting . . . . .	195
159: Tear It Apart . . . . .	196
160: Fire Again . . . . .	197
161: Constructive Problem . . . . .	198
162: Accounting . . . . .	200
163: IQ test . . . . .	201
164: Bargaining Table . . . . .	202
165: Triangle . . . . .	203
166: Make It Permutation . . . . .	208
167: Pull Your Luck . . . . .	209
168: Unforgivable Curse (hard version) . . . . .	210
169: Unforgivable Curse (easy version) . . . . .	211
170: Same Count One . . . . .	213
constructive algorithms . . . . .	214
171: Jumping Through Segments . . . . .	214

172: ABBC or BACB . . . . .	215
173: Salyg1n and the MEX Game . . . . .	216
174: Madoka and Childish Pranks . . . . .	218
175: Two-Colored Dominoes . . . . .	219
176: Kolya and Movie Theatre . . . . .	221
177: Row Major . . . . .	222
178: Insert Zero and Invert Prefix . . . . .	223
179: No Prime Differences . . . . .	224
180: Bracket Coloring . . . . .	225
181: Flipper . . . . .	227
182: Find Color . . . . .	229
183: Li Hua and Chess . . . . .	229
184: Search in Parallel . . . . .	231
185: Umka and a Long Flight . . . . .	233
186: Shocking Arrangement . . . . .	234
187: Sum on Subarrays . . . . .	235
188: Sequence Master . . . . .	237
189: Chris and Magic Square . . . . .	238
190: The Very Beautiful Blanket . . . . .	239
dp . . . . .	240
191: Anji's Binary Tree . . . . .	240
192: Block Sequence . . . . .	242
194: Game on Permutation . . . . .	243
195: Nastya and Potions . . . . .	244
196: Strong Password . . . . .	246
197: Tenzing and Balls . . . . .	247
198: Copil Copac Draws Trees . . . . .	248
199: Round Table Knights . . . . .	249
200: Anfisa the Monkey . . . . .	250
201: Living Sequence . . . . .	251
202: Hard problem . . . . .	252
203: Serval and Toxel's Arrays . . . . .	253
204: Ice and Fire . . . . .	254
205: Remove the Bracket . . . . .	256
206: Playoff . . . . .	257
207: Hamiltonian Wall . . . . .	258
208: Hossam and Friends . . . . .	259
209: The Humanoid . . . . .	260

210: Zero-Sum Prefixes . . . . .	262
greedy . . . . .	263
211: Romantic Glasses . . . . .	263
212: Watering an Array . . . . .	264
213: Heavy Intervals . . . . .	265
214: Game with Marbles (Hard Version) . . . . .	267
215: Game with Marbles (Easy Version) . . . . .	268
216: Game with Multiset . . . . .	269
217: Largest Subsequence . . . . .	270
218: Add, Divide and Floor . . . . .	272
219: Theofanis' Nightmare . . . . .	273
220: Insert and Equalize . . . . .	274
221: Alex's whims . . . . .	275
222: Queue Sort . . . . .	277
223: Milena and Admirer . . . . .	278
224: Smilo and Monsters . . . . .	279
225: Dances (Easy version) . . . . .	280
226: Iva & Pav . . . . .	281
227: Reverse Madness . . . . .	283
228: Card Game . . . . .	284
229: Sets and Union . . . . .	285
230: Make it Alternating . . . . .	287
implementation . . . . .	288
231: Anonymous Informant . . . . .	288
232: Decreasing String . . . . .	289
233: Fill in the Matrix . . . . .	291
234: Dominant Character . . . . .	292
235: Game of Ball Passing . . . . .	293
236: Queries for the Array . . . . .	294
237: Petya and Catacombs . . . . .	295
238: K-Dominant Character . . . . .	296
239: Prefix Permutation Sums . . . . .	297
240: The Morning Star . . . . .	299
241: We Were Both Children . . . . .	300
242: Particles . . . . .	301
243: Rudolf and Snowflakes (simple version) . . . . .	302
244: LuoTianyi and the Show . . . . .	303
245: Hits Different . . . . .	305

246: Easy Number Challenge . . . . .	306
247: XOR and OR . . . . .	307
248: Cd and pwd commands . . . . .	307
249: Progress Bar . . . . .	309
250: Guilty - to the kitchen! . . . . .	310
math . . . . .	311
251: Yarik and Musical Notes . . . . .	311
252: Torn Lucky Ticket . . . . .	312
253: Rubik's Cube Coloring (easy version) . . . . .	314
254: Divide and Equalize . . . . .	315
255: Money Trees . . . . .	317
256: Colorful Table . . . . .	318
257: Vupsen, Pupsen and 0 . . . . .	319
258: Array Elimination . . . . .	320
259: Minimize Distance . . . . .	321
260: Battling with Numbers . . . . .	322
261: Divisor Chain . . . . .	324
262: Ice Cream Balls . . . . .	326
263: Position in Fraction . . . . .	327
264: Pride . . . . .	327
265: The Walkway . . . . .	328
266: Almost Identity Permutations . . . . .	330
267: Sum and Product . . . . .	331
268: Power of Points . . . . .	332
269: Strong Vertices . . . . .	334
270: Dual (Easy Version) . . . . .	335
misc . . . . .	336
271: Greetings . . . . .	336
272: Effects of Anti Pimples . . . . .	338
273: Rumor . . . . .	339
274: Hometask . . . . .	340
275: Wooden Toy Festival . . . . .	342
276: Round Dance . . . . .	343
277: Dreaming of Freedom . . . . .	345
278: Forever Winter . . . . .	346
279: LCM Challenge . . . . .	347
280: Almost Increasing Subsequence . . . . .	347
281: Strongly Composite . . . . .	348

282: Petya and Inequiations . . . . .	350
283: Painting Eggs . . . . .	351
284: Hamsters and Tigers . . . . .	352
285: Lucky Tickets . . . . .	353
286: Email address . . . . .	354
287: Making Anti-Palindromes . . . . .	355
288: Replace To Make Regular Bracket Sequence . . . . .	356
289: Load Balancing . . . . .	357
290: Queries about less or equal elements . . . . .	358
291: Extract Numbers . . . . .	359
292: Queries on a String . . . . .	360
293: Cola . . . . .	361
294: Repaintings . . . . .	362
295: Spelling Check . . . . .	363
296: Multiplication Table . . . . .	363
297: Company Income Growth . . . . .	364
298: Fractal . . . . .	365
299: Extra-terrestrial Intelligence . . . . .	366
300: Road Map . . . . .	367
blue . . . . .	368
constructive algorithms . . . . .	368
301: Colorful Grid . . . . .	368
302: Array Painting . . . . .	370
303: Rudolph and Mimic . . . . .	371
304: Tenzing and His Animal Friends . . . . .	373
305: Ira and Flamenco . . . . .	374
306: Fish Graph . . . . .	376
307: Unique Palindromes . . . . .	378
308: 3-cycles . . . . .	380
309: Make Palindrome . . . . .	380
310: Four Segments . . . . .	381
311: Binary String Sorting . . . . .	383
312: Train Splitting . . . . .	384
313: Watch the Videos . . . . .	385
314: Letter Exchange . . . . .	387
315: Bit Guessing Game . . . . .	388
316: Lucky Permutation . . . . .	390
317: Boris and His Amazing Haircut . . . . .	391

318: Range = Sum . . . . .	393
319: Restore the Permutation . . . . .	394
320: Yet Another Problem . . . . .	395
dp . . . . .	397
321: Kim's Quest . . . . .	397
322: Merge Not Sort . . . . .	398
323: Robot Queries . . . . .	400
324: Bakry and Partitioning . . . . .	402
325: The Number of Imposters . . . . .	403
326: Minimum Maximum Distance . . . . .	405
327: Completely Searching for Inversions . . . . .	406
328: Magic Will Save the World . . . . .	408
329: Credit Card . . . . .	409
330: Andrey and Escape from Capygrad . . . . .	411
331: PermuTree (easy version) . . . . .	412
332: Rudolf and CodeVid-23 . . . . .	414
333: Omsk Metro (simple version) . . . . .	415
334: Ransom Numbers . . . . .	418
335: Ksyusha and Chinchilla . . . . .	419
336: Caesar's Legions . . . . .	421
337: Running Miles . . . . .	422
338: Don't Blame Me . . . . .	423
339: Xenia and Weights . . . . .	425
340: Phone Talks . . . . .	426
graphs . . . . .	427
341: Time Travel . . . . .	427
342: Mad City . . . . .	429
343: Gardening Friends . . . . .	431
344: Igor In the Museum . . . . .	432
345: Roads not only in Berland . . . . .	434
346: Roads in Berland . . . . .	435
347: System Administrator . . . . .	436
348: Dijkstra? . . . . .	437
349: Two Paths . . . . .	438
350: A Wide, Wide Graph . . . . .	439
351: Directed Roads . . . . .	440
352: Game on Axis . . . . .	442
353: Friendly Spiders . . . . .	444

354: SlavicG's Favorite Problem . . . . .	446
greedy . . . . .	447
355: Bicycles . . . . .	447
356: Split Plus K . . . . .	449
357: Programming Competition . . . . .	450
358: Triangle Construction . . . . .	451
359: Shift and Reverse . . . . .	452
360: Yet Another Monster Fight . . . . .	454
361: Maximum And Queries (easy version) . . . . .	455
362: Absolute Beauty . . . . .	457
363: Neutral Tonality . . . . .	458
364: Dances (Hard Version) . . . . .	459
365: Medium Design . . . . .	461
366: Tree XOR . . . . .	462
367: Prefix Purchase . . . . .	464
368: Korney Korneevich and XOR (easy version) . . . . .	465
369: Frog Traveler . . . . .	466
370: Cyclic Operations . . . . .	467
371: Candy Party (Easy Version) . . . . .	469
372: Manipulating History . . . . .	471
373: Sorting By Multiplication . . . . .	472
374: Matrix Cascade . . . . .	473
implementation . . . . .	474
375: Accumulator Apex . . . . .	474
376: The Third Letter . . . . .	475
377: Professor Higashikata . . . . .	477
378: k-th equality . . . . .	478
379: Survey in Class . . . . .	480
380: Maximum Xor Secondary . . . . .	481
381: Nearest Fraction . . . . .	482
382: Petya and Divisors . . . . .	483
383: Parking Lot . . . . .	483
384: Game of chess unfinished . . . . .	485
385: Computer Game . . . . .	486
386: Warehouse . . . . .	487
387: The Butcher . . . . .	489
388: Schedule . . . . .	491
389: Codeforces World Finals . . . . .	492

390: Mail Stamps . . . . .	494
391: Sequence of points . . . . .	495
392: Jabber ID . . . . .	496
393: Li Hua and Tree . . . . .	497
394: BerOS file system . . . . .	499
math . . . . .	500
395: Mathematical Problem . . . . .	500
396: Divisibility Test . . . . .	501
397: XOR Construction . . . . .	502
398: Suspicious logarithms . . . . .	503
399: Vasilije Loves Number Theory . . . . .	505
400: Sum of XOR Functions . . . . .	507
401: Selling a Menagerie . . . . .	508
402: Yet Another Sorting Problem . . . . .	510
403: Madoka and the Best School in Russia . . . . .	512
404: Remove Extra One . . . . .	513
405: XK Segments . . . . .	514
406: Marco and GCD Sequence . . . . .	515
407: Ralph And His Magic Field . . . . .	516
408: Dual (Hard Version) . . . . .	517
409: Lisa and the Martians . . . . .	519
410: Imbalanced Arrays . . . . .	520
411: Ntarsis' Set . . . . .	522
412: Vika and Price Tags . . . . .	523
413: Rudolf and Snowflakes (hard version) . . . . .	524
414: Rating System . . . . .	526
misc . . . . .	527
415: Collapsing Strings . . . . .	527
416: Unusual Entertainment . . . . .	528
417: Nephren gives a riddle . . . . .	530
418: Maximum Subsequence . . . . .	532
419: Fancy Number . . . . .	533
420: LuoTianyi and the Floating Islands (Easy Version) . . . . .	534
421: Xenia and Bit Operations . . . . .	536
422: Comb . . . . .	537
423: Magic Triples (Easy Version) . . . . .	538
424: The Union of k-Segments . . . . .	540
425: Phone Number . . . . .	541

426: Hyperdrive . . . . .	543
427: Let's Go Rolling! . . . . .	544
428: Old Berland Language . . . . .	545
429: Black Cells . . . . .	546
430: Animals . . . . .	547
431: Wonderful Randomized Sum . . . . .	548
432: String Problem . . . . .	549
433: Flea . . . . .	551
434: Shooting Gallery . . . . .	551
435: Segments . . . . .	553
436: Checkout Assistant . . . . .	554
437: Platforms . . . . .	554
438: Fish . . . . .	555
439: Monitor . . . . .	556
440: Laser . . . . .	557
441: Long Legs . . . . .	558
442: Climbing the Tree . . . . .	559
443: Unlucky Numbers . . . . .	561
444: Candy Store . . . . .	562
<b>violet . . . . .</b>	<b>563</b>
brute force . . . . .	563
445: Journey . . . . .	563
446: Chocolate Bar . . . . .	565
447: Number With The Given Amount Of Divisors . . . . .	566
448: Sum Graph . . . . .	567
449: Different Arrays . . . . .	569
450: Sheikh (Hard Version) . . . . .	571
data structures . . . . .	572
451: Maximize The Value . . . . .	572
452: A Growing Tree . . . . .	574
453: Anya and the Mysterious String . . . . .	575
454: Vlad and the Mountains . . . . .	577
455: Minimum spanning tree for each edge . . . . .	579
456: Parade . . . . .	580
457: Sereja and Brackets . . . . .	582
458: Petr . . . . .	583
459: Petya, Petya, Petr, and Palindromes . . . . .	584
460: Mishka and Interesting sum . . . . .	585

461: Hot Start Up (hard version) . . . . .	587
462: Koxia and Game . . . . .	588
463: Count Binary Strings . . . . .	590
464: Hossam and (sub-)palindromic tree . . . . .	592
465: Multi-Colored Segments . . . . .	594
dp . . . . .	596
466: Light Bulbs (Easy Version) . . . . .	596
467: Array Collapse . . . . .	598
468: Count BFS Graph . . . . .	600
469: Transitive Graph . . . . .	602
470: Small GCD . . . . .	603
471: Counting Rhyme . . . . .	604
472: Pchelyonok and Segments . . . . .	606
473: Candy Party (Hard Version) . . . . .	607
474: Speedrun . . . . .	609
475: Maximum Questions . . . . .	611
476: Unusual Sequences . . . . .	612
477: Square Subsets . . . . .	613
478: Counting Arrays . . . . .	615
479: Ralph and Mushrooms . . . . .	617
480: Cowboys . . . . .	618
481: Vanya and Brackets . . . . .	620
482: Ball Sorting . . . . .	620
483: Range Sorting (Easy Version) . . . . .	622
484: Let's Play Osu! . . . . .	624
485: Petya and Spiders . . . . .	625
greedy . . . . .	626
486: Freedom of Choice . . . . .	626
487: Autosynthesis . . . . .	628
488: Treelabeling . . . . .	629
489: Ithea Plays With Chtholly . . . . .	631
490: Gluttony . . . . .	632
491: Counting Graphs . . . . .	633
492: Pairs of Segments . . . . .	635
493: Gadgets for dollars and pounds . . . . .	636
494: Rearrange Brackets . . . . .	638
495: C*++ Calculations . . . . .	640
496: Parquet . . . . .	641

497: Seller Bob . . . . .	643
498: Monsters . . . . .	644
499: Accommodation . . . . .	645
500: The way home . . . . .	647
501: Maximum Subarray . . . . .	648
502: Another Array Problem . . . . .	649
503: The Harmonization of XOR . . . . .	650
504: Timofey and Black-White Tree . . . . .	653
505: The Human Equation . . . . .	655
implementation . . . . .	656
506: Is It Flower? . . . . .	656
507: Sweets Game . . . . .	658
508: Chris and Road . . . . .	659
509: Copy of a Copy of a Copy . . . . .	661
math . . . . .	662
510: Blueprint for Seating . . . . .	662
511: Cyclic MEX . . . . .	664
512: Geo Game . . . . .	665
513: Monocarp and the Set . . . . .	667
514: Salyg1n and Array (simple version) . . . . .	669
515: Replace With Product . . . . .	670
516: Guess Game . . . . .	671
517: The Great Equalizer . . . . .	674
518: String Mark . . . . .	675
519: Below the Diagonal . . . . .	677
520: The BOSS Can Count Pairs . . . . .	679
521: Red-Blue Operations (Easy Version) . . . . .	680
522: Collisions . . . . .	681
523: Longest Subsequence . . . . .	683
524: Intersection . . . . .	684
525: Equation . . . . .	685
526: Vlad and the Nice Paths (easy version) . . . . .	686
527: Another Wine Tasting Event . . . . .	687
528: Moving Dots . . . . .	689
misc . . . . .	690
530: Happy Sets . . . . .	690
531: Restoration of string . . . . .	691
532: More Wrong . . . . .	693

533: Bertown roads . . . . .	694
534: Rectangle Puzzle . . . . .	695
535: Area of Two Circles' Intersection . . . . .	696
536: Smart Boy . . . . .	697
537: Moon Craters . . . . .	698
538: What Has Dirichlet Got to Do with That? . . . . .	701
539: Ant on the Tree . . . . .	702
540: Thanos Nim . . . . .	703
541: Stripe 2 . . . . .	704
542: Flag 2 . . . . .	705
543: Start of the season . . . . .	707
544: The hat . . . . .	707
545: Lucky Sorting . . . . .	709
546: Work Group . . . . .	711
547: Graph Cost . . . . .	712
548: GCD Queries . . . . .	713
549: Carry Bit . . . . .	714
551: Wish I Knew How to Sort . . . . .	716
552: MEX vs MED . . . . .	717
<b>orange . . . . .</b>	<b>719</b>
brute force . . . . .	719
553: Lights . . . . .	719
554: Swapping Characters . . . . .	721
555: Solitaire . . . . .	722
556: Safe . . . . .	725
557: Race . . . . .	726
558: Safe cracking . . . . .	728
559: Toys . . . . .	729
560: Multitest Generator . . . . .	730
561: Tree Master . . . . .	732
562: Serval and Shift-Shift-Shift . . . . .	733
563: Three Chairs . . . . .	735
564: Rectangle Shrinking . . . . .	737
565: Decomposition . . . . .	739
566: Kirill and Company . . . . .	741
data structures . . . . .	743
567: Happy Life in University . . . . .	743
568: Yet Another Inversions Problem . . . . .	744

569: Light Bulbs (Hard Version) . . . . .	746
570: Sofia and Strings . . . . .	748
571: Infinite Card Game . . . . .	750
572: wxhtzdy ORO Tree . . . . .	752
573: Optimal Insertion . . . . .	755
574: Arithmetic Operations . . . . .	757
575: Exotic Queries . . . . .	758
576: Almost Difference . . . . .	760
577: Eyes Closed . . . . .	761
578: Ralph And His Tour in Binary Country . . . . .	763
579: Envy . . . . .	764
580: Trees and Segments . . . . .	766
581: Max to the Right of Min . . . . .	768
582: Competition . . . . .	769
583: Tenzing and Triangle . . . . .	771
584: Omsk Metro (hard version) . . . . .	772
585: MEX of LCM . . . . .	775
586: Fill the Matrix . . . . .	776
dfs and similar . . . . .	777
587: Lomsat gelral . . . . .	777
588: Ring Road 2 . . . . .	779
589: Between . . . . .	781
590: Scheme . . . . .	782
591: Sum Over Zero . . . . .	784
592: Tokens on Graph . . . . .	785
593: Edge Reverse . . . . .	787
dp . . . . .	789
594: Small Permutation Problem (Easy Version) . . . . .	789
595: Compressed Tree . . . . .	791
596: Rubik's Cube Coloring (hard version) . . . . .	792
597: Another MEX Problem . . . . .	795
598: Imagination Castle . . . . .	797
599: Digital Wallet . . . . .	798
600: Non-Intersecting Subpermutations . . . . .	800
601: Balanced String . . . . .	801
602: Railguns . . . . .	802
603: Combinatorics Problem . . . . .	805
604: Petya and Coloring . . . . .	806

605: Nuclear Fusion . . . . .	807
606: Lesson Timetable . . . . .	809
607: Bath Queue . . . . .	810
608: Chain Chips . . . . .	811
609: Sequence . . . . .	813
610: Vlad and the Nice Paths (hard version) . . . . .	814
611: A Simple Task . . . . .	815
612: There Should Be a Lot of Maximums . . . . .	816
613: Minibuses on Venus (easy version) . . . . .	818
implementation . . . . .	820
614: Sweets for Everyone! . . . . .	820
615: Bowls . . . . .	822
616: How Many Squares? . . . . .	823
617: Persistent Bookcase . . . . .	825
618: Symmetree . . . . .	827
619: Hospital Queue . . . . .	829
620: Monsters (hard version) . . . . .	831
math . . . . .	833
621: Evaluate It and Back Again . . . . .	833
622: Hemose in ICPC ? . . . . .	834
623: Vika and Bonuses . . . . .	836
624: In Search of Truth (Easy Version) . . . . .	838
625: LuoTianyi and the Floating Islands (Hard Version) . . . . .	839
626: Strange town . . . . .	841
627: Magic Triples (Hard Version) . . . . .	842
628: Square Root of Permutation . . . . .	844
629: Berland Square . . . . .	845
630: Li Hua and Array . . . . .	846
631: Memory and Scores . . . . .	848
632: ZS and The Birthday Paradox . . . . .	850
633: City Union . . . . .	851
634: Explosions? . . . . .	854
635: Gaining Rating . . . . .	856
636: Node Pairs . . . . .	857
637: Valid Bitonic Permutations . . . . .	858
638: Game of the Year . . . . .	859
639: Partial Sorting . . . . .	861
640: Yet Another Array Counting Problem . . . . .	862

misc . . . . .	864
641: Salyg1n and Array (hard version) . . . . .	864
642: Playoff Fixing . . . . .	865
643: Sausage Maximization . . . . .	868
644: Cannon . . . . .	870
645: Hercule Poirot Problem . . . . .	871
646: Nearest vectors . . . . .	873
647: New Game with a Chess Piece . . . . .	878
648: Test . . . . .	879
649: Berland collider . . . . .	881
650: Deletion of Repeats . . . . .	886
651: Queue . . . . .	887
652: Vittorio Plays with LEGO Bricks . . . . .	890
653: Chemistry Lab . . . . .	891
654: Intersection and Union . . . . .	893
rose . . . . .	895
brute force . . . . .	895
655: Vova Escapes the Matrix . . . . .	895
656: Removing Graph . . . . .	897
657: Balancing Weapons . . . . .	899
658: Survival of the Weakest (easy version) . . . . .	901
659: Routing . . . . .	902
660: Serval and Music Game . . . . .	905
661: Magician and Pigs (Easy Version) . . . . .	906
662: Rebrending . . . . .	908
663: Project Manager . . . . .	910
664: Josuke and Complete Graph . . . . .	912
665: Function Sum . . . . .	913
666: Hossam and a Letter . . . . .	915
667: Balance (Hard version) . . . . .	917
constructive algorithms . . . . .	918
668: Interactive Game with Coloring . . . . .	918
669: Great Grids . . . . .	920
670: In Search of Truth (Hard Version) . . . . .	922
671: Aztec Catacombs . . . . .	923
672: Two Paths . . . . .	925
673: The Fox and the Complete Tree Traversal . . . . .	928
674: Square Table . . . . .	930

675: Multithreading . . . . .	931
676: Greedy Change . . . . .	933
677: Guess the String . . . . .	934
678: Inverse Transformation . . . . .	937
679: Xorcerer's Stones . . . . .	939
680: Cactus Wall . . . . .	941
681: Doremy's Experimental Tree . . . . .	943
682: Make It Connected . . . . .	944
683: The Beach . . . . .	946
684: Joking (Easy Version) . . . . .	948
data structures . . . . .	951
685: Field Should Not Be Empty . . . . .	951
686: Babysitting . . . . .	953
687: Colorful Constructive . . . . .	954
688: Minimum Array . . . . .	956
689: Kuzya and Homework . . . . .	958
690: Frequency Queries . . . . .	959
691: Madoka and the Sixth-graders . . . . .	961
692: Willem, Chtholly and Seniorious . . . . .	963
693: Fast Travel Text Editor . . . . .	966
694: Rollbacks (Hard Version) . . . . .	967
695: Rollbacks (Easy Version) . . . . .	969
696: More Queries to Array... . . . . .	971
697: Editorial for Two . . . . .	972
698: Palindrome Partition . . . . .	974
699: Range Sorting (Hard Version) . . . . .	976
700: Walk the Runway . . . . .	978
701: LuoTianyi and XOR-Tree . . . . .	979
702: Frogs and mosquitoes . . . . .	981
703: Shooting Gallery . . . . .	982
704: Don't fear, DravDe is kind . . . . .	985
dp . . . . .	986
705: Construct Tree . . . . .	986
706: Small Permutation Problem (Hard Version) . . . . .	988
707: One-X . . . . .	989
708: Maximum And Queries (hard version) . . . . .	991
709: Fancy Arrays . . . . .	992
710: Vasya and Maximum Profit . . . . .	994

711: I Wanna be the Team Leader . . . . .	996
712: Axel and Marston in Bitland . . . . .	998
713: Korney Korneevich and XOR (hard version) . . . . .	999
714: Travel Plan . . . . .	1001
715: Poachers . . . . .	1003
716: Non-equal Neighbours . . . . .	1005
717: Keen Tree Calculation . . . . .	1007
718: Mighty Rock Tower . . . . .	1013
719: Divide, XOR, and Conquer . . . . .	1015
720: Strange Calculation and Cats . . . . .	1017
721: Maximum Element . . . . .	1019
722: Connecting Vertices . . . . .	1021
723: Maximum Monogonosity . . . . .	1022
724: Expected Destruction . . . . .	1024
greedy . . . . .	1025
725: Alice and Recoloring 1 . . . . .	1025
726: Royal Questions . . . . .	1027
727: Two Permutations (Easy Version) . . . . .	1029
728: Tenzing and Tree . . . . .	1031
730: Hyperregular Bracket Strings . . . . .	1033
731: Population Size . . . . .	1034
732: Red-Blue Operations (Hard Version) . . . . .	1036
733: Trial for Chief . . . . .	1037
734: King's Problem? . . . . .	1039
735: Traveling in Berland . . . . .	1040
736: Labeling the Tree with Distances . . . . .	1042
737: Colored Subgraphs . . . . .	1044
738: Spinach Pizza . . . . .	1046
739: Library game . . . . .	1048
740: Blocking Chips . . . . .	1050
741: Velepin and Marketing . . . . .	1051
742: XOR, Tree, and Queries . . . . .	1053
743: Laboratory on Pluto . . . . .	1054
744: Anya's Simultaneous Exhibition . . . . .	1057
math . . . . .	1058
745: Construct Matrix . . . . .	1058
746: Multiple Lamps . . . . .	1060
747: Trees and XOR Queries Again . . . . .	1061

748: Brukhovich and Exams . . . . .	1063
749: Bored Bakry . . . . .	1064
750: Ideal Farm . . . . .	1066
751: Cutting Rectangle . . . . .	1067
752: Array Equalizer . . . . .	1068
753: Lihmuf Balling . . . . .	1070
754: Ina of the Mountain . . . . .	1071
755: The Boss's Identity . . . . .	1072
756: Igor and Interesting Numbers . . . . .	1074
757: Vika and Wiki . . . . .	1076
758: Vika and Stone Skipping . . . . .	1077
759: Min Cost Permutation (Easy Version) . . . . .	1079
760: Boxes and Balls . . . . .	1081
761: Lottery . . . . .	1083
762: Twin Clusters . . . . .	1084
763: Typewriter . . . . .	1086
764: Count Supersequences . . . . .	1088
misc . . . . .	1090
765: Hypercatapult Commute . . . . .	1090
766: Matrix Problem . . . . .	1092
767: Maximize Mex . . . . .	1094
768: Lexichromatography . . . . .	1096
769: Weights . . . . .	1098
770: BerDonalds . . . . .	1099
771: Opening Portals . . . . .	1100
772: Number Challenge . . . . .	1102
773: Similar Polynomials . . . . .	1103
774: Random Walk . . . . .	1105
775: Emperor's Problem . . . . .	1107
776: Number Table . . . . .	1112
777: Interesting Sequence . . . . .	1114
778: The Great Marathon . . . . .	1115
779: Timber . . . . .	1118
780: Testing . . . . .	1119
781: Inverse Function . . . . .	1122
782: Tram . . . . .	1125
783: Helper . . . . .	1127
784: Hide-and-Seek . . . . .	1130

785: TV Game . . . . .	1139
786: Quarrel . . . . .	1141
787: Tickets . . . . .	1142
788: XOR Counting . . . . .	1143
789: Broken robot . . . . .	1145
790: Tree . . . . .	1146
791: Tetragon . . . . .	1147
792: Traveling Graph . . . . .	1152
793: Notepad . . . . .	1154
794: Balance . . . . .	1155
red . . . . .	1158
combinatorics . . . . .	1158
795: Last Number . . . . .	1158
796: Festival Organization . . . . .	1159
797: Graph Coloring (easy version) . . . . .	1163
798: Graph Coloring (hard version) . . . . .	1165
799: Majority . . . . .	1167
800: List Generation . . . . .	1168
801: Antifibonacci Cut . . . . .	1170
data structures . . . . .	1172
802: Palindromic Problem . . . . .	1172
803: Local Deletions . . . . .	1174
804: Leha and security system . . . . .	1177
805: BRT Contract . . . . .	1178
806: Last Man Standing . . . . .	1180
807: ...Wait for it... . . . . .	1182
808: XOR Partition . . . . .	1184
809: Two Centroids . . . . .	1186
810: LuoTianyi and the Function . . . . .	1187
811: Cyclical Quest . . . . .	1189
812: k-Maximum Subsequence Sum . . . . .	1191
813: Tricky and Clever Password . . . . .	1192
814: Li Hua and Path . . . . .	1195
815: Points . . . . .	1196
816: Holes . . . . .	1198
817: M-tree . . . . .	1199
818: Gasoline prices . . . . .	1201
819: Magician and Pigs (Hard Version) . . . . .	1205

820: Colorful Tree Again . . . . .	1207
821: Weighed Tree Radius . . . . .	1210
dp . . . . .	1212
822: Xenia and String Problem . . . . .	1212
823: Difficult Mountain . . . . .	1214
825: Elevators of Tamem . . . . .	1216
826: Jackets and Packets . . . . .	1218
827: e-Government . . . . .	1220
828: Miriany and Matchstick . . . . .	1222
829: Two Subsequences . . . . .	1224
830: Mex Tree . . . . .	1225
831: Sequence Transformation . . . . .	1227
832: Misha and Apples . . . . .	1229
833: Forward, march! . . . . .	1231
834: Minibuses on Venus (hard version) . . . . .	1233
835: Prediction . . . . .	1235
836: R3D3s Summer Adventure . . . . .	1236
837: Another n-dimensional chocolate bar . . . . .	1238
838: Halve or Subtract . . . . .	1240
839: Serval and Brain Power . . . . .	1241
840: Parmigiana With Seafood . . . . .	1243
841: Edge Queries . . . . .	1245
greedy . . . . .	1247
842: Group Division . . . . .	1247
843: Alice and Recoloring 2 . . . . .	1249
844: Most Different Tree . . . . .	1251
845: Flower-like Pseudotree . . . . .	1253
846: Three Swaps . . . . .	1256
847: GCD Master (hard version) . . . . .	1258
848: GCD Master (easy version) . . . . .	1260
849: Teamwork . . . . .	1262
850: Wonderful Jump . . . . .	1264
math . . . . .	1265
851: Matrix Rank (Hard Version) . . . . .	1265
852: Matrix Rank (Easy Version) . . . . .	1267
853: Split . . . . .	1269
854: Hard Design . . . . .	1271
855: Ksusha and Square . . . . .	1273

856: Lazy Numbers . . . . .	1275
857: Swaps . . . . .	1277
858: Nephren Runs a Cinema . . . . .	1278
859: Lust . . . . .	1281
860: Evaluate RBS . . . . .	1283
861: Mod Mod Mod . . . . .	1289
862: PermuTree (hard version) . . . . .	1290
863: Game Bundles . . . . .	1291
864: Triangle Platinum? . . . . .	1294
865: Tree Weights . . . . .	1297
866: Min Cost Permutation (Hard Version) . . . . .	1299
867: Swimmers in the Pool . . . . .	1301
868: Tenzing and Random Operations . . . . .	1305
869: Tenzing and Random Real Numbers . . . . .	1306
870: Doctor's Brown Hypothesis . . . . .	1308
misc . . . . .	1310
871: Pumping Lemma . . . . .	1310
872: Landscaping . . . . .	1312
873: Symmetric Projections . . . . .	1319
874: Monocarp and a Strategic Game . . . . .	1321
875: Stuck Conveyor . . . . .	1326
876: Spider . . . . .	1329
877: Fading into Fog . . . . .	1332
878: Graph Game . . . . .	1334
879: Toy Machine . . . . .	1336
880: Petya and Rectangle . . . . .	1337
881: Edge coloring of bipartite graph . . . . .	1341
882: Cut Length . . . . .	1342
883: Parquet Re-laying . . . . .	1348
884: DravDe saves the world . . . . .	1351
885: Palisection . . . . .	1357
886: Communication Towers . . . . .	1359
887: Approximate Diameter . . . . .	1361
888: Strange Triples . . . . .	1363
889: Removal Sequences . . . . .	1366
890: Hero to Zero . . . . .	1367
891: Infinite Chess . . . . .	1369
892: The Game of the Century . . . . .	1373

893: MCF . . . . .	1374
894: Doremy's Perfect DS Class (Medium Version) . . . . .	1376
895: Doremy's Perfect DS Class (Easy Version) . . . . .	1378
896: Decent Division . . . . .	1379
898: Location . . . . .	1381
899: Kazaee . . . . .	1384
<b>trees . . . . .</b>	<b>1385</b>
900: Michael and Hotel . . . . .	1385
901: Baldman and the military . . . . .	1387
902: Bracket Insertion . . . . .	1389
903: Distance to the Path . . . . .	1391
<b>black . . . . .</b>	<b>1392</b>
<b>constructive algorithms . . . . .</b>	<b>1392</b>
904: Minimum Segments . . . . .	1392
905: Good Colorings . . . . .	1395
906: Two Permutations (Hard Version) . . . . .	1396
907: Magic Square . . . . .	1399
908: Bus Routes . . . . .	1404
909: OH NO1 (-2-3-4) . . . . .	1406
910: Maximum Permutation . . . . .	1408
911: Koxia, Mahiru and Winter Festival . . . . .	1409
912: Diverse Coloring . . . . .	1413
913: Unequal Adjacent Elements . . . . .	1416
914: Dangerous Laser Power . . . . .	1418
<b>data structures . . . . .</b>	<b>1420</b>
915: Parallel Swaps Sort . . . . .	1420
916: Diamond Theft . . . . .	1422
917: Clubstep . . . . .	1424
918: Standard Graph Problem . . . . .	1426
919: MEXanization . . . . .	1429
920: Freak Joker Process . . . . .	1431
921: Welcome home, Chtholly . . . . .	1436
922: Rivalries . . . . .	1439
923: LuoTianyi and Cartridge . . . . .	1443
924: A task for substrings . . . . .	1447
925: Iron Man . . . . .	1449
926: Flow Control . . . . .	1452
927: Treasure Hunt . . . . .	1454

928: Codeforces Scoreboard . . . . .	1456
929: Two Subtrees . . . . .	1458
930: Doremy's Paint 2 . . . . .	1461
931: MEX Tree Manipulation . . . . .	1464
dp . . . . .	1467
932: Optimizations From Chelsu . . . . .	1467
933: Ball-Stackable . . . . .	1469
934: Jellyfish and Incription . . . . .	1472
935: Min-Sum-Max . . . . .	1476
936: Redundant Routes . . . . .	1479
937: Goldberg Machine 3 . . . . .	1483
938: Asterism Stream . . . . .	1487
939: Sloth . . . . .	1488
940: Panda Meetups . . . . .	1490
941: Old Mobile . . . . .	1494
942: Zombies . . . . .	1496
943: Deja Vu . . . . .	1499
944: Willy-nilly, Crack, Into Release! . . . . .	1501
945: Bosco and Particle . . . . .	1504
946: The Maximum Prefix . . . . .	1507
947: Code Lock . . . . .	1509
948: Tree Cutting . . . . .	1512
949: Crossing the Railways . . . . .	1514
950: Segment Covering . . . . .	1519
951: Infinite Game . . . . .	1521
greedy . . . . .	1524
952: Half-sum . . . . .	1524
953: N Machines . . . . .	1526
math . . . . .	1528
954: Fugitive Frenzy . . . . .	1528
955: Parallel Universes (Hard) . . . . .	1530
956: Survival of the Weakest (hard version) . . . . .	1533
957: Window Signals (easy version) . . . . .	1535
958: Koxia and Sequence . . . . .	1538
959: Koxia and Bracket . . . . .	1539
960: Anti-median (Hard Version) . . . . .	1542
961: Anti-median (Easy Version) . . . . .	1545

misc . . . . .	1549
962: Indefinite Clownfish . . . . .	1549
963: Minimums or Medians . . . . .	1552
964: Olympic Team Building . . . . .	1554
965: Doremy's Perfect DS Class (Hard Version) . . . . .	1556
966: Minecraft Series . . . . .	1559
967: Joking (Hard Version) . . . . .	1562
trees . . . . .	1564
968: Roads in E City . . . . .	1564
Appendix: some common template code . . . . .	1567
MInt and MLong . . . . .	1567
Comb . . . . .	1569
DSU . . . . .	1570
Fenwick . . . . .	1571
HLD . . . . .	1572
LazySegmentTree . . . . .	1574

Problems are written by [Codeforces](#) writers. Codeforces platform is run by [MikeMirzayanov](#). Submissions are by [jiangly](#), and the book is collated by [2.7](#).

## Introduction

There are many good players at codeforces. Like many people, I read their submissions for inspirations and practice helps. Among them, jiangly stands out for having one of the most readable code.

Benq and Um\_nik use a large template, and tourist also fall in that category. Jiangly, however, only copies templates when needed, and generally writes clean code. Personally, I find it much easier to learn from shorter, more-readable submissions. I first had the idea for this project in Jan21, when jiangly just became an LGM, and it is great to see jiangly climb to be one of the best players. While I don't know about the history of the jiangly fan club, perhaps that is why so many of us are there.

I took the first 50 pages of jiangly's submissions, combined them with tags / other information, and put them together as a large problem set. I suspect it is unnecessary to try all the problems at a level when trying to improve, given the sheer quantity of problems here.

The difficulty ranges are divided into the following, roughly matching the rank colors. Not including 3000+ because of a coding bug.

1. Sub-1200 (Gray)
2. 1250-1600 (Green)

3. 1650-1900 (Blue)
4. 1950-2100 (Violet)
5. 2150-2300 (Orange)
6. 2350-2600 (Rose)
7. 2650-3000 (Red)

For each range, I sorted the tags by frequency, and selected the tags with the most frequent appearances. I chose these until the tags account for at least 70% of all tags by frequency. For each I currently included the link to the problem and a link to jiangly's solution. At the end of each section I also included an assortment of 20 problems, randomly selected from the remaining ones.

**What might make this a good selection of problems?** I think there are two common approaches to practicing problems, namely solving contests & solving handout problems. Both have their pros and cons.

1. Solving contests gets you an *accurate* sample of problems you will see in contexts, but does not give you the right level. A 2k-rated player will encounter both 3k-rated and 800-rated problems in a typical contest, and neither is very educational.
2. Handout problems can be tuned to be the right level. However, they often focus on interesting, but infrequent topics. You will find many exotic data structures browsing through Codeforces. The less flashy techniques (e.g. Greedy and DP), however, make up the majority of the problems.

I hope the selected problems can get the *best of both worlds*. The chapters are divided by difficulty, so that one can find problems that are at the right level. The sections of a chapter are selected using the most common tags found on Codeforces, so they should be teaching the most important techniques. Of course, I didn't include every problem of a given tag, since solving 10-12 of each is already plenty of work.

**Why do these here, instead of from the problemset page?** I think solving from the problemset page is a great approach as well, but this might still be an improvement. 1. It is easier to schedule practice with a book. One can say "solve problems 430-440 from the blue book" and you will easily know where to find them. 2. Each of these problems contain a good selected solution. The solution here is answering "how would a black-level player solve this under time pressure?", and are perhaps more practical than the curated solutions. They will not be perfect, but they should be good. 3. It saves you the hassle of finding problems.

**What is up with the struct's?** There are some code that are frequently repeated, such as that for modular arithmetic. Including them tends to make the code look less neat, so I took them out. The common ones are found in the appendix.

**What does this book not have?** Of course, given I compiled this using a program, there are a few limitations. 1. Explanation of the concepts. You can look around Codeforces, wikipedia, or ask an AI to

explain it. 2. Quality control. If a problem is way too easy / brutal for its difficulty, or has nothing to do with the tag, feel free to comment it.

**Finally, an observation:** if you can solve all the problems in this book without looking at hints, in (30 min for hard ones, 5min for easy ones), your tournament performances should be almost indistinguishable from his. There are some notable differences, of course: 1. You know that some black-level player, namely jiangly, was able to solve this problem. Jiangly may not have known that. 2. You know the tags that the problem received.

Now, enjoy the problems!

## gray

### brute force

#### 1: Erase First or Second Letter

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a string  $s$  of length  $n$ . Let's define two operations you can apply on the string:

remove the first character of the string;

remove the second character of the string.

Your task is to find the number of distinct non-empty strings that can be generated by applying the given operations on the initial string any number of times (possibly zero), in any order.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     string s;
8     cin >> s;
9     array<int, 26> cnt{};
10    int dist = 0;
11    int ans = 0;
12    for (auto c : s) {
13        dist += cnt[c - 'a']++ == 0;
14        ans += dist;
15    }
16    cout << ans << "\n";
17 }
18 int main() {
19     ios::sync_with_stdio(false);
20     cin.tie(nullptr);
21     int t;
22     cin >> t;
23     while (t--) {
24         solve();
25     }
26     return 0;
27 }
```

## 2: Easy As ABC

- Time limit: 1 second
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

You are playing a word puzzle. The puzzle starts with a 3 by 3 grid, where each cell contains either the letter A, B, or C.

The goal of this puzzle is to find the lexicographically smallest possible word of length 3. The word can be formed by choosing three different cells where the cell containing the first letter is adjacent to the cell containing the second letter, and the cell containing the second letter is adjacent to the cell containing the third letter.

Two cells are adjacent to each other if they share a border or a corner, as shown in the following illustration. Formally, if  $(r, c)$  denotes the cell in the  $r$ -th row and  $c$ -th column, then cell  $(r, c)$  is adjacent to cell  $(r, c + 1)$ ,  $(r - 1, c + 1)$ ,  $(r - 1, c)$ ,  $(r - 1, c - 1)$ ,  $(r, c - 1)$ ,  $(r + 1, c - 1)$ ,  $(r + 1, c)$ , and  $(r + 1, c + 1)$ .

Determine the lexicographically smallest possible word of length 3 that you can find within the grid.

A string  $s$  of length  $n$  is lexicographically smaller than string  $t$  of the same length if there exists an integer  $1 \leq i \leq n$  such that  $s_j = t_j$  for all  $1 \leq j < i$ , and  $s_i < t_i$  in alphabetical order. The following illustration shows some examples on some grids and their the lexicographically smallest possible word of length 3 that you can find within the grids.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     array<string, 3> s;
8     for (int i = 0; i < 3; i++) {
9         cin >> s[i];
10    }
11    array<array<int, 3>, 3> vis{};
12    string t;
13    string ans = "CCC";
14    auto dfs = [&](auto self, int i, int x, int y) -> void {
15        if (vis[x][y]) {
16            return;
17        }
18        t += s[x][y];
19        vis[x][y] = 1;
20        if (i == 2) {

```

```

21         ans = min(ans, t);
22     } else {
23         for (int nx = 0; nx < 3; nx++) {
24             for (int ny = 0; ny < 3; ny++) {
25                 if (abs(x - nx) <= 1 && abs(y - ny) <= 1) {
26                     self(self, i + 1, nx, ny);
27                 }
28             }
29         }
30     }
31     t.pop_back();
32     vis[x][y] = 0;
33 };
34 for (int x = 0; x < 3; x++) {
35     for (int y = 0; y < 3; y++) {
36         dfs(dfs, 0, x, y);
37     }
38 }
39 cout << ans << "\n";
40 return 0;
41 }

```

### 3: Don't Try to Count

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Given a string  $x$  of length  $n$  and a string  $s$  of length  $m$  ( $n \cdot m \leq 25$ ), consisting of lowercase Latin letters, you can apply any number of operations to the string  $x$ .

In one operation, you append the current value of  $x$  to the end of the string  $x$ . Note that the value of  $x$  will change after this.

For example, if  $x = \text{"aba"}$ , then after applying operations,  $x$  will change as follows:  $\text{"aba"} \rightarrow \text{"abaaba"}$   $\rightarrow \text{"abaabaabaaba"}$ .

After what minimum number of operations  $s$  will appear in  $x$  as a substring? A substring of a string is defined as a contiguous segment of it.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;

```

```

7     string x, s;
8     cin >> x >> s;
9     int ans = 0;
10    while (x.find(s) == -1 && (ans == 0 || x.size() < 2 * m)) {
11        x = x + x;
12        ans++;
13    }
14    if (x.find(s) == -1) {
15        cout << -1 << "\n";
16    } else {
17        cout << ans << "\n";
18    }
19 }
20 int main() {
21     ios::sync_with_stdio(false);
22     cin.tie(nullptr);
23     int t;
24     cin >> t;
25     while (t--) {
26         solve();
27     }
28     return 0;
29 }
```

#### 4: Grandma Capa Knits a Scarf

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Grandma Capa has decided to knit a scarf and asked Grandpa Sher to make a pattern for it, a pattern is a string consisting of lowercase English letters. Grandpa Sher wrote a string  $s$  of length  $n$ .

Grandma Capa wants to knit a beautiful scarf, and in her opinion, a beautiful scarf can only be knit from a string that is a palindrome. She wants to change the pattern written by Grandpa Sher, but to avoid offending him, she will choose one lowercase English letter and erase some (at her choice, possibly none or all) occurrences of that letter in string  $s$ .

She also wants to minimize the number of erased symbols from the pattern. Please help her and find the minimum number of symbols she can erase to make string  $s$  a palindrome, or tell her that it's impossible. Notice that she can only erase symbols equal to the one letter she chose.

A string is a palindrome if it is the same from the left to the right and from the right to the left. For example, the strings 'kek', 'abacaba', 'r' and 'papicipap' are palindromes, while the strings 'abb' and 'iq' are not.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     string s;
8     cin >> s;
9     int ans = n + 1;
10    for (char x = 'a'; x <= 'z'; x++) {
11        int l = 0, r = n - 1;
12        int res = 0;
13        while (l < r) {
14            if (s[l] == s[r]) {
15                l++, r--;
16            } else if (s[l] == x) {
17                l++, res++;
18            } else if (s[r] == x) {
19                r--, res++;
20            } else {
21                res = n + 1;
22                break;
23            }
24        }
25        ans = min(ans, res);
26    }
27    if (ans == n + 1) {
28        ans = -1;
29    }
30    cout << ans << "\n";
31 }
32 int main() {
33     ios::sync_with_stdio(false);
34     cin.tie(nullptr);
35     int t;
36     cin >> t;
37     while (t--) {
38         solve();
39     }
40     return 0;
41 }
```

## 5: Two Vessels

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have two vessels with water. The first vessel contains  $a$  grams of water, and the second vessel contains  $b$  grams of water. Both vessels are very large and can hold any amount of water.

You also have an empty cup that can hold up to  $c$  grams of water.

In one move, you can scoop up to  $c$  grams of water from any vessel and pour it into the other vessel. Note that the mass of water poured in one move does not have to be an integer.

What is the minimum number of moves required to make the masses of water in the vessels equal? Note that you cannot perform any actions other than the described moves.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int a, b, c;
6     cin >> a >> b >> c;
7     int d = abs(a - b);
8     int ans = (d + 2 * c - 1) / (2 * c);
9     cout << ans << "\n";
10 }
11 int main() {
12     ios::sync_with_stdio(false);
13     cin.tie(nullptr);
14     int t;
15     cin >> t;
16     while (t--) {
17         solve();
18     }
19     return 0;
20 }
```

## 6: Download More RAM

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Did you know you can download more RAM? There is a shop with  $n$  different pieces of software that increase your RAM. The  $i$ -th RAM increasing software takes  $a_i$  GB of memory to run (temporarily, once the program is done running, you get the RAM back), and gives you an additional  $b_i$  GB of RAM (permanently). Each software can only be used once. Your PC currently has  $k$  GB of RAM.

Note that you can't use a RAM-increasing software if it takes more GB of RAM to use than what you currently have.

Since RAM is the most important thing in the world, you wonder, what is the maximum possible amount of RAM achievable?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> a(n), b(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    for (int i = 0; i < n; i++) {
12        cin >> b[i];
13    }
14    vector<int> p(n);
15    iota(p.begin(), p.end(), 0);
16    sort(p.begin(), p.end(),
17          [&](int i, int j) {
18              return a[i] < a[j];
19          });
20    for (auto i : p) {
21        if (a[i] <= k) {
22            k += b[i];
23        }
24    }
25    cout << k << "\n";
26 }
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(nullptr);
30     int t;
31     cin >> t;
32     while (t--) {
33         solve();
34     }
35     return 0;
36 }
```

## 7: Cirno's Perfect Bitmasks Classroom

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Cirno's perfect bitmasks classroom has just started!

Cirno gave her students a positive integer  $x$ . As an assignment, her students need to find the minimum positive integer  $y$ , which satisfies the following two conditions:

$$x \text{ and } y > 0$$

$$x \text{ and } y > 0$$

Where and is the bitwise AND operation, and xor is the bitwise XOR operation.

Among the students was Mystia, who was truly baffled by all these new operators. Please help her!

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int x;
6     cin >> x;
7     if (x & (x - 1)) {
8         cout << (x & -x) << "\n";
9     } else if (x == 1) {
10        cout << 3 << "\n";
11    } else {
12        cout << x + 1 << "\n";
13    }
14 }
15 int main() {
16     ios::sync_with_stdio(false);
17     cin.tie(nullptr);
18     int t;
19     cin >> t;
20     while (t--) {
21         solve();
22     }
23     return 0;
24 }
```

## 8: Pizza Separation

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Students Vasya and Petya are studying at the BSU (Byteland State University). At one of the breaks they decided to order a pizza. In this problem pizza is a circle of some radius. The pizza was delivered already cut into n pieces. The i-th piece is a sector of angle equal to  $a_i$ . Vasya and Petya want to divide all pieces of pizza into two continuous sectors in such way that the difference between angles of these sectors is minimal. Sector angle is sum of angles of all pieces in it. Pay attention, that one of sectors can be empty.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    int ans = 360;
14    for (int l = 0; l < n; l++) {
15        int sum = 0;
16        for (int r = l; r < n; r++) {
17            sum += a[r];
18            ans = min(ans, abs(2 * sum - 360));
19        }
20    }
21    cout << ans << "\n";
22    return 0;
23 }
```

## 9: Beautiful Divisors

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Recently Luba learned about a special kind of numbers that she calls beautiful numbers. The number is called beautiful iff its binary representation consists of  $k + 1$  consecutive ones, and then  $k$  consecutive zeroes.

Some examples of beautiful numbers:

12 (110);

1102 (610);

11110002 (12010);

1111100002 (49610).

More formally, the number is beautiful iff there exists some positive integer  $k$  such that the number is equal to  $(2k - 1) * (2k - 1)$ .

Luba has got an integer number  $n$ , and she wants to find its greatest beautiful divisor. Help her to find it!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     int ans = 1;
10    for (int x = 1; ; x *= 2) {
11        int v = (2 * x - 1) * x;
12        if (v > n) {
13            break;
14        }
15        if (n % v == 0) {
16            ans = v;
17        }
18    }
19    cout << ans << "\n";
20    return 0;
21 }
```

## 10: Fancy Coins

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp is going to make a purchase with cost of exactly  $m$  burles.

He has two types of coins, in the following quantities:

coins worth 1 burle:  $a_1$  regular coins and infinitely many fancy coins;

coins worth  $k$  burles:  $a_k$  regular coins and infinitely many fancy coins.

Monocarp wants to make his purchase in such a way that there's no change - the total worth of provided coins is exactly  $m$ . He can use both regular and fancy coins. However, he wants to spend as little fancy coins as possible.

What's the smallest total number of fancy coins he can use to make a purchase?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int m, k, a1, ak;
6     cin >> m >> k >> a1 >> ak;
7     int t = max(0, min(m / k, (m - a1 + k - 1) / k));
8     int ans = max(0, t - ak) + max(0, m - t * k - a1);
9     cout << ans << "\n";
10 }
11 int main() {
12     ios::sync_with_stdio(false);
13     cin.tie(nullptr);
14     int t;
15     cin >> t;
16     while (t--) {
17         solve();
18     }
19     return 0;
20 }
```

## 11: ACM ICPC

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In a small but very proud high school it was decided to win ACM ICPC. This goal requires to compose as many teams of three as possible, but since there were only 6 students who wished to participate, the decision was to build exactly two teams.

After practice competition, participant number  $i$  got a score of  $a_i$ . Team score is defined as sum of scores of its participants. High school management is interested if it's possible to build two teams with equal scores. Your task is to answer that question.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int a[6];
8     for (int i = 0; i < 6; i++) {
9         cin >> a[i];
10    }
11    for (int s = 0; s < (1 << 6); s++) {
12        if (__builtin_popcount(s) == 3) {
13            int bal = 0;
```

```

14         for (int i = 0; i < 6; i++) {
15             if (s >> i & 1) {
16                 bal += a[i];
17             } else {
18                 bal -= a[i];
19             }
20         }
21         if (bal == 0) {
22             cout << "YES\n";
23             return 0;
24         }
25     }
26 }
27 cout << "NO\n";
28 return 0;
29 }
```

## 12: Local Extrema

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$ . Some element of this array  $a_i$  is a local minimum iff it is strictly less than both of its neighbours (that is,  $a_i < a_{i-1}$  and  $a_i < a_{i+1}$ ). Also the element can be called local maximum iff it is strictly greater than its neighbours (that is,  $a_i > a_{i-1}$  and  $a_i > a_{i+1}$ ). Since  $a_1$  and  $a_n$  have only one neighbour each, they are neither local minima nor local maxima.

An element is called a local extremum iff it is either local maximum or local minimum. Your task is to calculate the number of local extrema in the given array.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    int ans = 0;
14    for (int i = 1; i < n - 1; i++) {
15        if (a[i] > a[i - 1] && a[i] > a[i + 1]) {
16            ans++;
17        }
18    }
19 }
```

```

17         }
18     if (a[i] < a[i - 1] && a[i] < a[i + 1]) {
19         ans++;
20     }
21 }
22 cout << ans << "\n";
23 return 0;
24 }
```

### 13: Another Permutation Problem

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Andrey is just starting to come up with problems, and it's difficult for him. That's why he came up with a strange problem about permutations<sup>†</sup> and asks you to solve it. Can you do it?

Let's call the cost of a permutation  $p$  of length  $n$  the value of the expression:

Find the maximum cost among all permutations of length  $n$ .

<sup>†</sup>A permutation of length  $n$  is an array consisting of  $n$  distinct integers from 1 to  $n$  in arbitrary order. For example, [2, 3, 1, 5, 4] is a permutation, but [1, 2, 2] is not a permutation (2 appears twice in the array), and [1, 3, 4] is also not a permutation ( $n = 3$  but there is 4 in the array).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 void solve() {
6     int n;
7     cin >> n;
8     int ans = 0;
9     for (int l = n * n; ; l--) {
10        DSU dsu(n + 1);
11        int sum = 0;
12        bool ok = true;
13        for (int i = n; i >= 1; i--) {
14            int x = dsu.find(min(n, l / i));
15            if (x == 0) {
16                ok = false;
17                break;
18            }
19            sum += i * x;
20            dsu.merge(x - 1, x);
21        }
22        if (!ok) {
23            break;
```

```

24         }
25         ans = max(ans, sum - l);
26     }
27     cout << ans << "\n";
28 }
29 int main() {
30     ios::sync_with_stdio(false);
31     cin.tie(nullptr);
32     int t;
33     cin >> t;
34     while (t--) {
35         solve();
36     }
37     return 0;
38 }
```

## 14: Fibonacchassis

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Ntarsis has received two integers  $n$  and  $k$  for his birthday. He wonders how many fibonacci-like sequences of length  $k$  can be formed with  $n$  as the  $k$ -th element of the sequence.

A sequence of non-decreasing non-negative integers is considered fibonacci-like if  $f_i = f_{i-1} + f_{i-2}$  for all  $i > 2$ , where  $f_i$  denotes the  $i$ -th element in the sequence. Note that  $f_1$  and  $f_2$  can be arbitrary.

For example, sequences such as  $[4, 5, 9, 14]$  and  $[0, 1, 1]$  are considered fibonacci-like sequences, while  $[0, 0, 0, 1, 1]$ ,  $[1, 2, 1, 3]$ , and  $[-1, -1, -2]$  are not: the first two do not always satisfy  $f_i = f_{i-1} + f_{i-2}$ , the latter does not satisfy that the elements are non-negative.

Impress Ntarsis by helping him with this task.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 vector<i64> f{1, 1};
5 void solve() {
6     int n, k;
7     cin >> n >> k;
8     k--;
9     if (k - 1 >= 44 || f[k - 1] > n) {
10         cout << 0 << "\n";
11     }
12 }
```

```

13     i64 a = f[k - 1], b = f[k];
14     i64 x = f[k + 1];
15     if (k % 2 == 0) {
16         x = -x;
17     }
18     x = n * x % b;
19     if (x < 0) {
20         x += b;
21     }
22     if (x * a > n) {
23         cout << 0 << "\n";
24         return;
25     }
26     cout << (n - x * a) / (a * b) + 1 << "\n";
27 }
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(nullptr);
31     for (int i = 2; i < 50; i++) {
32         f.push_back(f[i - 2] + f[i - 1]);
33     }
34     int t;
35     cin >> t;
36     while (t--) {
37         solve();
38     }
39     return 0;
40 }
```

## 15: Desorting

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Call an array  $a$  of length  $n$  sorted if  $a_1 \leq a_2 \leq \dots \leq a_{n-1} \leq a_n$ .

Ntarsis has an array  $a$  of length  $n$ .

He is allowed to perform one type of operation on it (zero or more times):

Choose an index  $i$  ( $1 \leq i \leq n - 1$ ).

Add 1 to  $a_1, a_2, \dots, a_i$ .

Subtract 1 from  $a_{i+1}, a_{i+2}, \dots, a_n$ .

The values of  $a$  can be negative after an operation.

Determine the minimum operations needed to make  $a$  not sorted.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    int ans = 1E9;
12    for (int i = 1; i < n; i++) {
13        ans = min(ans, max(0, (a[i] - a[i - 1] + 2) / 2));
14    }
15    cout << ans << "\n";
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

## 16: Longest Divisors Interval

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Given a positive integer  $n$ , find the maximum size of an interval  $[l, r]$  of positive integers such that, for every  $i$  in the interval (i.e.,  $l \leq i \leq r$ ),  $n$  is a multiple of  $i$ .

Given two integers  $l \leq r$ , the size of the interval  $[l, r]$  is  $r - l + 1$  (i.e., it coincides with the number of integers belonging to the interval).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     i64 n;
6     cin >> n;
7     int ans = 1;
8     while (n % (ans + 1) == 0) {
9         ans++;
10    }
```

```

10      }
11      cout << ans << "\n";
12  }
13 int main() {
14     ios::sync_with_stdio(false);
15     cin.tie(nullptr);
16     int t;
17     cin >> t;
18     while (t--) {
19         solve();
20     }
21     return 0;
22 }
```

## 17: Balanced Round

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are the author of a Codeforces round and have prepared  $n$  problems you are going to set, problem  $i$  having difficulty  $a_i$ . You will do the following process:

remove some (possibly zero) problems from the list;

rearrange the remaining problems in any order you wish.

A round is considered balanced if and only if the absolute difference between the difficulty of any two consecutive problems is at most  $k$  (less or equal than  $k$ ).

What is the minimum number of problems you have to remove so that an arrangement of problems is balanced?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    int ans = 0;
12    sort(a.begin(), a.end());
13    int lst = 0;
14    for (int i = 1; i <= n; i++) {
15        if (i == n || a[i] - a[i - 1] > k) {
16            ans = max(ans, i - lst);
```

```

17         lst = i;
18     }
19 }
20 cout << n - ans << "\n";
21 }
22 int main() {
23     ios::sync_with_stdio(false);
24     cin.tie(nullptr);
25     int t;
26     cin >> t;
27     while (t--) {
28         solve();
29     }
30     return 0;
31 }
```

## 18: Rudolph and Tic-Tac-Toe

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Rudolph invented the game of tic-tac-toe for three players. It has classic rules, except for the third player who plays with pluses. Rudolf has a  $3 \times 3$  field - the result of the completed game. Each field cell contains either a cross, or a nought, or a plus sign, or nothing. The game is won by the player who makes a horizontal, vertical or diagonal row of 3's of their symbols.

Rudolph wants to find the result of the game. Either exactly one of the three players won or it ended in a draw. It is guaranteed that multiple players cannot win at the same time.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     string s[3];
6     for (int i = 0; i < 3; i++) {
7         cin >> s[i];
8     }
9     for (int i = 0; i < 3; i++) {
10         if (s[i][0] == s[i][1] && s[i][1] == s[i][2] && s[i][0] != '.') {
11             cout << s[i][0] << "\n";
12             return;
13         }
14         if (s[0][i] == s[1][i] && s[1][i] == s[2][i] && s[0][i] != '.') {
15             cout << s[0][i] << "\n";
16             return;
17         }
18     }
19     if (s[0][0] == s[1][1] && s[1][1] == s[2][2] && s[0][0] != '.') {
```

```

20         cout << s[0][0] << "\n";
21     return;
22 }
23 if (s[2][0] == s[1][1] && s[1][1] == s[0][2] && s[2][0] != '.') {
24     cout << s[2][0] << "\n";
25     return;
26 }
27 cout << "DRAW\n";
28 }
29 int main() {
30     ios::sync_with_stdio(false);
31     cin.tie(nullptr);
32     int t;
33     cin >> t;
34     while (t--) {
35         solve();
36     }
37     return 0;
38 }
```

## 19: Trust Nobody

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There is a group of  $n$  people. Some of them might be liars, who always tell lies. Other people always tell the truth. The  $i$ -th person says “There are at least  $l_i$  liars amongst us”. Determine if what people are saying is contradictory, or if it is possible. If it is possible, output the number of liars in the group. If there are multiple possible answers, output any one of them.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> l(n);
8     for (int i = 0; i < n; i++) {
9         cin >> l[i];
10    }
11    for (int x = 0; x <= n; x++) {
12        int cnt = 0;
13        for (int i = 0; i < n; i++) {
14            cnt += x < l[i];
15        }
16        if (cnt == x) {
17            cout << x << "\n";
18    }
19 }
```

```

18         return;
19     }
20 }
21 cout << -1 << "\n";
22 }
23 int main() {
24     ios::sync_with_stdio(false);
25     cin.tie(nullptr);
26     int t;
27     cin >> t;
28     while (t--) {
29         solve();
30     }
31     return 0;
32 }
```

## 20: Gold Rush

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Initially you have a single pile with  $n$  gold nuggets. In an operation you can do the following:

Take any pile and split it into two piles, so that one of the resulting piles has exactly twice as many gold nuggets as the other. (All piles should have an integer number of nuggets.)

One possible move is to take a pile of size 6 and split it into piles of sizes 2 and 4, which is valid since 4 is twice as large as 2.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     int t = 0;
8     while (n % 3 == 0 && m % 3 == 0) {
9         n /= 3;
10        m /= 3;
11    }
12    while (n % 3 == 0) {
13        n /= 3;
14        t++;
15    }
16    if (m % n != 0) {
17        cout << "NO\n";
18        return;
19    }
20    m /= n;
```

```

21     if ((m & (m - 1)) == 0 && m <= (1 << t)) {
22         cout << "YES\n";
23     } else {
24         cout << "NO\n";
25     }
26 }
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(nullptr);
30     int t;
31     cin >> t;
32     while (t--) {
33         solve();
34     }
35     return 0;
36 }
```

## constructive algorithms

### 21: Binary Imbalance

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a string  $s$ , consisting only of characters ‘0’ and/or ‘1’.

In one operation, you choose a position  $i$  from 1 to  $|s| - 1$ , where  $|s|$  is the current length of string  $s$ . Then you insert a character between the  $i$ -th and the  $(i + 1)$ -st characters of  $s$ . If  $s_i = s_{i+1}$ , you insert ‘1’. If  $s_i \neq s_{i+1}$ , you insert ‘0’.

Is it possible to make the number of zeroes in the string strictly greater than the number of ones, using any number of operations (possibly, none)?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     string s;
8     cin >> s;
9     if (count(s.begin(), s.end(), '1') == n) {
10        cout << "NO\n";
11    } else {
12        cout << "YES\n";
```

```

13     }
14 }
15 int main() {
16     ios::sync_with_stdio(false);
17     cin.tie(nullptr);
18     int t;
19     cin >> t;
20     while (t--) {
21         solve();
22     }
23     return 0;
24 }
```

## 22: Sorting with Twos

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array of integers  $a_1, a_2, \dots, a_n$ . In one operation, you do the following:

Choose a non-negative integer  $m$ , such that  $2^m \leq n$ .

Subtract 1 from  $a_i$  for all integers  $i$ , such that  $1 \leq i \leq 2^m$ .

Can you sort the array in non-decreasing order by performing some number (possibly zero) of operations?

An array is considered non-decreasing if  $a_i \leq a_{i+1}$  for all integers  $i$  such that  $1 \leq i \leq n - 1$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    for (int i = 1; i < n; i++) {
12        if ((i & (i - 1)) != 0 && a[i] < a[i - 1]) {
13            cout << "NO\n";
14            return;
15        }
16    }
17    cout << "YES\n";
18 }
19 int main() {
20     ios::sync_with_stdio(false);
```

```

21     cin.tie(nullptr);
22     int t;
23     cin >> t;
24     while (t--) {
25         solve();
26     }
27     return 0;
28 }
```

### 23: Chips on the Board

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a board of size  $n \times n$  ( $n$  rows and  $n$  columns) and two arrays of positive integers  $a$  and  $b$  of size  $n$ .

Your task is to place the chips on this board so that the following condition is satisfied for every cell  $(i, j)$ :

there exists at least one chip in the same column or in the same row as the cell  $(i, j)$ . I. e. there exists a cell  $(x, y)$  such that there is a chip in that cell, and either  $x = i$  or  $y = j$  (or both).

The cost of putting a chip in the cell  $(i, j)$  is equal to  $a_i + b_j$ .

For example, for  $n = 3$ ,  $a = [1, 4, 1]$  and  $b = [3, 2, 2]$ . One of the possible chip placements is as follows:

The total cost of that placement is  $(1 + 3) + (1 + 2) + (1 + 2) = 10$ .

Calculate the minimum possible total cost of putting chips according to the rules above.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     i64 sa = 0;
8     int mina = 1E9;
9     for (int i = 0; i < n; i++) {
10         int a;
11         cin >> a;
12         sa += a;
```

```

13         mina = min(mina, a);
14     }
15     i64 sb = 0;
16     int minb = 1E9;
17     for (int i = 0; i < n; i++) {
18         int b;
19         cin >> b;
20         sb += b;
21         minb = min(minb, b);
22     }
23     i64 ans = min(sa + 1LL * n * minb, sb + 1LL * n * mina);
24     cout << ans << "\n";
25 }
26 int main() {
27     ios::sync_with_stdio(false);
28     cin.tie(nullptr);
29     int t;
30     cin >> t;
31     while (t--) {
32         solve();
33     }
34     return 0;
35 }
```

## 24: MEXanized Array

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given three non-negative integers  $n$ ,  $k$ , and  $x$ . Find the maximum possible sum of elements in an array consisting of non-negative integers, which has  $n$  elements, its MEX is equal to  $k$ , and all its elements do not exceed  $x$ . If such an array does not exist, output  $-1$ .

The MEX (minimum excluded) of an array is the smallest non-negative integer that does not belong to the array. For instance:

The MEX of  $[2, 2, 1]$  is 0, because 0 does not belong to the array.

The MEX of  $[3, 1, 0, 1]$  is 2, because 0 and 1 belong to the array, but 2 does not.

The MEX of  $[0, 3, 1, 2]$  is 4, because 0, 1, 2 and 3 belong to the array, but 4 does not.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
```

```

4 void solve() {
5     int n, k, x;
6     cin >> n >> k >> x;
7     if (k > n || k > x + 1) {
8         cout << -1 << "\n";
9     } else if (k == x) {
10        cout << k * (k - 1) / 2 + (n - k) * (k - 1) << "\n";
11    } else {
12        cout << k * (k - 1) / 2 + (n - k) * x << "\n";
13    }
14 }
15 int main() {
16     ios::sync_with_stdio(false);
17     cin.tie(nullptr);
18     int t;
19     cin >> t;
20     while (t--) {
21         solve();
22     }
23     return 0;
24 }
```

## 25: Make It Zero

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

During Zhongkao examination, Reycloer met an interesting problem, but he cannot come up with a solution immediately. Time is running out! Please help him.

Initially, you are given an array  $a$  consisting of  $n \geq 2$  integers, and you want to change all elements in it to 0.

In one operation, you select two indices  $l$  and  $r$  ( $1 \leq l \leq r \leq n$ ) and do the following:

Let  $s = a_l \oplus a_{l+1} \oplus \dots \oplus a_r$ , where  $\oplus$  denotes the bitwise XOR operation;

Then, for all  $l \leq i \leq r$ , replace  $a_i$  with  $s$ .

You can use the operation above in any order at most 8 times in total.

Find a sequence of operations, such that after performing the operations in order, all elements in  $a$  are equal to 0. It can be proven that the solution always exists.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
```

```

4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    cout << 5 << "\n";
12    cout << 1 << " " << n << "\n";
13    cout << 1 << " " << n << "\n";
14    cout << 1 << " " << n - 1 << "\n";
15    cout << n - 1 << " " << n << "\n";
16    cout << n - 1 << " " << n << "\n";
17 }
18 int main() {
19     ios::sync_with_stdio(false);
20     cin.tie(nullptr);
21     int t;
22     cin >> t;
23     while (t--) {
24         solve();
25     }
26     return 0;
27 }
```

## 26: XOR Palindromes

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a binary string  $s$  of length  $n$  (a string that consists only of 0 and 1). A number  $x$  is good if there exists a binary string  $l$  of length  $n$ , containing  $x$  ones, such that if each symbol  $s_i$  is replaced by  $s_i \oplus l_i$  (where  $\oplus$  denotes the bitwise XOR operation), then the string  $s$  becomes a palindrome.

You need to output a binary string  $t$  of length  $n + 1$ , where  $t_i$  ( $0 \leq i \leq n$ ) is equal to 1 if number  $i$  is good, and 0 otherwise.

A palindrome is a string that reads the same from left to right as from right to left. For example, 01010, 1111, 0110 are palindromes.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     string s;
8     cin >> s;
```

```

9     int l = 0, r = 0;
10    for (int i = 0; i < n - 1 - i; i++) {
11        if (s[i] == s[n - 1 - i]) {
12            r += 2;
13        } else {
14            l++, r++;
15        }
16    }
17    string ans(n + 1, '0');
18    for (int i = l; i <= r; i += 2) {
19        ans[i] = '1';
20        if (n % 2 == 1) {
21            ans[i + 1] = '1';
22        }
23    }
24    cout << ans << "\n";
25 }
26 int main() {
27     ios::sync_with_stdio(false);
28     cin.tie(nullptr);
29     int t;
30     cin >> t;
31     while (t--) {
32         solve();
33     }
34     return 0;
35 }
```

## 27: green\_gold\_dog, array and permutation

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

green\_gold\_dog has an array  $a$  of length  $n$ , and he wants to find a permutation  $b$  of length  $n$  such that the number of distinct numbers in the element-wise difference between array  $a$  and permutation  $b$  is maximized.

A permutation of length  $n$  is an array consisting of  $n$  distinct integers from 1 to  $n$  in any order. For example, [2, 3, 1, 5, 4] is a permutation, but [1, 2, 2] is not a permutation (as 2 appears twice in the array) and [1, 3, 4] is also not a permutation (as  $n = 3$ , but 4 appears in the array).

The element-wise difference between two arrays  $a$  and  $b$  of length  $n$  is an array  $c$  of length  $n$ , where  $c_i = a_i - b_i$  ( $1 \leq i \leq n$ ).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
```

```

4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<int> p(n);
12    iota(p.begin(), p.end(), 0);
13    sort(p.begin(), p.end(),
14        [&](int i, int j) {
15            return a[i] < a[j];
16        });
17    vector<int> b(n);
18    for (int i = 0; i < n; i++) {
19        b[p[i]] = n - i;
20    }
21    for (int i = 0; i < n; i++) {
22        cout << b[i] << " \n"[i == n - 1];
23    }
24 }
25 int main() {
26     ios::sync_with_stdio(false);
27     cin.tie(nullptr);
28     int t;
29     cin >> t;
30     while (t--) {
31         solve();
32     }
33     return 0;
34 }
```

## 28: Madoka and the Elegant Gift

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Madoka's father just reached 1 million subscribers on Mathub! So the website decided to send him a personalized award - The Mathhub's Bit Button!

The Bit Button is a rectangular table with  $n$  rows and  $m$  columns with 0 or 1 in each cell. After exploring the table Madoka found out that:

A subrectangle  $A$  is contained in a subrectangle  $B$  if there's no cell contained in  $A$  but not contained in  $B$ .

Two subrectangles intersect if there is a cell contained in both of them.

A subrectangle is called black if there's no cell with value 0 inside it.

A subrectangle is called nice if it's black and it's not contained in another black subrectangle.

The table is called elegant if there are no two nice intersecting subrectangles.

For example, in the first illustration the red subrectangle is nice, but in the second one it's not, because it's contained in the purple subrectangle.

Help Madoka to determine whether the table is elegant.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<string> s(n);
8     for (int i = 0; i < n; i++) {
9         cin >> s[i];
10    }
11    for (int i = 0; i < n - 1; i++) {
12        for (int j = 0; j < m - 1; j++) {
13            if (s[i][j] - '0' + s[i][j + 1] - '0' + s[i + 1][j] - '0' + s[i + 1][j + 1] - '0' == 3) {
14                cout << "NO\n";
15                return;
16            }
17        }
18    }
19    cout << "YES\n";
20 }
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     int t;
25     cin >> t;
26     while (t--) {
27         solve();
28     }
29     return 0;
30 }
```

## 29: Patchouli's Magical Talisman

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Patchouli is making a magical talisman. She initially has  $n$  magical tokens. Their magical power can be represented with positive integers  $a_1, a_2, \dots, a_n$ .

Patchouli may perform the following two operations on the tokens.

Fusion: Patchouli chooses two tokens, removes them, and creates a new token with magical power equal to the sum of the two chosen tokens.

Reduction: Patchouli chooses a token with an even value of magical power  $x$ , removes it and creates a new token with magical power equal to  $\frac{x}{2}$ .

Tokens are more effective when their magical powers are odd values. Please help Patchouli to find the minimum number of operations she needs to make magical powers of all tokens odd values.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     int even = 0;
8     int l = 30;
9     for (int i = 0; i < n; i++) {
10         int a;
11         cin >> a;
12         even += (a % 2 == 0);
13         l = min(l, __builtin_ctz(a));
14     }
15     int ans = l == 0 ? even : n - 1 + l;
16     cout << ans << "\n";
17 }
18 int main() {
19     ios::sync_with_stdio(false);
20     cin.tie(nullptr);
21     int t;
22     cin >> t;
23     while (t--) {
24         solve();
25     }
26     return 0;
27 }
```

### 30: Difference of GCDs

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given three integers  $n$ ,  $l$ , and  $r$ . You need to construct an array  $a_1, a_2, \dots, a_n$  ( $l \leq a_i \leq r$ ) such that  $\gcd(i, a_i)$  are all distinct or report there's no solution.

Here  $\gcd(x, y)$  denotes the greatest common divisor (GCD) of integers  $x$  and  $y$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, l, r;
6     cin >> n >> l >> r;
7     vector<int> a(n);
8     for (int i = 1; i <= n; i++) {
9         if (r / i == (l - 1) / i) {
10             cout << "NO\n";
11             return;
12         }
13         a[i - 1] = r / i * i;
14     }
15     cout << "YES\n";
16     for (int i = 0; i < n; i++) {
17         cout << a[i] << " \n"[i == n - 1];
18     }
19 }
20 int main() {
21     ios::sync_with_stdio(false);
22     cin.tie(nullptr);
23     int t;
24     cin >> t;
25     while (t--) {
26         solve();
27     }
28     return 0;
29 }
```

### 31: Two Binary Strings

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given two strings  $a$  and  $b$  of equal length, consisting of only characters 0 and/or 1; both strings start with character 0 and end with character 1.

You can perform the following operation any number of times (possibly zero):

choose one of the strings and two equal characters in it; then turn all characters between them into those characters.

Formally, you choose one of these two strings (let the chosen string be  $s$ ), then pick two integers  $l$  and  $r$  such that  $1 \leq l < r \leq |s|$  and  $s_l = s_r$ , then replace every character  $s_i$  such that  $l < i < r$  with  $s_l$ .

For example, if the chosen string is 010101, you can transform it into one of the following strings by applying one operation:

000101 if you choose  $l = 1$  and  $r = 3$ ;

000001 if you choose  $l = 1$  and  $r = 5$ ;

010001 if you choose  $l = 3$  and  $r = 5$ ;

010111 if you choose  $l = 4$  and  $r = 6$ ;

011111 if you choose  $l = 2$  and  $r = 6$ ;

011101 if you choose  $l = 2$  and  $r = 4$ .

You have to determine if it's possible to make the given strings equal by applying this operation any number of times.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     string a, b;
6     cin >> a >> b;
7     int n = a.size();
8     for (int i = 0; i < n - 1; i++) {
9         if (a[i] == '0' && b[i] == '0' && a[i + 1] == '1' && b[i + 1] == '1') {
10             cout << "YES\n";
11             return;
12         }
13     }
14     cout << "NO\n";
15 }
16 int main() {
17     ios::sync_with_stdio(false);
18     cin.tie(nullptr);
19     int t;
20     cin >> t;
21     while (t--) {
22         solve();
23     }
24     return 0;
25 }
```

## 32: Prime Deletion

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

A prime number is a positive integer that has exactly two different positive divisors: 1 and the integer itself. For example, 2, 3, 13 and 101 are prime numbers; 1, 4, 6 and 42 are not.

You are given a sequence of digits from 1 to 9, in which every digit from 1 to 9 appears exactly once.

You are allowed to do the following operation several (maybe zero) times: choose any digit from the sequence and delete it. However, you cannot perform this operation if the sequence consists of only two digits.

Your goal is to obtain a sequence which represents a prime number. Note that you cannot reorder the digits in the sequence.

Print the resulting sequence, or report that it is impossible to perform the operations so that the resulting sequence is a prime number.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     string s;
6     cin >> s;
7     if (s.find('1') < s.find('3')) {
8         cout << "13\n";
9     } else {
10        cout << "31\n";
11    }
12 }
13 int main() {
14     ios::sync_with_stdio(false);
15     cin.tie(nullptr);
16     int t;
17     cin >> t;
18     while (t--) {
19         solve();
20     }
21     return 0;
22 }
```

### 33: Swap and Reverse

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a string  $s$  of length  $n$  consisting of lowercase English letters, and an integer  $k$ . In one step you can perform any one of the two operations below:

Pick an index  $i$  ( $1 \leq i \leq n - 2$ ) and swap  $s_i$  and  $s_{i+2}$ .

Pick an index  $i$  ( $1 \leq i \leq n - k + 1$ ) and reverse the order of letters formed by the range  $[i, i + k - 1]$  of the string. Formally, if the string currently is equal to  $s_1 \dots s_{i-1} s_i s_{i+1} \dots s_{i+k-2} s_{i+k-1} s_{i+k} \dots s_{n-1} s_n$ , change it to  $s_1 \dots s_{i-1} s_{i+k-1} s_{i+k-2} \dots s_{i+1} s_i s_{i+k} \dots s_{n-1} s_n$ .

You can make as many steps as you want (possibly, zero). Your task is to find the lexicographically smallest string you can obtain after some number of steps.

A string  $a$  is lexicographically smaller than a string  $b$  of the same length if and only if the following holds:

in the first position where  $a$  and  $b$  differ, the string  $a$  has a letter that appears earlier in the alphabet than the corresponding letter in  $b$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     string s;
8     cin >> s;
9     if (k % 2 == 0) {
10         sort(s.begin(), s.end());
11     } else {
12         for (int p = 0; p < 2; p++) {
13             string t;
14             for (int i = p; i < n; i += 2) {
15                 t += s[i];
16             }
17             sort(t.begin(), t.end());
18             for (int i = p; i < n; i += 2) {
19                 s[i] = t[i / 2];
20             }
21         }
22     }
23     cout << s << "\n";
24 }
25 int main() {
26     ios::sync_with_stdio(false);
27     cin.tie(nullptr);
28     int t;
29     cin >> t;
30     while (t--) {
31         solve();
32     }
33     return 0;
34 }
```

### 34: Increasing and Decreasing

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given three integers  $x$ ,  $y$ , and  $n$ .

Your task is to construct an array  $a$  consisting of  $n$  integers which satisfies the following conditions:

$$a_1 = x, a_n = y;$$

$a$  is strictly increasing (i.e.  $a_1 < a_2 < \dots < a_n$ );

if we denote  $b_i = a_{i+1} - a_i$  for  $1 \leq i \leq n - 1$ , then  $b$  is strictly decreasing (i.e.  $b_1 > b_2 > \dots > b_{n-1}$ ).

If there is no such array  $a$ , print a single integer  $-1$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int x, y, n;
6     cin >> x >> y >> n;
7     if (y - x < n * (n - 1) / 2) {
8         cout << -1 << "\n";
9         return;
10    }
11    vector<int> a(n);
12    a[0] = x;
13    a[n - 1] = y;
14    int t = 1;
15    for (int i = n - 2; i > 0; i--) {
16        y -= t;
17        t++;
18        a[i] = y;
19    }
20    for (int i = 0; i < n; i++) {
21        cout << a[i] << " \n"[i == n - 1];
22    }
23 }
24 int main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
27     int t;
28     cin >> t;
29     while (t--) {
30         solve();
31     }
32     return 0;
33 }
```

### 35: Sequence Game

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Tema and Vika are playing the following game.

First, Vika comes up with a sequence of positive integers  $a$  of length  $m$  and writes it down on a piece of paper. Then she takes a new piece of paper and writes down the sequence  $b$  according to the following rule:

First, she writes down  $a_1$ .

Then, she writes down only those  $a_i$  ( $2 \leq i \leq m$ ) such that  $a_{i-1} \leq a_i$ . Let the length of this sequence be denoted as  $n$ .

For example, from the sequence  $a = [4, 3, 2, 6, 3, 3]$ , Vika will obtain the sequence  $b = [4, 6, 3]$ .

She then gives the piece of paper with the sequence  $b$  to Tema. He, in turn, tries to guess the sequence  $a$ .

Tema considers winning in such a game highly unlikely, but still wants to find at least one sequence  $a$  that could have been originally chosen by Vika. Help him and output any such sequence.

Note that the length of the sequence you output should not exceed the input sequence length by more than two times.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> b(n);
8     for (int i = 0; i < n; i++) {
9         cin >> b[i];
10    }
11    vector<int> a;
12    a.reserve(2 * n);
13    for (int i = 0; i < n; i++) {
14        if (i == 0 || b[i] >= b[i - 1]) {
15            a.push_back(b[i]);
16        } else {
17            a.push_back(b[i]);
18            a.push_back(b[i]);
19        }
20    }
21 }
```

```

19         }
20     }
21     const int m = a.size();
22     cout << m << "\n";
23     for (int i = 0; i < m; i++) {
24         cout << a[i] << " \n"[i == m - 1];
25     }
26 }
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(nullptr);
30     int t;
31     cin >> t;
32     while (t--) {
33         solve();
34     }
35     return 0;
36 }
```

### 36: Not a Substring

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A bracket sequence is a string consisting of characters ‘(’ and/or ‘)’. A regular bracket sequence is a bracket sequence that can be transformed into a correct arithmetic expression by inserting characters ‘1’ and ‘+’ between the original characters of the sequence. For example:

bracket sequences “()()” and “(()” are regular (they can be transformed into “(1)+(1)” and “((1+1)+1)”, respectively);

bracket sequences “)”, “(” and “)” are not regular.

You are given a bracket sequence  $s$ ; let’s define its length as  $n$ . Your task is to find a regular bracket sequence  $t$  of length  $2n$  such that  $s$  does not occur in  $t$  as a contiguous substring, or report that there is no such sequence.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     string s;
6     cin >> s;
7     int n = s.size();
```

```

8     if (s == "()") {
9         cout << "NO\n";
10    return;
11   }
12   cout << "YES\n";
13   bool alt = true;
14   for (int i = 1; i < n; i++) {
15       if (s[i] == s[i - 1]) {
16           alt = false;
17       }
18   }
19   if (alt) {
20       cout << string(n, '(') + string(n, ')') << "\n";
21   } else {
22       for (int i = 1; i <= n; i++) {
23           cout << "()";
24       }
25       cout << "\n";
26   }
27 }
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(nullptr);
31     int t;
32     cin >> t;
33     while (t--) {
34         solve();
35     }
36     return 0;
37 }
```

### 37: Yet Another Permutation Problem

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Alex got a new game called “GCD permutations” as a birthday present. Each round of this game proceeds as follows:

First, Alex chooses a permutation<sup>†</sup>  $a_1, a_2, \dots, a_n$  of integers from 1 to  $n$ .

Then, for each  $i$  from 1 to  $n$ , an integer  $d_i = \gcd(a_i, a_{(i \bmod n)+1})$  is calculated.

The score of the round is the number of distinct numbers among  $d_1, d_2, \dots, d_n$ .

Alex has already played several rounds so he decided to find a permutation  $a_1, a_2, \dots, a_n$  such that its score is as large as possible.

Recall that  $\gcd(x, y)$  denotes the greatest common divisor (GCD) of numbers  $x$  and  $y$ , and  $x \bmod y$  denotes the remainder of dividing  $x$  by  $y$ .

<sup>†</sup>A permutation of length  $n$  is an array consisting of  $n$  distinct integers from 1 to  $n$  in arbitrary order. For example,  $[2, 3, 1, 5, 4]$  is a permutation, but  $[1, 2, 2]$  is not a permutation (2 appears twice in the array), and  $[1, 3, 4]$  is also not a permutation ( $n = 3$  but there is 4 in the array).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a;
8     a.reserve(n);
9     for (int i = 1; i <= n; i += 2) {
10         for (int j = i; j <= n; j *= 2) {
11             a.push_back(j);
12         }
13     }
14     for (int i = 0; i < n; i++) {
15         cout << a[i] << " \n"[i == n - 1];
16     }
17 }
18 int main() {
19     ios::sync_with_stdio(false);
20     cin.tie(nullptr);
21     int t;
22     cin >> t;
23     while (t--) {
24         solve();
25     }
26     return 0;
27 }
```

### 38: Olya and Game with Arrays

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Artem suggested a game to the girl Olya. There is a list of  $n$  arrays, where the  $i$ -th array contains  $m_i \geq 2$  positive integers  $a_{i,1}, a_{i,2}, \dots, a_{i,m_i}$ .

Olya can move at most one (possibly 0) integer from each array to another array. Note that integers can be moved from one array only once, but integers can be added to one array multiple times, and all the movements are done at the same time.

The beauty of the list of arrays is defined as the sum  $\sum_{i=1}^n \min_{j=1}^{m_i} a_{i,j}$ . In other words, for each array, we find the minimum value in it and then sum up these values.

The goal of the game is to maximize the beauty of the list of arrays. Help Olya win this challenging game!

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<vector<int>> a(n);
8     int min = 1E9;
9     int min2 = 1E9;
10    i64 ans = 0;
11    for (int i = 0; i < n; i++) {
12        int m;
13        cin >> m;
14        a[i].resize(m);
15        for (int j = 0; j < m; j++) {
16            cin >> a[i][j];
17        }
18        nth_element(a[i].begin(), a[i].begin() + 1, a[i].end());
19        min = min(min, a[i][0]);
20        min2 = min(min2, a[i][1]);
21        ans += a[i][1];
22    }
23    ans -= min2;
24    ans += min;
25    cout << ans << "\n";
26 }
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(nullptr);
30     int t;
31     cin >> t;
32     while (t--) {
33         solve();
34     }
35     return 0;
36 }
```

### 39: United We Stand

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Given an array  $a$  of length  $n$ , containing integers. And there are two initially empty arrays  $b$  and  $c$ . You need to add each element of array  $a$  to exactly one of the arrays  $b$  or  $c$ , in order to satisfy the following conditions:

Both arrays  $b$  and  $c$  are non-empty. More formally, let  $l_b$  be the length of array  $b$ , and  $l_c$  be the length of array  $c$ . Then  $l_b, l_c \geq 1$ .

For any two indices  $i$  and  $j$  ( $1 \leq i \leq l_b, 1 \leq j \leq l_c$ ),  $c_j$  is not a divisor of  $b_i$ .

Output the arrays  $b$  and  $c$  that can be obtained, or output  $-1$  if they do not exist.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct SCC;
5 void solve() {
6     int n;
7     cin >> n;
8     SCC g(n);
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    for (int i = 0; i < n; i++) {
14        for (int j = 0; j < n; j++) {
15            if (a[i] % a[j] == 0) {
16                g.addEdge(i, j);
17            }
18        }
19    }
20    auto bel = g.work();
21    if (bel == vector(n, 0)) {
22        cout << -1 << "\n";
23        return;
24    }
25    vector<int> b, c;
26    for (int i = 0; i < n; i++) {
27        if (bel[i] == 0) {
28            b.push_back(i);
29        } else {
30            c.push_back(i);
31        }
32    }
33    cout << b.size() << " " << c.size() << "\n";
34    for (auto i : b) {
35        cout << a[i] << " \n"[i == b.back()];
36    }
37    for (auto i : c) {
38        cout << a[i] << " \n"[i == c.back()];
39    }
40 }
41 int main() {
42     ios::sync_with_stdio(false);
43     cin.tie(nullptr);
44     int t;
45     cin >> t;
46     while (t--) {
47         solve();
48     }
49     return 0;
50 }
```

## 40: Escalator Conversations

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

One day, Vlad became curious about who he can have a conversation with on the escalator in the subway. There are a total of  $n$  passengers. The escalator has a total of  $m$  steps, all steps indexed from 1 to  $m$  and  $i$ -th step has height  $i \cdot k$ .

Vlad's height is  $H$  centimeters. Two people with heights  $a$  and  $b$  can have a conversation on the escalator if they are standing on different steps and the height difference between them is equal to the height difference between the steps.

For example, if two people have heights 170 and 180 centimeters, and  $m = 10$ ,  $k = 5$ , then they can stand on steps numbered 7 and 5, where the height difference between the steps is equal to the height difference between the two people:  $k \cdot 2 = 5 \cdot 2 = 10 = 180 - 170$ . There are other possible ways.

Given an array  $h$  of size  $n$ , where  $h_i$  represents the height of the  $i$ -th person. Vlad is interested in how many people he can have a conversation with on the escalator individually.

For example, if  $n = 5$ ,  $m = 3$ ,  $k = 3$ ,  $H = 11$ , and  $h = [5, 4, 14, 18, 2]$ , Vlad can have a conversation with the person with height 5 (Vlad will stand on step 1, and the other person will stand on step 3) and with the person with height 14 (for example, Vlad can stand on step 3, and the other person will stand on step 2). Vlad cannot have a conversation with the person with height 2 because even if they stand on the extreme steps of the escalator, the height difference between them will be 6, while their height difference is 9. Vlad cannot have a conversation with the rest of the people on the escalator, so the answer for this example is 2.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m, k, H;
6     cin >> n >> m >> k >> H;
7     int ans = 0;
8     for (int i = 0; i < n; i++) {
9         int h;
10        cin >> h;
11        if (h != H && (h - H) % k == 0 && abs(h - H) < m * k) {
12            ans++;
13        }
}

```

```

14     }
15     cout << ans << "\n";
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

**greedy****41: Training Before the Olympiad**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Masha and Olya have an important team olympiad coming up soon. In honor of this, Masha, for warm-up, suggested playing a game with Olya:

There is an array  $a$  of size  $n$ . Masha goes first, and the players take turns. Each move is described by the following sequence of actions:

- If the size of the array is 1, the game ends.
- The player who is currently playing chooses two different indices  $i, j$  ( $1 \leq i, j \leq |a|$ ), and performs the following operation - removes  $a_i$  and  $a_j$  from the array and adds to the array a number equal to  $\lfloor \frac{a_i + a_j}{2} \rfloor \cdot 2$ . In other words, first divides the sum of the numbers  $a_i, a_j$  by 2 rounding down, and then multiplies the result by 2.

Masha aims to maximize the final number, while Olya aims to minimize it.

Masha and Olya decided to play on each non-empty prefix of the initial array  $a$ , and asked for your help.

For each  $k = 1, 2, \dots, n$ , answer the following question. Let only the first  $k$  elements of the array  $a$  be present in the game, with indices  $1, 2, \dots, k$  respectively. What number will remain at the end with optimal play by both players?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     i64 sum = 0;
8     int odd = 0;
9     for (int i = 1; i <= n; i++) {
10         int a;
11         cin >> a;
12         sum += a;
13         odd += a % 2;
14         int res;
15         if (odd == 1 && i == 1) {
16             res = 0;
17         } else {
18             res = odd / 3;
19             if (odd % 3 == 1) {
20                 res += 1;
21             }
22         }
23         cout << sum - res << "\n"[i == n];
24     }
25 }
26 int main() {
27     ios::sync_with_stdio(false);
28     cin.tie(nullptr);
29     int t;
30     cin >> t;
31     while (t--) {
32         solve();
33     }
34     return 0;
35 }
```

## 42: Unnatural Language Processing

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Lura was bored and decided to make a simple language using the five letters a, b, c, d, e. There are two types of letters:

vowels - the letters a and e. They are represented by V.

consonants - the letters b, c, and d. They are represented by C.

A word in the language is a sequence of syllables. Lura has written a word in the language, but she doesn't know how to split it into syllables. Help her break the word into syllables.

For example, given the word bacedbab, it would be split into syllables as ba . ced . bab (the dot . represents a syllable boundary).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 bool isVowel(char c) {
5     return c == 'a' || c == 'e';
6 }
7 void solve() {
8     int n;
9     cin >> n;
10    string s;
11    cin >> s;
12    vector<string> t;
13    for (int i = n - 1; i >= 0; ) {
14        if (isVowel(s[i])) {
15            t.push_back(s.substr(i - 1, 2));
16            i -= 2;
17        } else {
18            t.push_back(s.substr(i - 2, 3));
19            i -= 3;
20        }
21    }
22    reverse(t.begin(), t.end());
23    for (int i = 0; i < t.size(); i++) {
24        cout << t[i] << ".\n"[i == t.size() - 1];
25    }
26 }
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(nullptr);
30     int t;
31     cin >> t;
32     while (t--) {
33         solve();
34     }
35     return 0;
36 }
```

### 43: Three Activities

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Winter holidays are coming up. They are going to last for  $n$  days.

During the holidays, Monocarp wants to try all of these activities exactly once with his friends:

go skiing;

watch a movie in a cinema;

play board games.

Monocarp knows that, on the  $i$ -th day, exactly  $a_i$  friends will join him for skiing,  $b_i$  friends will join him for a movie and  $c_i$  friends will join him for board games.

Monocarp also knows that he can't try more than one activity in a single day.

Thus, he asks you to help him choose three distinct days  $x, y, z$  in such a way that the total number of friends to join him for the activities ( $a_x + b_y + c_z$ ) is maximized.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n), b(n), c(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    for (int i = 0; i < n; i++) {
12        cin >> b[i];
13    }
14    for (int i = 0; i < n; i++) {
15        cin >> c[i];
16    }
17    vector<int> pa(n), pb(n), pc(n);
18    iota(pa.begin(), pa.end(), 0);
19    iota(pb.begin(), pb.end(), 0);
20    iota(pc.begin(), pc.end(), 0);
21    sort(pa.begin(), pa.end(),
22          [&](int i, int j) {
23              return a[i] > a[j];
24          });
25    sort(pb.begin(), pb.end(),
26          [&](int i, int j) {
27              return b[i] > b[j];
28          });
29    sort(pc.begin(), pc.end(),
30          [&](int i, int j) {
31              return c[i] > c[j];
32          });
33    int ans = 0;
34    for (int i = 0; i < 3; i++) {
35        for (int j = 0; j < 3; j++) {
36            for (int k = 0; k < 3; k++) {
37                int x = pa[i], y = pb[j], z = pc[k];
38                if (x != y && y != z && x != z) {
39                    ans = max(ans, a[x] + b[y] + c[z]);
40                }
41            }
42        }
43    }
}

```

```

44     cout << ans << "\n";
45 }
46 int main() {
47     ios::sync_with_stdio(false);
48     cin.tie(nullptr);
49     int t;
50     cin >> t;
51     while (t--) {
52         solve();
53     }
54     return 0;
55 }
```

## 44: Quests

- Time limit: 2.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp is playing a computer game. In order to level up his character, he can complete quests. There are  $n$  quests in the game, numbered from 1 to  $n$ .

Monocarp can complete quests according to the following rules:

the 1-st quest is always available for completion;

the  $i$ -th quest is available for completion if all quests  $j < i$  have been completed at least once.

Note that Monocarp can complete the same quest multiple times.

For each completion, the character gets some amount of experience points:

for the first completion of the  $i$ -th quest, he gets  $a_i$  experience points;

for each subsequent completion of the  $i$ -th quest, he gets  $b_i$  experience points.

Monocarp is a very busy person, so he has free time to complete no more than  $k$  quests. Your task is to calculate the maximum possible total experience Monocarp can get if he can complete no more than  $k$  quests.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
```

```

6     cin >> n >> k;
7     vector<int> a(n), b(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    for (int i = 0; i < n; i++) {
12        cin >> b[i];
13    }
14    i64 ans = 0;
15    i64 sum = 0;
16    int max = 0;
17    for (int i = 0; i < n && i + 1 <= k; i++) {
18        sum += a[i];
19        max = max(max, b[i]);
20        ans = max(ans, sum + 1LL * max * (k - i - 1));
21    }
22    cout << ans << "\n";
23 }
24 int main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
27     int t;
28     cin >> t;
29     while (t--) {
30         solve();
31     }
32     return 0;
33 }
```

## 45: Begginer's Zelda

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a tree<sup>†</sup>. In one zelda-operation you can do follows:

Choose two vertices of the tree  $u$  and  $v$ ;

Compress all the vertices on the path from  $u$  to  $v$  into one vertex. In other words, all the vertices on path from  $u$  to  $v$  will be erased from the tree, a new vertex  $w$  will be created. Then every vertex  $s$  that had an edge to some vertex on the path from  $u$  to  $v$  will have an edge to the vertex  $w$ .

Determine the minimum number of zelda-operations required for the tree to have only one vertex.

<sup>†</sup>A tree is a connected acyclic undirected graph.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
```

```

3  using i64 = long long;
4  void solve() {
5      int n;
6      cin >> n;
7      vector<int> deg(n);
8      for (int i = 1; i < n; i++) {
9          int u, v;
10         cin >> u >> v;
11         u--, v--;
12         deg[u] += 1;
13         deg[v] += 1;
14     }
15     int ans = (count(deg.begin(), deg.end(), 1) + 1) / 2;
16     cout << ans << "\n";
17 }
18 int main() {
19     ios::sync_with_stdio(false);
20     cin.tie(nullptr);
21     int t;
22     cin >> t;
23     while (t--) {
24         solve();
25     }
26     return 0;
27 }
```

## 46: Removal of Unattractive Pairs

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vlad found a string  $s$  consisting of  $n$  lowercase Latin letters, and he wants to make it as short as possible.

To do this, he can remove any pair of adjacent characters from  $s$  any number of times, provided they are different. For example, if  $s=racoон$ , then by removing one pair of characters he can obtain the strings coon, roon, raon, and raco, but he cannot obtain racn (because the removed letters were the same) or rcon (because the removed letters were not adjacent).

What is the minimum length Vlad can achieve by applying any number of deletions?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
```

```

5     int n;
6     cin >> n;
7     string s;
8     cin >> s;
9     array<int, 26> cnt{};
10    for (auto c : s) {
11        cnt[c - 'a'] += 1;
12    }
13    int ans = n - 2 * min(n / 2, n - *max_element(cnt.begin(), cnt.end()));
14    cout << ans << "\n";
15 }
16 int main() {
17     ios::sync_with_stdio(false);
18     cin.tie(nullptr);
19     int t;
20     cin >> t;
21     while (t--) {
22         solve();
23     }
24     return 0;
25 }
```

## 47: Chip and Ribbon

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There is a ribbon divided into  $n$  cells, numbered from 1 to  $n$  from left to right. Initially, an integer 0 is written in each cell.

Monocarp plays a game with a chip. The game consists of several turns. During the first turn, Monocarp places the chip in the 1-st cell of the ribbon. During each turn except for the first turn, Monocarp does exactly one of the two following actions:

move the chip to the next cell (i. e. if the chip is in the cell  $i$ , it is moved to the cell  $i + 1$ ). This action is impossible if the chip is in the last cell;

choose any cell  $x$  and teleport the chip into that cell. It is possible to choose the cell where the chip is currently located.

At the end of each turn, the integer written in the cell with the chip is increased by 1.

Monocarp's goal is to make some turns so that the 1-st cell contains the integer  $c_1$ , the 2-nd cell contains the integer  $c_2, \dots$ , the  $n$ -th cell contains the integer  $c_n$ . He wants to teleport the chip as few times as possible.

Help Monocarp calculate the minimum number of times he has to teleport the chip.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> c(n);
8     for (int i = 0; i < n; i++) {
9         cin >> c[i];
10    }
11    i64 ans = c[0];
12    for (int i = 1; i < n; i++) {
13        ans += max(0, c[i] - c[i - 1]);
14    }
15    cout << ans - 1 << "\n";
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

## 48: Line Trip

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There is a road, which can be represented as a number line. You are located in the point 0 of the number line, and you want to travel from the point 0 to the point  $x$ , and back to the point 0.

You travel by car, which spends 1 liter of gasoline per 1 unit of distance travelled. When you start at the point 0, your car is fully fueled (its gas tank contains the maximum possible amount of fuel).

There are  $n$  gas stations, located in points  $a_1, a_2, \dots, a_n$ . When you arrive at a gas station, you fully refuel your car. Note that you can refuel only at gas stations, and there are no gas stations in points 0 and  $x$ .

You have to calculate the minimum possible volume of the gas tank in your car (in liters) that will allow you to travel from the point 0 to the point  $x$  and back to the point 0.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, x;
6     cin >> n >> x;
7     vector<int> a(n + 1);
8     for (int i = 1; i <= n; i++) {
9         cin >> a[i];
10    }
11    int ans = 2 * (x - a[n]);
12    for (int i = 1; i <= n; i++) {
13        ans = max(ans, a[i] - a[i - 1]);
14    }
15    cout << ans << "\n";
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

## 49: StORage room

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In Cyprus, the weather is pretty hot. Thus, Theofanis saw this as an opportunity to create an ice cream company.

He keeps the ice cream safe from other ice cream producers by locking it inside big storage rooms. However, he forgot the password. Luckily, the lock has a special feature for forgetful people!

It gives you a table  $M$  with  $n$  rows and  $n$  columns of non-negative integers, and to open the lock, you need to find an array  $a$  of  $n$  elements such that:

$0 \leq a_i < 2^{30}$ , and

$M_{i,j} = a_i | a_j$  for all  $i \neq j$ , where  $|$  denotes the bitwise OR operation.

The lock has a bug, and sometimes it gives tables without any solutions. In that case, the ice cream will remain frozen for the rest of eternity.

Can you find an array to open the lock?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector M(n, vector<int>(n));
8     vector<int> a(n, (1 << 30) - 1);
9     for (int i = 0; i < n; i++) {
10         for (int j = 0; j < n; j++) {
11             cin >> M[i][j];
12             if (i != j) {
13                 a[i] &= M[i][j];
14                 a[j] &= M[i][j];
15             }
16         }
17     }
18     for (int i = 0; i < n; i++) {
19         for (int j = 0; j < n; j++) {
20             if (i != j && M[i][j] != (a[i] | a[j])) {
21                 cout << "NO\n";
22                 return;
23             }
24         }
25     }
26     cout << "YES\n";
27     for (int i = 0; i < n; i++) {
28         cout << a[i] << " \n"[i == n - 1];
29     }
30 }
31 int main() {
32     ios::sync_with_stdio(false);
33     cin.tie(nullptr);
34     int t;
35     cin >> t;
36     while (t--) {
37         solve();
38     }
39     return 0;
40 }
```

## 50: Halloumi Boxes

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Theofanis is busy after his last contest, as now, he has to deliver many halloumis all over the world. He stored them inside  $n$  boxes and each of which has some number  $a_i$  written on it.

He wants to sort them in non-decreasing order based on their number, however, his machine works in a strange way. It can only reverse any subarray<sup>†</sup> of boxes with length at most  $k$ .

Find if it's possible to sort the boxes using any number of reverses.

<sup>†</sup> Reversing a subarray means choosing two indices  $i$  and  $j$  (where  $1 \leq i \leq j \leq n$ ) and changing the array  $a_1, a_2, \dots, a_n$  to  $a_1, a_2, \dots, a_{i-1}, a_j, a_{j-1}, \dots, a_i, a_{j+1}, \dots, a_{n-1}, a_n$ . The length of the subarray is then  $j - i + 1$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    if (k > 1 || is_sorted(a.begin(), a.end())) {
12        cout << "YES\n";
13    } else {
14        cout << "NO\n";
15    }
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

## 51: Getting Points

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp is a student at Berland State University. Due to recent changes in the Berland education system, Monocarp has to study only one subject - programming.

The academic term consists of  $n$  days, and in order not to get expelled, Monocarp has to earn at least  $P$  points during those  $n$  days. There are two ways to earn points - completing practical tasks and

attending lessons. For each practical task Monocarp fulfills, he earns  $t$  points, and for each lesson he attends, he earns  $l$  points.

Practical tasks are unlocked “each week” as the term goes on: the first task is unlocked on day 1 (and can be completed on any day from 1 to  $n$ ), the second task is unlocked on day 8 (and can be completed on any day from 8 to  $n$ ), the third task is unlocked on day 15, and so on.

Every day from 1 to  $n$ , there is a lesson which can be attended by Monocarp. And every day, Monocarp chooses whether to study or to rest the whole day. When Monocarp decides to study, he attends a lesson and can complete no more than 2 tasks, which are already unlocked and not completed yet. If Monocarp rests the whole day, he skips a lesson and ignores tasks.

Monocarp wants to have as many days off as possible, i. e. he wants to maximize the number of days he rests. Help him calculate the maximum number of days he can rest!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     i64 P;
7     int l, t;
8     cin >> n >> P >> l >> t;
9     int task = (n - 1) / 7 + 1;
10    int ans = max((P - 1) / (l + 2LL * t) + 1, (P - 1LL * t * task - 1) / l + 1);
11    ans = n - ans;
12    cout << ans << "\n";
13 }
14 int main() {
15     ios::sync_with_stdio(false);
16     cin.tie(nullptr);
17     int t;
18     cin >> t;
19     while (t--) {
20         solve();
21     }
22     return 0;
23 }
```

## 52: Cover in Water

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Filip has a row of cells, some of which are blocked, and some are empty. He wants all empty cells to have water in them. He has two actions at his disposal:

1 - place water in an empty cell.

2 - remove water from a cell and place it in any other empty cell.

If at some moment cell  $i$  ( $2 \leq i \leq n - 1$ ) is empty and both cells  $i - 1$  and  $i + 1$  contains water, then it becomes filled with water.

Find the minimum number of times he needs to perform action 1 in order to fill all empty cells with water.

Note that you don't need to minimize the use of action 2. Note that blocked cells neither contain water nor can Filip place water in them.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     string s;
8     cin >> s;
9     int ans = 0;
10    for (int l = 0, r = 0; l < n; l = r) {
11        while (r < n && s[l] == s[r]) {
12            r++;
13        }
14        if (s[l] == '.') {
15            ans += r - l;
16            if (r - l >= 3) {
17                ans = 2;
18                break;
19            }
20        }
21    }
22    cout << ans << "\n";
23 }
24 int main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
27     int t;
28     cin >> t;
29     while (t--) {
30         solve();
31     }
32     return 0;
33 }
```

### 53: Yarik and Array

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A subarray is a continuous part of array.

Yarik recently found an array  $a$  of  $n$  elements and became very interested in finding the maximum sum of a non empty subarray. However, Yarik doesn't like consecutive integers with the same parity, so the subarray he chooses must have alternating parities for adjacent elements.

For example,  $[1, 2, 3]$  is acceptable, but  $[1, 2, 4]$  is not, as 2 and 4 are both even and adjacent.

You need to help Yarik by finding the maximum sum of such a subarray.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    int ans = -1E9;
12    int suf = -1E9;
13    for (int i = 0; i < n; i++) {
14        if (i && (a[i] - a[i - 1]) % 2 == 0) {
15            suf = 0;
16        }
17        suf = max(suf, 0) + a[i];
18        ans = max(ans, suf);
19    }
20    cout << ans << "\n";
21 }
22 int main() {
23     ios::sync_with_stdio(false);
24     cin.tie(nullptr);
25     int t;
26     cin >> t;
27     while (t--) {
28         solve();
29     }
30     return 0;
31 }
```

## 54: Points and Minimum Distance

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given a sequence of integers  $a$  of length  $2n$ . You have to split these  $2n$  integers into  $n$  pairs; each pair will represent the coordinates of a point on a plane. Each number from the sequence  $a$  should become the  $x$  or  $y$  coordinate of exactly one point. Note that some points can be equal.

After the points are formed, you have to choose a path  $s$  that starts from one of these points, ends at one of these points, and visits all  $n$  points at least once.

The length of path  $s$  is the sum of distances between all adjacent points on the path. In this problem, the distance between two points  $(x_1, y_1)$  and  $(x_2, y_2)$  is defined as  $|x_1 - x_2| + |y_1 - y_2|$ .

Your task is to form  $n$  points and choose a path  $s$  in such a way that the length of path  $s$  is minimized.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(2 * n);
8     for (int i = 0; i < 2 * n; i++) {
9         cin >> a[i];
10    }
11    sort(a.begin(), a.end());
12    int ans = a[n - 1] - a[0] + a[2 * n - 1] - a[n];
13    cout << ans << "\n";
14    for (int i = 0; i < n; i++) {
15        cout << a[i] << " " << a[n + i] << "\n";
16    }
17 }
18 int main() {
19     ios::sync_with_stdio(false);
20     cin.tie(nullptr);
21     int t;
22     cin >> t;
23     while (t--) {
24         solve();
25     }
26     return 0;
27 }
```

## 55: Haunted House

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a number in binary representation consisting of exactly  $n$  bits, possibly, with leading zeroes. For example, for  $n = 5$  the number 6 will be given as 00110, and for  $n = 4$  the number 9 will be given as 1001.

Let's fix some integer  $i$  such that  $1 \leq i \leq n$ . In one operation you can swap any two adjacent bits in the binary representation. Your goal is to find the smallest number of operations you are required to perform to make the number divisible by  $2^i$ , or say that it is impossible.

Please note that for each  $1 \leq i \leq n$  you are solving the problem independently.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     string s;
8     cin >> s;
9     i64 ans = 0;
10    int l = n, r = n;
11    for (int i = 1; i <= n; i++) {
12        if (ans != -1) {
13            if (l > n - i && s[n - i] == '0') {
14                l -= 1;
15            } else {
16                while (l > 0 && s[l - 1] == '1') {
17                    l -= 1;
18                }
19                if (l == 0) {
20                    ans = -1;
21                } else {
22                    ans += r - l;
23                }
24                l -= 1;
25            }
26            r -= 1;
27        }
28        cout << ans << " \n"[i == n];
29    }
30 }
31 int main() {
32     ios::sync_with_stdio(false);
33     cin.tie(nullptr);

```

```

34     int t;
35     cin >> t;
36     while (t--) {
37         solve();
38     }
39     return 0;
40 }
```

## 56: Simple Design

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A positive integer is called  $k$ -beautiful, if the digit sum of the decimal representation of this number is divisible by  $k$ <sup>†</sup>. For example, 9272 is 5-beautiful, since the digit sum of 9272 is  $9 + 2 + 7 + 2 = 20$ .

You are given two integers  $x$  and  $k$ . Please find the smallest integer  $y \geq x$  which is  $k$ -beautiful.

<sup>†</sup> An integer  $n$  is divisible by  $k$  if there exists an integer  $m$  such that  $n = k \cdot m$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int f(int x) {
5     return x ? f(x / 10) + x % 10 : 0;
6 }
7 void solve() {
8     int x, k;
9     cin >> x >> k;
10    while (f(x) % k != 0) {
11        x++;
12    }
13    cout << x << "\n";
14 }
15 int main() {
16     ios::sync_with_stdio(false);
17     cin.tie(nullptr);
18     int t;
19     cin >> t;
20     while (t--) {
21         solve();
22     }
23     return 0;
24 }
```

## 57: Gamer Hemose

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

One day, Ahmed\_Hossam went to Hemose and said “Let’s solve a gym contest!”. Hemose didn’t want to do that, as he was playing Valorant, so he came up with a problem and told it to Ahmed to distract him. Sadly, Ahmed can’t solve it... Could you help him?

There is an Agent in Valorant, and he has  $n$  weapons. The  $i$ -th weapon has a damage value  $a_i$ , and the Agent will face an enemy whose health value is  $H$ .

The Agent will perform one or more moves until the enemy dies.

In one move, he will choose a weapon and decrease the enemy’s health by its damage value. The enemy will die when his health will become less than or equal to 0. However, not everything is so easy: the Agent can’t choose the same weapon for 2 times in a row.

What is the minimum number of times that the Agent will need to use the weapons to kill the enemy?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, H;
6     cin >> n >> H;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    nth_element(a.begin(), a.begin() + 1, a.end(), greater());
12    int ans = H / (a[0] + a[1]) * 2;
13    H %= a[0] + a[1];
14    if (H > 0) {
15        ans += 1;
16    }
17    if (H > a[0]) {
18        ans += 1;
19    }
20    cout << ans << "\n";
21 }
22 int main() {
23     ios::sync_with_stdio(false);
24     cin.tie(nullptr);
25     int t;
26     cin >> t;
27     while (t--) {
28         solve();
29     }

```

```

30     return 0;
31 }
```

## 58: Make Them Equal

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Theofanis has a string  $s_1 s_2 \dots s_n$  and a character  $c$ . He wants to make all characters of the string equal to  $c$  using the minimum number of operations.

In one operation he can choose a number  $x$  ( $1 \leq x \leq n$ ) and for every position  $i$ , where  $i$  is not divisible by  $x$ , replace  $s_i$  with  $c$ .

Find the minimum number of operations required to make all the characters equal to  $c$  and the  $x$ -s that he should use in his operations.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     char c;
8     cin >> c;
9     string s;
10    cin >> s;
11    if (s == string(n, c)) {
12        cout << 0 << "\n";
13        return;
14    }
15    for (int i = 1; i <= n; i++) {
16        bool ok = true;
17        for (int j = i; j <= n; j += i) {
18            if (s[j - 1] != c) {
19                ok = false;
20            }
21        }
22        if (ok) {
23            cout << 1 << "\n";
24            cout << i << "\n";
25            return;
26        }
27    }
28    cout << 2 << "\n";
29    cout << n - 1 << " " << n << "\n";
30 }
31 int main() {
```

```

32     ios::sync_with_stdio(false);
33     cin.tie(nullptr);
34     int t;
35     cin >> t;
36     while (t--) {
37         solve();
38     }
39     return 0;
40 }
```

## 59: Helmets in Night Light

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Pak Chanek is the chief of a village named Khuntien. On one night filled with lights, Pak Chanek has a sudden and important announcement that needs to be notified to all of the  $n$  residents in Khuntien.

First, Pak Chanek shares the announcement directly to one or more residents with a cost of  $p$  for each person. After that, the residents can share the announcement to other residents using a magical helmet-shaped device. However, there is a cost for using the helmet-shaped device. For each  $i$ , if the  $i$ -th resident has got the announcement at least once (either directly from Pak Chanek or from another resident), he/she can share the announcement to at most  $a_i$  other residents with a cost of  $b_i$  for each share.

If Pak Chanek can also control how the residents share the announcement to other residents, what is the minimum cost for Pak Chanek to notify all  $n$  residents of Khuntien about the announcement?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, p;
6     cin >> n >> p;
7     vector<int> a(n), b(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    for (int i = 0; i < n; i++) {
12        cin >> b[i];
13    }
14    vector<int> ord(n);
15    iota(ord.begin(), ord.end(), 0);
16    sort(ord.begin(), ord.end(),
17          [&](int i, int j) {
18              return b[i] < b[j];
19          });
20 }
```

```

19         });
20     i64 ans = p;
21     int k = n - 1;
22     for (auto i : ord) {
23         while (k > 0 && a[i] > 0) {
24             a[i]--, k--;
25             ans += min(p, b[i]);
26         }
27     }
28     cout << ans << "\n";
29 }
30 int main() {
31     ios::sync_with_stdio(false);
32     cin.tie(nullptr);
33     int t;
34     cin >> t;
35     while (t--) {
36         solve();
37     }
38     return 0;
39 }
```

## 60: How Much Does Daytona Cost?

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

We define an integer to be the most common on a subsegment, if its number of occurrences on that subsegment is larger than the number of occurrences of any other integer in that subsegment. A subsegment of an array is a consecutive segment of elements in the array  $a$ .

Given an array  $a$  of size  $n$ , and an integer  $k$ , determine if there exists a non-empty subsegment of  $a$  where  $k$  is the most common element.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    if (find(a.begin(), a.end(), k) != a.end()) {
12        cout << "YES\n";
13    } else {
14        cout << "NO\n";
15    }
}
```

```

16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

## implementation

### 61: 2023

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In a sequence  $a$ , whose product was equal to 2023,  $k$  numbers were removed, leaving a sequence  $b$  of length  $n$ . Given the resulting sequence  $b$ , find any suitable sequence  $a$  and output which  $k$  elements were removed from it, or state that such a sequence could not have existed.

Notice that you are not guaranteed that such array exists.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     int res = 2023;
8     vector<int> b(n);
9     for (int i = 0; i < n; i++) {
10         cin >> b[i];
11     }
12     for (int i = 0; i < n; i++) {
13         if (res % b[i]) {
14             cout << "NO\n";
15             return;
16         }
17         res /= b[i];
18     }
19     cout << "YES\n";
20     for (int i = 0; i < k; i++) {
21         cout << (i == 0 ? res : 1) << " \n"[i == k - 1];
```

```

22     }
23 }
24 int main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
27     int t;
28     cin >> t;
29     while (t--) {
30         solve();
31     }
32     return 0;
33 }
```

## 62: Can I Square?

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Calin has  $n$  buckets, the  $i$ -th of which contains  $a_i$  wooden squares of side length 1.

Can Calin build a square using all the given squares?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     i64 s = 0;
8     for (int i = 0; i < n; i++) {
9         int a;
10        cin >> a;
11        s += a;
12    }
13    i64 res = sqrt(s);
14    if (res * res == s) {
15        cout << "YES\n";
16    } else {
17        cout << "NO\n";
18    }
19 }
20 int main() {
21     ios::sync_with_stdio(false);
22     cin.tie(nullptr);
23     int t;
24     cin >> t;
25     while (t--) {
26         solve();
27     }
28     return 0;
29 }
```

## 63: Not Quite Latin Square

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A Latin square is a  $3 \times 3$  grid made up of the letters A, B, and C such that:

in each row, the letters A, B, and C each appear once, and

in each column, the letters A, B, and C each appear once.

You are given a Latin square, but one of the letters was replaced with a question mark ?. Find the letter that was replaced.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     char ans = 'A' ^ 'B' ^ 'C' ^ '?';
6     for (int i = 0; i < 9; i++) {
7         char c;
8         cin >> c;
9         ans ^= c;
10    }
11    cout << ans << "\n";
12 }
13 int main() {
14     ios::sync_with_stdio(false);
15     cin.tie(nullptr);
16     int t;
17     cin >> t;
18     while (t--) {
19         solve();
20     }
21     return 0;
22 }
```

## 64: Odd One Out

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given three digits  $a, b, c$ . Two of them are equal, but the third one is different from the other two.

Find the value that occurs exactly once.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int a, b, c;
6     cin >> a >> b >> c;
7     cout << (a ^ b ^ c) << "\n";
8 }
9 int main() {
10    ios::sync_with_stdio(false);
11    cin.tie(nullptr);
12    int t;
13    cin >> t;
14    while (t--) {
15        solve();
16    }
17    return 0;
18 }
```

## 65: Distinct Buttons

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are located at the point  $(0, 0)$  of an infinite Cartesian plane. You have a controller with 4 buttons which can perform one of the following operations:

U: move from  $(x, y)$  to  $(x, y + 1)$ ;

R: move from  $(x, y)$  to  $(x + 1, y)$ ;

D: move from  $(x, y)$  to  $(x, y - 1)$ ;

L: move from  $(x, y)$  to  $(x - 1, y)$ .

Unfortunately, the controller is broken. If you press all the 4 buttons (in any order), the controller stops working. It means that, during the whole trip, you can only press at most 3 distinct buttons (any number of times, in any order).

There are  $n$  special points in the plane, with integer coordinates  $(x_i, y_i)$ .

Can you visit all the special points (in any order) without breaking the controller?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     array<bool, 4> req{};
6     int n;
7     cin >> n;
8     while (n--) {
9         int x, y;
10        cin >> x >> y;
11        if (x > 0) {
12            req[0] = true;
13        }
14        if (x < 0) {
15            req[1] = true;
16        }
17        if (y > 0) {
18            req[2] = true;
19        }
20        if (y < 0) {
21            req[3] = true;
22        }
23    }
24    if (count(req.begin(), req.end(), true) == 4) {
25        cout << "NO\n";
26    } else {
27        cout << "YES\n";
28    }
29 }
30 int main() {
31     ios::sync_with_stdio(false);
32     cin.tie(nullptr);
33     int t;
34     cin >> t;
35     while (t--) {
36         solve();
37     }
38     return 0;
39 }
```

## 66: Problem Solving Log

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp is participating in a programming contest, which features 26 problems, named from 'A' to 'Z'. The problems are sorted by difficulty. Moreover, it's known that Monocarp can solve problem 'A' in 1 minute, problem 'B' in 2 minutes, ..., problem 'Z' in 26 minutes.

After the contest, you discovered his contest log - a string, consisting of uppercase Latin letters, such that the  $i$ -th letter tells which problem Monocarp was solving during the  $i$ -th minute of the contest. If

Monocarp had spent enough time in total on a problem to solve it, he solved it. Note that Monocarp could have been thinking about a problem after solving it.

Given Monocarp's contest log, calculate the number of problems he solved during the contest.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     string s;
8     cin >> s;
9     array<int, 26> cnt{};
10    for (auto c : s) {
11        cnt[c - 'A'] += 1;
12    }
13    int ans = 0;
14    for (int i = 0; i < 26; i++) {
15        ans += (cnt[i] > i);
16    }
17    cout << ans << "\n";
18 }
19 int main() {
20     ios::sync_with_stdio(false);
21     cin.tie(nullptr);
22     int t;
23     cin >> t;
24     while (t--) {
25         solve();
26     }
27     return 0;
28 }
```

## 67: Rating Increase

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp is a great solver of adhoc problems. Recently, he participated in an Educational Codeforces Round, and gained rating!

Monocarp knew that, before the round, his rating was  $a$ . After the round, it increased to  $b$  ( $b > a$ ). He wrote both values one after another to not forget them.

However, he wrote them so close to each other, that he can't tell now where the first value ends and the second value starts.

Please, help him find some values  $a$  and  $b$  such that:

neither of them has a leading zero;

both of them are strictly greater than 0;

$b > a$ ;

they produce the given value  $ab$  when written one after another.

If there are multiple answers, you can print any of them.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     string s;
6     cin >> s;
7     for (int i = 1; i < s.size(); i++) {
8         string a = s.substr(0, i);
9         string b = s.substr(i);
10        if (b[0] != '0' && stoi(b) > stoi(a)) {
11            cout << a << " " << b << "\n";
12            return;
13        }
14    }
15    cout << -1 << "\n";
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

## 68: YetanotherbrokenKeoard

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Polycarp has a problem - his laptop keyboard is broken.

Now, when he presses the ‘b’ key, it acts like an unusual backspace: it deletes the last (rightmost) lowercase letter in the typed string. If there are no lowercase letters in the typed string, then the press is completely ignored.

Similarly, when he presses the ‘B’ key, it deletes the last (rightmost) uppercase letter in the typed string. If there are no uppercase letters in the typed string, then the press is completely ignored.

In both cases, the letters ‘b’ and ‘B’ are not added to the typed string when these keys are pressed.

Consider an example where the sequence of key presses was “ARaBbbbitBaby”. In this case, the typed string will change as follows: “”  $\xrightarrow{A}$  “A”  $\xrightarrow{R}$  “AR”  $\xrightarrow{a}$  “ARa”  $\xrightarrow{B}$  “Aa”  $\xrightarrow{b}$  “A”  $\xrightarrow{j}$  “Ai”  $\xrightarrow{t}$  “Ait”  $\xrightarrow{B}$  “it”  $\xrightarrow{a}$  “ita”  $\xrightarrow{b}$  “it”  $\xrightarrow{y}$  “ity”.

Given a sequence of pressed keys, output the typed string after processing all key presses.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     string s;
6     cin >> s;
7     vector<int> u, l;
8     for (int i = 0; i < s.size(); i++) {
9         if (s[i] == 'B') {
10             if (!u.empty()) {
11                 u.pop_back();
12             }
13         } else if (s[i] == 'b') {
14             if (!l.empty()) {
15                 l.pop_back();
16             }
17         } else if (isupper(s[i])) {
18             u.push_back(i);
19         } else {
20             l.push_back(i);
21         }
22     }
23     int i = 0, j = 0;
24     while (i < u.size() || j < l.size()) {
25         if (i < u.size() && (j == l.size() || u[i] < l[j])) {
26             cout << s[u[i++]];
27         } else {
28             cout << s[l[j++]];
29         }
30     }
31     cout << "\n";
32 }
33 int main() {
34     ios::sync_with_stdio(false);
35     cin.tie(nullptr);

```

```

36     int t;
37     cin >> t;
38     while (t--) {
39         solve();
40     }
41     return 0;
42 }
```

**69: Rook**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

As you probably know, chess is a game that is played on a board with 64 squares arranged in an  $8 \times 8$  grid. Columns of this board are labeled with letters from a to h, and rows are labeled with digits from 1 to 8. Each square is described by the row and column it belongs to.

The rook is a piece in the game of chess. During its turn, it may move any non-zero number of squares horizontally or vertically. Your task is to find all possible moves for a rook on an empty chessboard.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     string s;
6     cin >> s;
7     for (char x = 'a'; x <= 'h'; x++) {
8         for (char y = '1'; y <= '8'; y++) {
9             if ((x == s[0]) ^ (y == s[1])) {
10                 cout << x << y << "\n";
11             }
12         }
13     }
14 }
15 int main() {
16     ios::sync_with_stdio(false);
17     cin.tie(nullptr);
18     int t;
19     cin >> t;
20     while (t--) {
21         solve();
22     }
23     return 0;
24 }
```

## 70: 250 Thousand Tons of TNT

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Alex is participating in the filming of another video of BrMeast, and BrMeast asked Alex to prepare 250 thousand tons of TNT, but Alex didn't hear him well, so he prepared  $n$  boxes and arranged them in a row waiting for trucks. The  $i$ -th box from the left weighs  $a_i$  tons.

All trucks that Alex is going to use hold the same number of boxes, denoted by  $k$ . Loading happens the following way:

The first  $k$  boxes goes to the first truck,

The second  $k$  boxes goes to the second truck,

...

The last  $k$  boxes goes to the  $\frac{n}{k}$ -th truck.

Upon loading is completed, each truck must have exactly  $k$  boxes. In other words, if at some point it is not possible to load exactly  $k$  boxes into the truck, then the loading option with that  $k$  is not possible.

Alex hates justice, so he wants the maximum absolute difference between the total weights of two trucks to be as great as possible. If there is only one truck, this value is 0.

Alex has quite a lot of connections, so for every  $1 \leq k \leq n$ , he can find a company such that each of its trucks can hold exactly  $k$  boxes. Print the maximum absolute difference between the total weights of any two trucks.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     vector<i64> s(n + 1);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11     }
12     for (int i = 0; i < n; i++) {
13         s[i + 1] = s[i] + a[i];
14     }

```

```

15     i64 ans = 0;
16     for (int k = 1; k <= n; k++) {
17         if (n % k == 0) {
18             i64 mx = 0, mn = s[n];
19             for (int i = 0; i < n; i += k) {
20                 i64 v = s[i + k] - s[i];
21                 mx = max(mx, v);
22                 mn = min(mn, v);
23             }
24             ans = max(ans, mx - mn);
25         }
26     }
27     cout << ans << "\n";
28 }
29 int main() {
30     ios::sync_with_stdio(false);
31     cin.tie(nullptr);
32     int t;
33     cin >> t;
34     while (t--) {
35         solve();
36     }
37     return 0;
38 }
```

## 71: Milica and String

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Milica has a string  $s$  of length  $n$ , consisting only of characters A and B. She wants to modify  $s$  so it contains exactly  $k$  instances of B. In one operation, she can do the following:

Select an integer  $i$  ( $1 \leq i \leq n$ ) and a character  $c$  ( $c$  is equal to either A or B).

Then, replace each of the first  $i$  characters of string  $s$  (that is, characters  $s_1, s_2, \dots, s_i$ ) with  $c$ .

Milica does not want to perform too many operations in order not to waste too much time on them.

She asks you to find the minimum number of operations required to modify  $s$  so it contains exactly  $k$  instances of B. She also wants you to find these operations (that is, integer  $i$  and character  $c$  selected in each operation).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
```

```

6     cin >> n >> k;
7     string s;
8     cin >> s;
9     int cnt = count(s.begin(), s.end(), 'B');
10    if (cnt == k) {
11        cout << 0 << "\n";
12        return;
13    }
14    for (int i = 0; i < n; i++) {
15        cnt -= (s[i] == 'B');
16        if (cnt == k) {
17            cout << 1 << "\n";
18            cout << i + 1 << " A\n";
19            return;
20        }
21        if (cnt + i + 1 == k) {
22            cout << 1 << "\n";
23            cout << i + 1 << " B\n";
24            return;
25        }
26    }
27 }
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(nullptr);
31     int t;
32     cin >> t;
33     while (t--) {
34         solve();
35     }
36     return 0;
37 }
```

## 72: Perfect Square

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Kristina has a matrix of size  $n$  by  $n$ , filled with lowercase Latin letters. The value of  $n$  is even.

She wants to change some characters so that her matrix becomes a perfect square. A matrix is called a perfect square if it remains unchanged when rotated 90° clockwise once.

Here is an example of rotating a matrix by 90°:

In one operation, Kristina can choose any cell and replace its value with the next character in the alphabet. If the character is equal to “z”, its value does not change.

Find the minimum number of operations required to make the matrix a perfect square.

For example, if the 4 by 4 matrix looks like this:

a	b	b	a
b	c	<b>b</b>	b
b	c	c	b
a	b	b	a

then it is enough to apply 1 operation to the letter b, highlighted in bold.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<string> s(n);
8     for (int i = 0; i < n; i++) {
9         cin >> s[i];
10    }
11    int ans = 0;
12    for (int i = 0; i < n / 2; i++) {
13        for (int j = 0; j < n / 2; j++) {
14            int sum = s[i][j] + s[j][n - 1 - i] + s[n - 1 - i][n - 1 - j] + s[n - 1 - j][i];
15            int max = max({s[i][j], s[j][n - 1 - i], s[n - 1 - i][n - 1 - j], s[n - 1 - j][i]});
16            ans += max * 4 - sum;
17        }
18    }
19    cout << ans << "\n";
20 }
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     int t;
25     cin >> t;
26     while (t--) {
27         solve();
28     }
29     return 0;
30 }
```

### 73: 1D Eraser

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a strip of paper  $s$  that is  $n$  cells long. Each cell is either black or white. In an operation you can take any  $k$  consecutive cells and make them all white.

Find the minimum number of operations needed to remove all black cells.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     string s;
8     cin >> s;
9     int ans = 0;
10    int t = -1;
11    for (int i = 0; i < n; i++) {
12        if (s[i] == 'B' && i > t) {
13            ans++;
14            t = i + k - 1;
15        }
16    }
17    cout << ans << "\n";
18 }
19 int main() {
20     ios::sync_with_stdio(false);
21     cin.tie(nullptr);
22     int t;
23     cin >> t;
24     while (t--) {
25         solve();
26     }
27     return 0;
28 }
```

## 74: Target Practice

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A  $10 \times 10$  target is made out of five “rings” as shown. Each ring has a different point value: the outermost ring - 1 point, the next ring - 2 points, ..., the center ring - 5 points.

Vlad fired several arrows at the target. Help him determine how many points he got.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int ans = 0;
6     for (int i = 0; i < 10; i++) {
7         for (int j = 0; j < 10; j++) {
8             char c;
9             cin >> c;
10            if (c == 'X') {
11                ans += min({i + 1, 10 - i, j + 1, 10 - j});
12            }
13        }
14    }
15    cout << ans << "\n";
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

## 75: Short Sort

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are three cards with letters a, b, c placed in a row in some order. You can do the following operation at most once:

Pick two cards, and swap them.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     string s;
6     cin >> s;
7     if (s == "abc" || s == "acb" || s == "bac" || s == "cba") {
8         cout << "YES\n";
9     } else {
10        cout << "NO\n";
11    }
12 }
```

```

13 int main() {
14     ios::sync_with_stdio(false);
15     cin.tie(nullptr);
16     int t;
17     cin >> t;
18     while (t--) {
19         solve();
20     }
21     return 0;
22 }
```

## 76: Divine Array

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Black is gifted with a Divine array  $a$  consisting of  $n$  ( $1 \leq n \leq 2000$ ) integers. Each position in  $a$  has an initial value. After shouting a curse over the array, it becomes angry and starts an unstoppable transformation.

The transformation consists of infinite steps. Array  $a$  changes at the  $i$ -th step in the following way: for every position  $j$ ,  $a_j$  becomes equal to the number of occurrences of  $a_j$  in  $a$  before starting this step.

Here is an example to help you understand the process better:

In the initial array, we had two 2-s, three 1-s, only one 4 and only one 3, so after the first step, each element became equal to the number of its occurrences in the initial array: all twos changed to 2, all ones changed to 3, four changed to 1 and three changed to 1.

The transformation steps continue forever.

You have to process  $q$  queries: in each query, Black is curious to know the value of  $a_x$  after the  $k$ -th step of transformation.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    int q;
```

```

12     cin >> q;
13     vector<int> ans(q);
14     vector<array<int, 3>> qry(q);
15     for (int i = 0; i < q; i++) {
16         int x, k;
17         cin >> x >> k;
18         x--;
19         k = min(k, 20);
20         qry[i] = {k, x, i};
21     }
22     sort(qry.begin(), qry.end());
23     int cur = 0;
24     for (auto [k, x, i] : qry) {
25         while (cur < k) {
26             vector<int> cnt(n + 1);
27             for (auto x : a) {
28                 cnt[x]++;
29             }
30             for (auto &x : a) {
31                 x = cnt[x];
32             }
33             cur++;
34         }
35         ans[i] = a[x];
36     }
37     for (int i = 0; i < q; i++) {
38         cout << ans[i] << "\n";
39     }
40 }
41 int main() {
42     ios::sync_with_stdio(false);
43     cin.tie(nullptr);
44     int t;
45     cin >> t;
46     while (t--) {
47         solve();
48     }
49     return 0;
50 }
```

## 77: Two Subsequences

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a string  $s$ . You need to find two non-empty strings  $a$  and  $b$  such that the following conditions are satisfied:

Strings  $a$  and  $b$  are both subsequences of  $s$ .

For each index  $i$ , character  $s_i$  of string  $s$  must belong to exactly one of strings  $a$  or  $b$ .

String  $a$  is lexicographically minimum possible; string  $b$  may be any possible string.

Given string  $s$ , print any valid  $a$  and  $b$ .

Reminder:

A string  $a$  ( $b$ ) is a subsequence of a string  $s$  if  $a$  ( $b$ ) can be obtained from  $s$  by deletion of several (possibly, zero) elements. For example, “dores”, “cf”, and “for” are subsequences of “codeforces”, while “decor” and “fork” are not.

A string  $x$  is lexicographically smaller than a string  $y$  if and only if one of the following holds:

$x$  is a prefix of  $y$ , but  $x \neq y$ ;

in the first position where  $x$  and  $y$  differ, the string  $x$  has a letter that appears earlier in the alphabet than the corresponding letter in  $y$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     string s;
6     cin >> s;
7     int x = min_element(s.begin(), s.end()) - s.begin();
8     cout << s[x] << " " << s.substr(0, x) + s.substr(x + 1) << "\n";
9 }
10 int main() {
11     ios::sync_with_stdio(false);
12     cin.tie(nullptr);
13     int t;
14     cin >> t;
15     while (t--) {
16         solve();
17     }
18     return 0;
19 }
```

## 78: The Corridor or There and Back Again

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are in a corridor that extends infinitely to the right, divided into square rooms. You start in room 1, proceed to room  $k$ , and then return to room 1. You can choose the value of  $k$ . Moving to an adjacent room takes 1 second.

Additionally, there are  $n$  traps in the corridor: the  $i$ -th trap is located in room  $d_i$  and will be activated  $s_i$  seconds after you enter the room  $d_i$ . Once a trap is activated, you cannot enter or exit a room with that trap.

Determine the maximum value of  $k$  that allows you to travel from room 1 to room  $k$  and then return to room 1 safely.

For instance, if  $n = 1$  and  $d_1 = 2, s_1 = 2$ , you can proceed to room  $k = 2$  and return safely (the trap will activate at the moment  $1 + s_1 = 1 + 2 = 3$ , it can't prevent you to return back). But if you attempt to reach room  $k = 3$ , the trap will activate at the moment  $1 + s_1 = 1 + 2 = 3$ , preventing your return (you would attempt to enter room 2 on your way back at second 3, but the activated trap would block you). Any larger value for  $k$  is also not feasible. Thus, the answer is  $k = 2$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     int ans = 1E9;
8     for (int i = 0; i < n; i++) {
9         int d, s;
10        cin >> d >> s;
11        ans = min(ans, d + (s - 1) / 2);
12    }
13    cout << ans << "\n";
14 }
15 int main() {
16     ios::sync_with_stdio(false);
17     cin.tie(nullptr);
18     int t;
19     cin >> t;
20     while (t--) {
21         solve();
22     }
23     return 0;
24 }
```

## 79: Life of a Flower

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Petya has got an interesting flower. Petya is a busy person, so he sometimes forgets to water it. You are given  $n$  days from Petya's live and you have to determine what happened with his flower in the end.

The flower grows as follows:

If the flower isn't watered for two days in a row, it dies.

If the flower is watered in the  $i$ -th day, it grows by 1 centimeter.

If the flower is watered in the  $i$ -th and in the  $(i - 1)$ -th day ( $i > 1$ ), then it grows by 5 centimeters instead of 1.

If the flower is not watered in the  $i$ -th day, it does not grow.

At the beginning of the 1-st day the flower is 1 centimeter tall. What is its height after  $n$  days?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    int ans = 1;
12    for (int i = 0; i < n; i++) {
13        if (a[i]) {
14            if (i && a[i - 1]) {
15                ans += 5;
16            } else {
17                ans += 1;
18            }
19        } else if (i && !a[i - 1]) {
20            ans = -1;
21            break;
22        }
23    }
24    cout << ans << "\n";
25 }
26 int main() {
27     ios::sync_with_stdio(false);
28     cin.tie(nullptr);
29     int t;
30     cin >> t;
31     while (t--) {
32         solve();
33     }
34     return 0;
35 }
```

## 80: Game

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are playing a very popular computer game. The next level consists of  $n$  consecutive locations, numbered from 1 to  $n$ , each of them containing either land or water. It is known that the first and last locations contain land, and for completing the level you have to move from the first location to the last. Also, if you become inside a location with water, you will die, so you can only move between locations with land.

You can jump between adjacent locations for free, as well as no more than once jump from any location with land  $i$  to any location with land  $i + x$ , spending  $x$  coins ( $x \geq 0$ ).

Your task is to spend the minimum possible number of coins to move from the first location to the last one.

Note that this is always possible since both the first and last locations are the land locations.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    int l = 0, r = n - 1;
12    while (l < r && a[l + 1] == 1) {
13        l++;
14    }
15    while (r > l && a[r - 1] == 1) {
16        r--;
17    }
18    int ans = r - l;
19    cout << ans << "\n";
20 }
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     int t;
25     cin >> t;
26     while (t--) {
27         solve();
```

```

28     }
29     return 0;
30 }
```

**math****81: Two Divisors**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A certain number  $1 \leq x \leq 10^9$  is chosen. You are given two integers  $a$  and  $b$ , which are the two largest divisors of the number  $x$ . At the same time, the condition  $1 \leq a < b < x$  is satisfied.

For the given numbers  $a, b$ , you need to find the value of  $x$ .

† The number  $y$  is a divisor of the number  $x$  if there is an integer  $k$  such that  $x = y \cdot k$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int a, b;
6     cin >> a >> b;
7     if (b % a == 0) {
8         cout << b / a * b << "\n";
9     } else {
10        cout << lcm(a, b) << "\n";
11    }
12 }
13 int main() {
14     ios::sync_with_stdio(false);
15     cin.tie(nullptr);
16     int t;
17     cin >> t;
18     while (t--) {
19         solve();
20     }
21     return 0;
22 }
```

**82: Least Product**

- Time limit: 1 second

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array of integers  $a_1, a_2, \dots, a_n$ . You can perform the following operation any number of times (possibly zero):

Choose any element  $a_i$  from the array and change its value to any integer between 0 and  $a_i$  (inclusive). More formally, if  $a_i < 0$ , replace  $a_i$  with any integer in  $[a_i, 0]$ , otherwise replace  $a_i$  with any integer in  $[0, a_i]$ .

Let  $r$  be the minimum possible product of all the  $a_i$  after performing the operation any number of times.

Find the minimum number of operations required to make the product equal to  $r$ . Also, print one such shortest sequence of operations. If there are multiple answers, you can print any of them.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     int sgn = 1;
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11         if (a[i] < 0) {
12             sgn *= -1;
13         } else if (a[i] == 0) {
14             sgn = 0;
15         }
16     }
17     if (sgn <= 0) {
18         cout << 0 << "\n";
19         return;
20     }
21     cout << 1 << "\n";
22     cout << 1 << " " << 0 << "\n";
23 }
24 int main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
27     int t;
28     cin >> t;
29     while (t--) {
30         solve();
31     }
32     return 0;
33 }
```

### 83: Make Almost Equal With Mod

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a_1, a_2, \dots, a_n$  of distinct positive integers. You have to do the following operation exactly once:

choose a positive integer  $k$ ;

for each  $i$  from 1 to  $n$ , replace  $a_i$  with  $a_i \bmod k^\dagger$ .

Find a value of  $k$  such that  $1 \leq k \leq 10^{18}$  and the array  $a_1, a_2, \dots, a_n$  contains exactly 2 distinct values at the end of the operation. It can be shown that, under the constraints of the problem, at least one such  $k$  always exists. If there are multiple solutions, you can print any of them.

$\dagger a \bmod b$  denotes the remainder after dividing  $a$  by  $b$ . For example:

$7 \bmod 3 = 1$  since  $7 = 3 \cdot 2 + 1$

$15 \bmod 4 = 3$  since  $15 = 4 \cdot 3 + 3$

$21 \bmod 1 = 0$  since  $21 = 21 \cdot 1 + 0$

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<i64> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    i64 k = 1;
12    while (true) {
13        for (int i = 0; i < n; i++) {
14            if (a[i] % k != a[0] % k) {
15                cout << k << "\n";
16                return;
17            }
18        }
19        k *= 2;
20    }
21 }
22 int main() {
23     ios::sync_with_stdio(false);
24     cin.tie(nullptr);
25     int t;

```

```

26     cin >> t;
27     while (t--) {
28         solve();
29     }
30     return 0;
31 }
```

## 84: Preparing for the Contest

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp is practicing for a big contest. He plans to solve  $n$  problems to make sure he's prepared. Each of these problems has a difficulty level: the first problem has a difficulty level of 1, the second problem has a difficulty level of 2, and so on, until the last ( $n$ -th) problem, which has a difficulty level of  $n$ .

Monocarp will choose some order in which he is going to solve all  $n$  problems. Whenever he solves a problem which is more difficult than the last problem he solved, he gets excited because he feels like he's progressing. He doesn't get excited when he solves the first problem in his chosen order.

For example, if Monocarp solves the problems in the order [3, 5, 4, 1, 6, 2], he gets excited twice (the corresponding problems are underlined).

Monocarp wants to get excited exactly  $k$  times during his practicing session. Help him to choose the order in which he has to solve the problems!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     for (int i = n - k; i <= n; i++) {
8         cout << i << " ";
9     }
10    for (int i = n - k - 1; i > 0; i--) {
11        cout << i << " ";
12    }
13    cout << "\n";
14 }
15 int main() {
16     ios::sync_with_stdio(false);
```

```

17     cin.tie(nullptr);
18     int t;
19     cin >> t;
20     while (t--) {
21         solve();
22     }
23     return 0;
24 }
```

## 85: Constructive Problems

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Gridlandia has been hit by flooding and now has to reconstruct all of its cities. Gridlandia can be described by an  $n \times m$  matrix.

Initially, all of its cities are in economic collapse. The government can choose to rebuild certain cities. Additionally, any collapsed city which has at least one vertically neighboring rebuilt city and at least one horizontally neighboring rebuilt city can ask for aid from them and become rebuilt without help from the government. More formally, collapsed city positioned in  $(i, j)$  can become rebuilt if both of the following conditions are satisfied:

At least one of cities with positions  $(i + 1, j)$  and  $(i - 1, j)$  is rebuilt;

At least one of cities with positions  $(i, j + 1)$  and  $(i, j - 1)$  is rebuilt.

If the city is located on the border of the matrix and has only one horizontally or vertically neighbouring city, then we consider only that city.

The government wants to know the minimum number of cities it has to rebuild such that after some time all the cities can be rebuilt.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     int ans = max(n, m);
8     cout << ans << "\n";
9 }
10 int main() {
11     ios::sync_with_stdio(false);
```

```

12     cin.tie(nullptr);
13     int t;
14     cin >> t;
15     while (t--) {
16         solve();
17     }
18     return 0;
19 }
```

## 86: Laura and Operations

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Laura is a girl who does not like combinatorics. Nemanja will try to convince her otherwise.

Nemanja wrote some digits on the board. All of them are either 1, 2, or 3. The number of digits 1 is  $a$ . The number of digits 2 is  $b$  and the number of digits 3 is  $c$ . He told Laura that in one operation she can do the following:

Select two different digits and erase them from the board. After that, write the digit (1, 2, or 3) different from both erased digits.

For example, let the digits be 1, 1, 1, 2, 3, 3. She can choose digits 1 and 3 and erase them. Then the board will look like this 1, 1, 2, 3. After that, she has to write another digit 2, so at the end of the operation, the board will look like 1, 1, 2, 3, 2.

Nemanja asked her whether it was possible for only digits of one type to remain written on the board after some operations. If so, which digits can they be?

Laura was unable to solve this problem and asked you for help. As an award for helping her, she will convince Nemanja to give you some points.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int a, b, c;
6     cin >> a >> b >> c;
7     cout << (b + c + 1) % 2 << " " << (c + a + 1) % 2 << " " << (a + b + 1) % 2 << "\n";
8 }
9 int main() {
10     ios::sync_with_stdio(false);
11     cin.tie(nullptr);
```

```

12     int t;
13     cin >> t;
14     while (t--) {
15         solve();
16     }
17     return 0;
18 }
```

## 87: Game with Integers

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vanya and Vova are playing a game. Players are given an integer  $n$ . On their turn, the player can add 1 to the current integer or subtract 1. The players take turns; Vanya starts. If after Vanya's move the integer is divisible by 3, then he wins. If 10 moves have passed and Vanya has not won, then Vova wins.

Write a program that, based on the integer  $n$ , determines who will win if both players play optimally.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     if (n % 3 != 0) {
8         cout << "First\n";
9     } else {
10        cout << "Second\n";
11    }
12 }
13 int main() {
14     ios::sync_with_stdio(false);
15     cin.tie(nullptr);
16     int t;
17     cin >> t;
18     while (t--) {
19         solve();
20     }
21     return 0;
22 }
```

## 88: Treasure Chest

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Monocarp has found a treasure map. The map represents the treasure location as an OX axis. Monocarp is at 0, the treasure chest is at  $x$ , the key to the chest is at  $y$ .

Obviously, Monocarp wants to open the chest. He can perform the following actions:

go 1 to the left or 1 to the right (spending 1 second);

pick the key or the chest up if he is in the same point as that object (spending 0 seconds);

put the chest down in his current point (spending 0 seconds);

open the chest if he's in the same point as the chest and has picked the key up (spending 0 seconds).

Monocarp can carry the chest, but the chest is pretty heavy. He knows that he can carry it for at most  $k$  seconds in total (putting it down and picking it back up doesn't reset his stamina).

What's the smallest time required for Monocarp to open the chest?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int x, y, k;
6     cin >> x >> y >> k;
7     int ans;
8     if (x > y) {
9         ans = x;
10    } else {
11        ans = y + max(0, y - x - k);
12    }
13    cout << ans << "\n";
14 }
15 int main() {
16     ios::sync_with_stdio(false);
17     cin.tie(nullptr);
18     int t;
19     cin >> t;
20     while (t--) {
21         solve();
22     }
23     return 0;
24 }
```

## 89: Deja Vu

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  of length  $n$ , consisting of positive integers, and an array  $x$  of length  $q$ , also consisting of positive integers.

There are  $q$  modification. On the  $i$ -th modification ( $1 \leq i \leq q$ ), for each  $j$  ( $1 \leq j \leq n$ ), such that  $a_j$  is divisible by  $2^{x_i}$ , you add  $2^{x_i-1}$  to  $a_j$ . Note that  $x_i$  ( $1 \leq x_i \leq 30$ ) is a positive integer not exceeding 30.

After all modification queries, you need to output the final array.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, q;
6     cin >> n >> q;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    array<int, 30> f{};
12    for (int i = 0; i < 30; i++) {
13        f[i] = 1 << i;
14    }
15    while (q--) {
16        int x;
17        cin >> x;
18        for (int i = 0; i < 30; i++) {
19            if ((f[i] & ((1 << x) - 1)) == 0) {
20                f[i] += 1 << (x - 1);
21            }
22        }
23    }
24    for (int i = 0; i < n; i++) {
25        int k = __builtin_ctz(a[i]);
26        a[i] += f[k] - (1 << k);
27        cout << a[i] << " \n"[i == n - 1];
28    }
29}
30 int main() {
31     ios::sync_with_stdio(false);
32     cin.tie(nullptr);
33     int t;
34     cin >> t;
35     while (t--) {
36         solve();
37     }

```

```

38     return 0;
39 }
```

## 90: Hemose Shopping

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Hemose was shopping with his friends Samez, AhmedZ, AshrafEzz, TheSawan and O\_E in Germany. As you know, Hemose and his friends are problem solvers, so they are very clever. Therefore, they will go to all discount markets in Germany.

Hemose has an array of  $n$  integers. He wants Samez to sort the array in the non-decreasing order. Since it would be a too easy problem for Samez, Hemose allows Samez to use only the following operation:

Choose indices  $i$  and  $j$  such that  $1 \leq i, j \leq n$ , and  $|i - j| \geq x$ . Then, swap elements  $a_i$  and  $a_j$ .

Choose indices  $i$  and  $j$  such that  $1 \leq i, j \leq n$ , and  $|i - j| \geq x$ . Then, swap elements  $a_i$  and  $a_j$ .

Can you tell Samez if there's a way to sort the array in the non-decreasing order by using the operation written above some finite number of times (possibly 0)?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, x;
6     cin >> n >> x;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    if (n - x <= x - 1) {
12        int l = n - x, r = x;
13        rotate(a.begin() + l, a.begin() + r, a.end());
14        sort(a.begin(), a.end() - (r - l));
15        rotate(a.begin() + l, a.end() - (r - l), a.end());
16        if (is_sorted(a.begin(), a.end())) {
17            cout << "YES\n";
18        } else {
19            cout << "NO\n";
20        }
21    } else {
22        cout << "YES\n";
23    }
24 }
25 int main() {
```

```

26     ios::sync_with_stdio(false);
27     cin.tie(nullptr);
28     int t;
29     cin >> t;
30     while (t--) {
31         solve();
32     }
33     return 0;
34 }
```

## 91: Special Numbers

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Theofanis really likes sequences of positive integers, thus his teacher (Yeltsa Kcir) gave him a problem about a sequence that consists of only special numbers.

Let's call a positive number special if it can be written as a sum of different non-negative powers of  $n$ . For example, for  $n = 4$  number 17 is special, because it can be written as  $4^0 + 4^2 = 1 + 16 = 17$ , but 9 is not.

Theofanis asks you to help him find the  $k$ -th special number if they are sorted in increasing order. Since this number may be too large, output it modulo  $10^9 + 7$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
```

```

24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 void solve() {
35     int n, k;
36     cin >> n >> k;
37     Z ans = 0;
38     Z p = 1;
39     for (int i = 0; i < 30; i++) {
40         if (k >> i & 1) {
41             ans += p;
42         }
43         p *= n;
44     }
45     cout << ans << "\n";
46 }
47 int main() {
48     ios::sync_with_stdio(false);
49     cin.tie(nullptr);
50     int t;
51     cin >> t;
52     while (t--) {
53         solve();
54     }
55     return 0;
56 }
```

## 92: Consecutive Sum Riddle

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Theofanis has a riddle for you and if you manage to solve it, he will give you a Cypriot snack halloumi for free (Cypriot cheese).

You are given an integer  $n$ . You need to find two integers  $l$  and  $r$  such that  $-10^{18} \leq l < r \leq 10^{18}$  and  $l + (l + 1) + \dots + (r - 1) + r = n$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
```

```

5     i64 n;
6     cin >> n;
7     cout << -n + 1 << " " << n << "\n";
8 }
9 int main() {
10    ios::sync_with_stdio(false);
11    cin.tie(nullptr);
12    int t;
13    cin >> t;
14    while (t--) {
15        solve();
16    }
17    return 0;
18 }
```

### 93: Three Threadlets

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Once upon a time, bartender Decim found three threadlets and a pair of scissors.

In one operation, Decim chooses any threadlet and cuts it into two threadlets, whose lengths are positive integers and their sum is equal to the length of the threadlet being cut.

For example, he can cut a threadlet of length 5 into threadlets of lengths 2 and 3, but he cannot cut it into threadlets of lengths 2.5 and 2.5, or lengths 0 and 5, or lengths 3 and 4.

Decim can perform at most three operations. He is allowed to cut the threadlets obtained from previous cuts. Will he be able to make all the threadlets of equal length?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int a, b, c;
6     cin >> a >> b >> c;
7     for (auto x : {a, a / 2, a / 3, a / 4}) {
8         if (x == 0) {
9             continue;
10        }
11        bool ok = true;
12        int cnt = 0;
13        for (auto y : {a, b, c}) {
14            if (y % x != 0) {
15                ok = false;
16            }
17        }
18        if (ok) {
19            cout << "YES" << endl;
20        } else {
21            cout << "NO" << endl;
22        }
23    }
24 }
```

```

16         break;
17     }
18     cnt += y / x - 1;
19   }
20   if (ok && cnt <= 3) {
21     cout << "YES\n";
22     return;
23   }
24 }
25 cout << "NO\n";
26 }
27 int main() {
28   ios::sync_with_stdio(false);
29   cin.tie(nullptr);
30   int t;
31   cin >> t;
32   while (t--) {
33     solve();
34   }
35   return 0;
36 }
```

## 94: Fear of the Dark

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp tries to get home from work. He is currently at the point  $O = (0, 0)$  of a two-dimensional plane; his house is at the point  $P = (P_x, P_y)$ .

Unfortunately, it is late in the evening, so it is very dark. Monocarp is afraid of the darkness. He would like to go home along a path illuminated by something.

Thankfully, there are two lanterns, located in the points  $A = (A_x, A_y)$  and  $B = (B_x, B_y)$ . You can choose any non-negative number  $w$  and set the power of both lanterns to  $w$ . If a lantern's power is set to  $w$ , it illuminates a circle of radius  $w$  centered at the lantern location (including the borders of the circle).

You have to choose the minimum non-negative value  $w$  for the power of the lanterns in such a way that there is a path from the point  $O$  to the point  $P$  which is completely illuminated. You may assume that the lanterns don't interfere with Monocarp's movement.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
```

```

2  using namespace std;
3  using i64 = long long;
4  # struct Point;
5  double dist(const Point &a, const Point &b) {
6      return hypot(a.x - b.x, a.y - b.y);
7  }
8  void solve() {
9      Point O, P, A, B;
10     cin >> P.x >> P.y >> A.x >> A.y >> B.x >> B.y;
11     double ans = min({
12         max(dist(O, A), dist(A, P)),
13         max(dist(O, B), dist(B, P)),
14         max({dist(O, A), dist(A, B) / 2, dist(B, P)}),
15         max({dist(O, B), dist(B, A) / 2, dist(A, P)}),
16     });
17     cout << ans << "\n";
18 }
19 int main() {
20     ios::sync_with_stdio(false);
21     cin.tie(nullptr);
22     cout << fixed << setprecision(10);
23     int t;
24     cin >> t;
25     while (t--) {
26         solve();
27     }
28     return 0;
29 }
```

## 95: Sum of Three

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp has an integer  $n$ .

He wants to represent his number as a sum of three distinct positive integers  $x$ ,  $y$ , and  $z$ . Additionally, Monocarp wants none of the numbers  $x$ ,  $y$ , and  $z$  to be divisible by 3.

Your task is to help Monocarp to find any valid triplet of distinct positive integers  $x$ ,  $y$ , and  $z$ , or report that such a triplet does not exist.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  using i64 = long long;
4  void solve() {
5      int n;
```

```

6     cin >> n;
7     for (int x = 1; x <= 10; x++) {
8         for (int y = x + 1; y <= 10; y++) {
9             int z = n - x - y;
10            if (x % 3 != 0 && y % 3 != 0 && z % 3 != 0 && z > y) {
11                cout << "YES\n";
12                cout << x << " " << y << " " << z << "\n";
13                return;
14            }
15        }
16    }
17    cout << "NO\n";
18 }
19 int main() {
20     ios::sync_with_stdio(false);
21     cin.tie(nullptr);
22     int t;
23     cin >> t;
24     while (t--) {
25         solve();
26     }
27     return 0;
28 }
```

## 96: Joyboard

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Chaneka, a gamer kid, invented a new gaming controller called joyboard. Interestingly, the joyboard she invented can only be used to play one game.

The joyboard has a screen containing  $n + 1$  slots numbered from 1 to  $n + 1$  from left to right. The  $n + 1$  slots are going to be filled with an array of non-negative integers  $[a_1, a_2, a_3, \dots, a_{n+1}]$ . Chaneka, as the player, must assign  $a_{n+1}$  with an integer between 0 and  $m$  inclusive. Then, for each  $i$  from  $n$  to 1, the value of  $a_i$  will be equal to the remainder of dividing  $a_{i+1}$  (the adjacent value to the right) by  $i$ . In other words,  $a_i = a_{i+1} \bmod i$ .

Chaneka wants it such that after every slot is assigned with an integer, there are exactly  $k$  distinct values in the entire screen (among all  $n + 1$  slots). How many valid ways are there for assigning a non-negative integer into slot  $n + 1$ ?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
```

```

5     int n, m, k;
6     cin >> n >> m >> k;
7     if (k == 1) {
8         cout << 1 << "\n";
9     } else if (k == 2) {
10        cout << min(m, n - 1) + m / n << "\n";
11    } else if (k == 3) {
12        cout << m - min(m, n - 1) - m / n << "\n";
13    } else {
14        cout << 0 << "\n";
15    }
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

## 97: Goals of Victory

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  teams in a football tournament. Each pair of teams match up once. After every match, Pak Chanek receives two integers as the result of the match, the number of goals the two teams score during the match. The efficiency of a team is equal to the total number of goals the team scores in each of its matches minus the total number of goals scored by the opponent in each of its matches.

After the tournament ends, Pak Dengklek counts the efficiency of every team. Turns out that he forgot about the efficiency of one of the teams. Given the efficiency of  $n - 1$  teams  $a_1, a_2, a_3, \dots, a_{n-1}$ . What is the efficiency of the missing team? It can be shown that the efficiency of the missing team can be uniquely determined.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     int ans = 0;
8     for (int i = 0; i < n - 1; i++) {
9         int x;
```

```

10         cin >> x;
11         ans -= x;
12     }
13     cout << ans << "\n";
14 }
15 int main() {
16     ios::sync_with_stdio(false);
17     cin.tie(nullptr);
18     int t;
19     cin >> t;
20     while (t--) {
21         solve();
22     }
23     return 0;
24 }
```

## 98: Vasilije in Cacak

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Aca and Milovan, two fellow competitive programmers, decided to give Vasilije a problem to test his skills.

Vasilije is given three positive integers:  $n$ ,  $k$ , and  $x$ , and he has to determine if he can choose  $k$  distinct integers between 1 and  $n$ , such that their sum is equal to  $x$ .

Since Vasilije is now in the weirdest city in Serbia where Aca and Milovan live, Cacak, the problem seems weird to him. So he needs your help with this problem.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     i64 x;
7     cin >> n >> k >> x;
8     if (x >= 1LL * k * (k + 1) / 2 && x <= 1LL * k * (n + n - k + 1) / 2) {
9         cout << "YES\n";
10    } else {
11        cout << "NO\n";
12    }
13 }
14 int main() {
15     ios::sync_with_stdio(false);
16     cin.tie(nullptr);
17     int t;
18     cin >> t;
```

```

19     while (t--) {
20         solve();
21     }
22     return 0;
23 }
```

**99: Aleksa and Stack**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

After the Serbian Informatics Olympiad, Aleksa was very sad, because he didn't win a medal (he didn't know stack), so Vasilije came to give him an easy problem, just to make his day better.

Vasilije gave Aleksa a positive integer  $n$  ( $n \geq 3$ ) and asked him to construct a strictly increasing array of size  $n$  of positive integers, such that

$3 \cdot a_{i+2}$  is not divisible by  $a_i + a_{i+1}$  for each  $i$  ( $1 \leq i \leq n - 2$ ).

Since Aleksa thinks he is a bad programmer now, he asked you to help him find such an array.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     for (int i = 0; i < n; i++) {
8         cout << int(5E8) + i << " \n"[i == n - 1];
9     }
10 }
11 int main() {
12     ios::sync_with_stdio(false);
13     cin.tie(nullptr);
14     int t;
15     cin >> t;
16     while (t--) {
17         solve();
18     }
19     return 0;
20 }
```

**100: Good Kid**

- Time limit: 1 second

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Slavic is preparing a present for a friend's birthday. He has an array  $a$  of  $n$  digits and the present will be the product of all these digits. Because Slavic is a good kid who wants to make the biggest product possible, he wants to add 1 to exactly one of his digits.

What is the maximum product Slavic can make?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    sort(a.begin(), a.end());
12    a[0]++;
13    int ans = 1;
14    for (int i = 0; i < n; i++) {
15        ans *= a[i];
16    }
17    cout << ans << "\n";
18 }
19 int main() {
20     ios::sync_with_stdio(false);
21     cin.tie(nullptr);
22     int t;
23     cin >> t;
24     while (t--) {
25         solve();
26     }
27     return 0;
28 }
```

## misc

### 101: LOL Lovers

- Time limit: 3 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  food items lying in a row on a long table. Each of these items is either a loaf of bread (denoted as a capital Latin letter ‘L’ with ASCII code 76) or an onion (denoted as a capital Latin letter ‘O’ with ASCII code 79). There is at least one loaf of bread and at least one onion on the table.

You and your friend want to divide the food on the table: you will take a prefix of this row (several leftmost items), and the friend will take the rest. However, there are several restrictions:

Each person should have at least one item.

The number of your loaves should differ from the number of your friend’s loaves.

The number of your onions should differ from the number of your friend’s onions.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     string s;
10    cin >> s;
11    vector<array<int, 2>> pre(n + 1);
12    for (int i = 0; i < n; i++) {
13        pre[i + 1] = pre[i];
14        if (s[i] == 'O') {
15            pre[i + 1][0] += 1;
16        } else {
17            pre[i + 1][1] += 1;
18        }
19    }
20    for (int i = 1; i < n; i++) {
21        if (pre[i][0] * 2 != pre[n][0] && pre[i][1] * 2 != pre[n][1]) {
22            cout << i << "\n";
23            return 0;
24        }
25    }
26    cout << -1 << "\n";
27    return 0;
28 }
```

## 102: Swap and Delete

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a binary string  $s$  (a string consisting only of 0-s and 1-s).

You can perform two types of operations on  $s$ :

delete one character from  $s$ . This operation costs 1 coin;

swap any pair of characters in  $s$ . This operation is free (costs 0 coins).

You can perform these operations any number of times and in any order.

Let's name a string you've got after performing operations above as  $t$ . The string  $t$  is good if for each  $i$  from 1 to  $|t|$   $t_i \neq s_i$  ( $|t|$  is the length of the string  $t$ ). The empty string is always good. Note that you are comparing the resulting string  $t$  with the initial string  $s$ .

What is the minimum total cost to make the string  $t$  good?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     string s;
6     cin >> s;
7     array<int, 2> cnt{};
8     for (auto x : s) {
9         cnt[x - '0'] += 1;
10    }
11    int n = s.size();
12    int ans = n;
13    auto cur = cnt;
14    for (int i = n - 1; i >= 0; i--) {
15        if (cur[0] <= cnt[1] && cur[1] <= cnt[0]) {
16            ans = n - 1 - i;
17            break;
18        }
19        cur[s[i] - '0'] -= 1;
20    }
21    cout << ans << "\n";
22 }
23 int main() {
24     ios::sync_with_stdio(false);
25     cin.tie(nullptr);
26     int t;
27     cin >> t;
28     while (t--) {
29         solve();
30     }
31     return 0;
32 }
```

### 103: Increasing Sequence

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a sequence  $a_1, a_2, \dots, a_n$ . A sequence  $b_1, b_2, \dots, b_n$  is called good, if it satisfies all of the following conditions:

$b_i$  is a positive integer for  $i = 1, 2, \dots, n$ ;

$b_i \neq a_i$  for  $i = 1, 2, \dots, n$ ;

$b_1 < b_2 < \dots < b_n$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     int x = 1;
8     for (int i = 0; i < n; i++) {
9         int a;
10        cin >> a;
11        if (x == a) {
12            x++;
13        }
14        x++;
15    }
16    cout << x - 1 << "\n";
17 }
18 int main() {
19     ios::sync_with_stdio(false);
20     cin.tie(nullptr);
21     int t;
22     cin >> t;
23     while (t--) {
24         solve();
25     }
26     return 0;
27 }
```

### 104: Rigged!

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

Monocarp organizes a weightlifting competition. There are  $n$  athletes participating in the competition, the  $i$ -th athlete has strength  $s_i$  and endurance  $e_i$ . The 1-st athlete is Monocarp's friend Polycarp, and Monocarp really wants Polycarp to win.

The competition will be conducted as follows. The jury will choose a positive (greater than zero) integer  $w$ , which denotes the weight of the barbell that will be used in the competition. The goal for each athlete is to lift the barbell as many times as possible. The athlete who lifts the barbell the most amount of times will be declared the winner (if there are multiple such athletes - there's no winner).

If the barbell's weight  $w$  is strictly greater than the strength of the  $i$ -th athlete  $s_i$ , then the  $i$ -th athlete will be unable to lift the barbell even one single time. Otherwise, the  $i$ -th athlete will be able to lift the barbell, and the number of times he does it will be equal to his endurance  $e_i$ .

For example, suppose there are 4 athletes with parameters  $s_1 = 7, e_1 = 4; s_2 = 9, e_2 = 3; s_3 = 4, e_3 = 6; s_4 = 2, e_4 = 2$ . If the weight of the barbell is 5, then:

the first athlete will be able to lift the barbell 4 times;

the second athlete will be able to lift the barbell 3 times;

the third athlete will be unable to lift the barbell;

the fourth athlete will be unable to lift the barbell.

Monocarp wants to choose  $w$  in such a way that Polycarp (the 1-st athlete) wins the competition. Help him to choose the value of  $w$ , or report that it is impossible.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> s(n), e(n);
8     bool ok = true;
9     for (int i = 0; i < n; i++) {
10         cin >> s[i] >> e[i];
11         if (i > 0 && s[i] >= s[0] && e[i] >= e[0]) {
12             ok = false;
13         }
14     }
15     cout << (ok ? s[0] : -1) << "\n";
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);

```

```

20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

## 105: Friendly Arrays

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given two arrays of integers -  $a_1, \dots, a_n$  of length  $n$ , and  $b_1, \dots, b_m$  of length  $m$ . You can choose any element  $b_j$  from array  $b$  ( $1 \leq j \leq m$ ), and for all  $1 \leq i \leq n$  perform  $a_i = a_i | b_j$ . You can perform any number of such operations.

After all the operations, the value of  $x = a_1 \oplus a_2 \oplus \dots \oplus a_n$  will be calculated. Find the minimum and maximum values of  $x$  that could be obtained after performing any set of operations.

Above,  $|$  is the bitwise OR operation, and  $\oplus$  is the bitwise XOR operation.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    int o = 0;
12    for (int i = 0; i < m; i++) {
13        int b;
14        cin >> b;
15        o |= b;
16    }
17    int x = 0, y = 0;
18    for (int i = 0; i < n; i++) {
19        x ^= a[i];
20        y ^= (a[i] | o);
21    }
22    if (x > y) {
23        swap(x, y);
24    }
}
```

```

25     cout << x << " " << y << "\n";
26 }
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(nullptr);
30     int t;
31     cin >> t;
32     while (t--) {
33         solve();
34     }
35     return 0;
36 }
```

## 106: Luntik and Subsequences

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Luntik came out for a morning stroll and found an array  $a$  of length  $n$ . He calculated the sum  $s$  of the elements of the array ( $s = \sum_{i=1}^n a_i$ ). Luntik calls a subsequence of the array  $a$  nearly full if the sum of the numbers in that subsequence is equal to  $s - 1$ .

Luntik really wants to know the number of nearly full subsequences of the array  $a$ . But he needs to come home so he asks you to solve that problem!

A sequence  $x$  is a subsequence of a sequence  $y$  if  $x$  can be obtained from  $y$  by deletion of several (possibly, zero or all) elements.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     int cnt[2] {};
8     for (int i = 0; i < n; i++) {
9         int a;
10        cin >> a;
11        if (a <= 1) {
12            cnt[a]++;
13        }
14    }
15    i64 ans = (1LL * cnt[1]) << cnt[0];
16    cout << ans << "\n";
17 }
```

```

18 int main() {
19     ios::sync_with_stdio(false);
20     cin.tie(nullptr);
21     int t;
22     cin >> t;
23     while (t--) {
24         solve();
25     }
26     return 0;
27 }
```

## 107: Luntik and Concerts

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Luntik has decided to try singing. He has  $a$  one-minute songs,  $b$  two-minute songs and  $c$  three-minute songs. He wants to distribute all songs into two concerts such that every song should be included to exactly one concert.

He wants to make the absolute difference of durations of the concerts as small as possible. The duration of the concert is the sum of durations of all songs in that concert.

Please help Luntik and find the minimal possible difference in minutes between the concerts durations.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int a, b, c;
6     cin >> a >> b >> c;
7     if (c % 2 == 1 && a == 0 && b == 0) {
8         cout << 3 << "\n";
9     } else if ((c % 2 == 1 && a == 1 && b == 0) || (c % 2 == 0 && a == 0 && b == 1)) {
10        cout << 2 << "\n";
11    } else if ((a + c) % 2 == 1) {
12        cout << 1 << "\n";
13    } else {
14        cout << 0 << "\n";
15    }
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
```

```

20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

**109: Era**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Shohag has an integer sequence  $a_1, a_2, \dots, a_n$ . He can perform the following operation any number of times (possibly, zero):

Select any positive integer  $k$  (it can be different in different operations).

Choose any position in the sequence (possibly the beginning or end of the sequence, or in between any two elements) and insert  $k$  into the sequence at this position.

This way, the sequence  $a$  changes, and the next operation is performed on this changed sequence.

For example, if  $a = [3, 3, 4]$  and he selects  $k = 2$ , then after the operation he can obtain one of the sequences  $[2, 3, 3, 4], [3, 2, 3, 4], [3, 3, 2, 4]$ , or  $[3, 3, 4, 2]$ .

Shohag wants this sequence to satisfy the following condition: for each  $1 \leq i \leq |a|$ ,  $a_i \leq i$ . Here,  $|a|$  denotes the size of  $a$ .

Help him to find the minimum number of operations that he has to perform to achieve this goal. We can show that under the constraints of the problem it's always possible to achieve this goal in a finite number of operations.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     int ans = 0;
8     for (int i = 1; i <= n; i++) {
9         int a;
10        cin >> a;
11        ans = max(ans, a - i);
12    }
}
```

```

13     cout << ans << "\n";
14 }
15 int main() {
16     ios::sync_with_stdio(false);
17     cin.tie(nullptr);
18     int t;
19     cin >> t;
20     while (t--) {
21         solve();
22     }
23     return 0;
24 }
```

## 110: A.M. Deviation

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A number  $a_2$  is said to be the arithmetic mean of two numbers  $a_1$  and  $a_3$ , if the following condition holds:  $a_1 + a_3 = 2 \cdot a_2$ .

We define an arithmetic mean deviation of three numbers  $a_1$ ,  $a_2$  and  $a_3$  as follows:  $d(a_1, a_2, a_3) = |a_1 + a_3 - 2 \cdot a_2|$ .

Arithmetic means a lot to Jeevan. He has three numbers  $a_1$ ,  $a_2$  and  $a_3$  and he wants to minimize the arithmetic mean deviation  $d(a_1, a_2, a_3)$ . To do so, he can perform the following operation any number of times (possibly zero):

Choose  $i, j$  from  $\{1, 2, 3\}$  such that  $i \neq j$  and increment  $a_i$  by 1 and decrement  $a_j$  by 1

Help Jeevan find out the minimum value of  $d(a_1, a_2, a_3)$  that can be obtained after applying the operation any number of times.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int a, b, c;
6     cin >> a >> b >> c;
7     cout << ((a + b + c) % 3 != 0) << "\n";
8 }
9 int main() {
10     ios::sync_with_stdio(false);
11     cin.tie(nullptr);
```

```

12     int t;
13     cin >> t;
14     while (t--) {
15         solve();
16     }
17     return 0;
18 }
```

### 111: Coloring Rectangles

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

David was given a red checkered rectangle of size  $n \times m$ . But he doesn't like it. So David cuts the original or any other rectangle piece obtained during the cutting into two new pieces along the grid lines. He can do this operation as many times as he wants.

As a result, he will get a set of rectangles. Rectangles  $1 \times 1$  are forbidden.

David also knows how to paint the cells blue. He wants each rectangle from the resulting set of pieces to be colored such that any pair of adjacent cells by side (from the same piece) have different colors.

What is the minimum number of cells David will have to paint?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     int ans = (n * m - 1) / 3 + 1;
8     cout << ans << "\n";
9 }
10 int main() {
11     ios::sync_with_stdio(false);
12     cin.tie(nullptr);
13     int t;
14     cin >> t;
15     while (t--) {
16         solve();
17     }
18     return 0;
19 }
```

## 112: Mathematical Addition

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Ivan decided to prepare for the test on solving integer equations. He noticed that all tasks in the test have the following form:

You are given two positive integers  $u$  and  $v$ , find any pair of integers (not necessarily positive)  $x, y$ , such that:

$$\frac{x}{u} + \frac{y}{v} = \frac{x+y}{u+v}.$$

The solution  $x = 0, y = 0$  is forbidden, so you should find any solution with  $(x, y) \neq (0, 0)$ .

Please help Ivan to solve some equations of this form.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int u, v;
6     cin >> u >> v;
7     cout << 1LL * u * u << " " << -1LL * v * v << "\n";
8 }
9 int main() {
10     ios::sync_with_stdio(false);
11     cin.tie(nullptr);
12     int t;
13     cin >> t;
14     while (t--) {
15         solve();
16     }
17     return 0;
18 }
```

## 113: Plus Minus Permutation

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given 3 integers -  $n, x, y$ . Let's call the score of a permutation<sup>†</sup>  $p_1, \dots, p_n$  the following value:

$$(p_{1 \cdot x} + p_{2 \cdot x} + \dots + p_{\lfloor \frac{n}{x} \rfloor \cdot x}) - (p_{1 \cdot y} + p_{2 \cdot y} + \dots + p_{\lfloor \frac{n}{y} \rfloor \cdot y})$$

In other words, the score of a permutation is the sum of  $p_i$  for all indices  $i$  divisible by  $x$ , minus the sum of  $p_i$  for all indices  $i$  divisible by  $y$ .

You need to find the maximum possible score among all permutations of length  $n$ .

For example, if  $n = 7, x = 2, y = 3$ , the maximum score is achieved by the permutation [2, 6, 1, 7, 5, 4, 3] and is equal to  $(6 + 7 + 4) - (1 + 4) = 17 - 5 = 12$ .

<sup>†</sup> A permutation of length  $n$  is an array consisting of  $n$  distinct integers from 1 to  $n$  in any order. For example, [2, 3, 1, 5, 4] is a permutation, but [1, 2, 2] is not a permutation (the number 2 appears twice in the array) and [1, 3, 4] is also not a permutation ( $n = 3$ , but the array contains 4).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, x, y;
6     cin >> n >> x >> y;
7     i64 l = lcm(1LL * x, 1LL * y);
8     int pos = n / x - n / l;
9     int neg = n / y - n / l;
10    i64 ans = 1LL * (n + n - pos + 1) * pos / 2 - 1LL * (neg + 1) * neg / 2;
11    cout << ans << "\n";
12 }
13 int main() {
14     ios::sync_with_stdio(false);
15     cin.tie(nullptr);
16     int t;
17     cin >> t;
18     while (t--) {
19         solve();
20     }
21     return 0;
22 }
```

## 114: Non-coprime Split

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given two integers  $l \leq r$ . You need to find positive integers  $a$  and  $b$  such that the following conditions are simultaneously satisfied:

$$l \leq a + b \leq r$$

$$\gcd(a, b) \neq 1$$

or report that they do not exist.

$\gcd(a, b)$  denotes the greatest common divisor of numbers  $a$  and  $b$ . For example,  $\gcd(6, 9) = 3$ ,  $\gcd(8, 9) = 1$ ,  $\gcd(4, 2) = 2$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int l, r;
6     cin >> l >> r;
7     for (int i = max(4, l); i <= r; i++) {
8         for (int j = 2; j * j <= i; j++) {
9             if (i % j == 0) {
10                 cout << j << " " << i - j << "\n";
11                 return;
12             }
13         }
14     }
15     cout << -1 << "\n";
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

## 115: GCD Arrays

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Consider the array  $a$  composed of all the integers in the range  $[l, r]$ . For example, if  $l = 3$  and  $r = 7$ , then  $a = [3, 4, 5, 6, 7]$ .

Given  $l$ ,  $r$ , and  $k$ , is it possible for  $\gcd(a)$  to be greater than 1 after doing the following operation at most  $k$  times?

Choose 2 numbers from  $a$ .

Permanently remove one occurrence of each of them from the array.

Insert their product back into  $a$ .

$\gcd(b)$  denotes the greatest common divisor (GCD) of the integers in  $b$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int l, r, k;
6     cin >> l >> r >> k;
7     bool ok;
8     if (l == r) {
9         if (l != 1) {
10             ok = true;
11         } else {
12             ok = false;
13         }
14     } else if (k >= (r + 1) / 2 - l / 2) {
15         ok = true;
16     } else {
17         ok = false;
18     }
19     cout << (ok ? "YES" : "NO") << "\n";
20 }
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     int t;
25     cin >> t;
26     while (t--) {
27         solve();
28     }
29     return 0;
30 }
```

## 116: Fun with Even Subarrays

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  of  $n$  elements. You can apply the following operation to it any number of times:

Select some subarray from  $a$  of even size  $2k$  that begins at position  $l$  ( $1 \leq l \leq l + 2 \cdot k - 1 \leq n, k \geq 1$ ) and for each  $i$  between 0 and  $k - 1$  (inclusive), assign the value  $a_{l+k+i}$  to  $a_{l+i}$ .

For example, if  $a = [2, 1, 3, 4, 5, 3]$ , then choose  $l = 1$  and  $k = 2$ , applying this operation the array will become  $a = [3, 4, 3, 4, 5, 3]$ .

Find the minimum number of operations (possibly zero) needed to make all the elements of the array equal.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    int v = a[n - 1];
12    int l = n;
13    while (l > 0 && a[l - 1] == v) {
14        l--;
15    }
16    int ans = 0;
17    while (l > 0) {
18        l = max(0, 2 * l - n);
19        while (l > 0 && a[l - 1] == v) {
20            l--;
21        }
22        ans++;
23    }
24    cout << ans << "\n";
25 }
26 int main() {
27     ios::sync_with_stdio(false);
28     cin.tie(nullptr);
29     int t;
30     cin >> t;
31     while (t--) {
32         solve();
33     }
34     return 0;
35 }
```

## 117: Min Max Swap

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given two arrays  $a$  and  $b$  of  $n$  positive integers each. You can apply the following operation to them any number of times:

Select an index  $i$  ( $1 \leq i \leq n$ ) and swap  $a_i$  with  $b_i$  (i. e.  $a_i$  becomes  $b_i$  and vice versa).

Find the minimum possible value of  $\max(a_1, a_2, \dots, a_n) \cdot \max(b_1, b_2, \dots, b_n)$  you can get after applying such operation any number of times (possibly zero).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<int> b(n);
12    for (int i = 0; i < n; i++) {
13        cin >> b[i];
14    }
15    for (int i = 0; i < n; i++) {
16        if (a[i] > b[i]) {
17            swap(a[i], b[i]);
18        }
19    }
20    int ans = *max_element(a.begin(), a.end())
21    * max_element(b.begin(), b.end());
22    cout << ans << "\n";
23 }
24 int main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
27     int t;
28     cin >> t;
29     while (t--) {
30         solve();
31     }
32     return 0;
33 }
```

## 118: Power Walking

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Sam is a kindergartener, and there are  $n$  children in his group. He decided to create a team with some of his children to play “brawl:go 2”.

Sam has  $n$  power-ups, the  $i$ -th has type  $a_i$ . A child's strength is equal to the number of different types among power-ups he has.

For a team of size  $k$ , Sam will distribute all  $n$  power-ups to  $k$  children in such a way that each of the  $k$  children receives at least one power-up, and each power-up is given to someone.

For each integer  $k$  from 1 to  $n$ , find the minimum sum of strengths of a team of  $k$  children Sam can get.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    set s(a.begin(), a.end());
12    int dist = s.size();
13    for (int k = 1; k <= n; k++) {
14        cout << max(dist, k) << " \n"[k == n];
15    }
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

## 119: Hard Way

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Sam lives in Awesomeburg, its downtown has a triangular shape. Also, the following is true about the triangle:

its vertices have integer coordinates,

the coordinates of vertices are non-negative, and

its vertices are not on a single line.

He calls a point on the downtown's border (that is the border of the triangle) safe if he can reach this point from at least one point of the line  $y = 0$  walking along some straight line, without crossing the interior of the triangle.

Find the total length of the unsafe parts of the downtown border. It can be proven that these parts are segments and their number is finite.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int x[3], y[3];
6     for (int i = 0; i < 3; i++) {
7         cin >> x[i] >> y[i];
8     }
9     int ans = 0;
10    for (int i = 0; i < 3; i++) {
11        if (y[0] == y[1] && y[0] > y[2]) {
12            ans += abs(x[0] - x[1]);
13        }
14        rotate(x, x + 1, x + 3);
15        rotate(y, y + 1, y + 3);
16    }
17    cout << ans << "\n";
18 }
19 int main() {
20     ios::sync_with_stdio(false);
21     cin.tie(nullptr);
22     int t;
23     cin >> t;
24     while (t--) {
25         solve();
26     }
27     return 0;
28 }
```

## 120: Madoka and Math Dad

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Madoka finally found the administrator password for her computer. Her father is a well-known popularizer of mathematics, so the password is the answer to the following problem.

Find the maximum decimal number without zeroes and with no equal digits in a row, such that the sum of its digits is  $n$ .

Madoka is too tired of math to solve it herself, so help her to solve this problem!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     int t = n % 3 == 1 ? 1 : 2;
8     while (n > 0) {
9         cout << t;
10        n -= t;
11        t = 3 - t;
12    }
13    cout << "\n";
14 }
15 int main() {
16     ios::sync_with_stdio(false);
17     cin.tie(nullptr);
18     int t;
19     cin >> t;
20     while (t--) {
21         solve();
22     }
23     return 0;
24 }
```

## 121: Direction Change

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a grid with  $n$  rows and  $m$  columns. Rows and columns are numbered from 1 to  $n$ , and from 1 to  $m$ . The intersection of the  $a$ -th row and  $b$ -th column is denoted by  $(a, b)$ .

Initially, you are standing in the top left corner  $(1, 1)$ . Your goal is to reach the bottom right corner  $(n, m)$ .

You can move in four directions from  $(a, b)$ : up to  $(a - 1, b)$ , down to  $(a + 1, b)$ , left to  $(a, b - 1)$  or right to  $(a, b + 1)$ .

You cannot move in the same direction in two consecutive moves, and you cannot leave the grid. What is the minimum number of moves to reach  $(n, m)$ ?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     int ans;
8     if (n == 1 && m == 1) {
9         ans = 0;
10    } else if (n + m == 3) {
11        ans = 1;
12    } else if (n == 1 || m == 1) {
13        ans = -1;
14    } else {
15        ans = max(n, m) * 2 - 3;
16        if (ans % 2 != (n + m) % 2) {
17            ans++;
18        }
19    }
20    cout << ans << "\n";
21 }
22 int main() {
23     ios::sync_with_stdio(false);
24     cin.tie(nullptr);
25     int t;
26     cin >> t;
27     while (t--) {
28         solve();
29     }
30     return 0;
31 }
```

## 122: Difference Operations

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  consisting of  $n$  positive integers.

You are allowed to perform this operation any number of times (possibly, zero):

choose an index  $i$  ( $2 \leq i \leq n$ ), and change  $a_i$  to  $a_i - a_{i-1}$ .

Is it possible to make  $a_i = 0$  for all  $2 \leq i \leq n$ ?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     bool ok = true;
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11         if (a[i] % a[0] != 0) {
12             ok = false;
13         }
14     }
15     cout << (ok ? "YES" : "NO") << "\n";
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

### 123: Ambitious Kid

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Chaneka, Pak Chanek's child, is an ambitious kid, so Pak Chanek gives her the following problem to test her ambition.

Given an array of integers  $[A_1, A_2, A_3, \dots, A_N]$ . In one operation, Chaneka can choose one element, then increase or decrease the element's value by 1. Chaneka can do that operation multiple times, even for different elements.

What is the minimum number of operations that must be done to make it such that  $A_1 \times A_2 \times A_3 \times \dots \times A_N = 0$ ?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int N;
8     cin >> N;
9     int ans = 1E9;
10    for (int i = 0; i < N; i++) {
11        int x;
12        cin >> x;
13        ans = min(ans, abs(x));
14    }
15    cout << ans << "\n";
16    return 0;
17 }
```

## 124: MEX Repetition

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a_1, a_2, \dots, a_n$  of pairwise distinct integers from 0 to  $n$ . Consider the following operation:

consecutively for each  $i$  from 1 to  $n$  in this order, replace  $a_i$  with  $\text{MEX}(a_1, a_2, \dots, a_n)$ .

Here  $\text{MEX}$  of a collection of integers  $c_1, c_2, \dots, c_m$  is defined as the smallest non-negative integer  $x$  which does not occur in the collection  $c$ . For example,  $\text{MEX}(0, 2, 2, 1, 4) = 3$  and  $\text{MEX}(1, 2) = 0$ .

Print the array after applying  $k$  such operations.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> a(n);
8     i64 b = 1LL * n * (n + 1) / 2;
9     for (int i = 0; i < n; i++) {
10        cin >> a[i];
11        b -= a[i];
12    }
13    a.push_back(b);
14    k %= (n + 1);
15    rotate(a.begin(), a.end() - k, a.end());
```

```

16     for (int i = 0; i < n; i++) {
17         cout << a[i] << " \n"[i == n - 1];
18     }
19 }
20 int main() {
21     ios::sync_with_stdio(false);
22     cin.tie(nullptr);
23     int t;
24     cin >> t;
25     while (t--) {
26         solve();
27     }
28     return 0;
29 }
```

## 125: Channel

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Petya is an administrator of a channel in one of the messengers. A total of  $n$  people are subscribed to his channel, and Petya is not considered a subscriber.

Petya has published a new post on the channel. At the moment of the publication, there were  $a$  subscribers online. We assume that every subscriber always reads all posts in the channel if they are online.

After this, Petya starts monitoring the number of subscribers online. He consecutively receives  $q$  notifications of the form “a subscriber went offline” or “a subscriber went online”. Petya does not know which exact subscriber goes online or offline. It is guaranteed that such a sequence of notifications could have indeed been received.

Petya wonders if all of his subscribers have read the new post. Help him by determining one of the following:

it is impossible that all  $n$  subscribers have read the post;

it is possible that all  $n$  subscribers have read the post;

it is guaranteed that all  $n$  subscribers have read the post.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
```

```

4 void solve() {
5     int n, a, q;
6     cin >> n >> a >> q;
7     bool ok = (a == n);
8     string s;
9     cin >> s;
10    int sum = a;
11    for (int i = 0; i < q; i++) {
12        a += (s[i] == '+' ? 1 : -1);
13        sum += (s[i] == '+' ? 1 : 0);
14        ok |= a == n;
15    }
16    if (ok) {
17        cout << "YES\n";
18    } else if (sum < n) {
19        cout << "NO\n";
20    } else {
21        cout << "MAYBE\n";
22    }
23 }
24 int main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
27     int t;
28     cin >> t;
29     while (t--) {
30         solve();
31     }
32     return 0;
33 }
```

## 126: Gift Carpet

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Recently, Tema and Vika celebrated Family Day. Their friend Arina gave them a carpet, which can be represented as an  $n \cdot m$  table of lowercase Latin letters.

Vika hasn't seen the gift yet, but Tema knows what kind of carpets she likes. Vika will like the carpet if she can read her name on. She reads column by column from left to right and chooses one or zero letters from current column.

Formally, the girl will like the carpet if it is possible to select four distinct columns in order from left to right such that the first column contains “v”, the second one contains “i”, the third one contains “k”, and the fourth one contains “a”.

Help Tema understand in advance whether Vika will like Arina's gift.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<string> s(n);
8     for (int i = 0; i < n; i++) {
9         cin >> s[i];
10    }
11    string t = "vika";
12    array<bool, 5> dp{};
13    dp[0] = 1;
14    for (int i = 0; i < m; i++) {
15        array<bool, 4> b{};
16        for (int j = 0; j < n; j++) {
17            int x = t.find(s[j][i]);
18            if (x != -1) {
19                b[x] = true;
20            }
21        }
22        for (int x = 3; x >= 0; x--) {
23            if (b[x]) {
24                dp[x + 1] |= dp[x];
25            }
26        }
27    }
28    cout << (dp[4] ? "YES" : "NO") << "\n";
29 }
30 int main() {
31     ios::sync_with_stdio(false);
32     cin.tie(nullptr);
33     int t;
34     cin >> t;
35     while (t--) {
36         solve();
37     }
38     return 0;
39 }
```

## 127: Boxes Packing

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Mishka has got  $n$  empty boxes. For every  $i$  ( $1 \leq i \leq n$ ),  $i$ -th box is a cube with side length  $a_i$ .

Mishka can put a box  $i$  into another box  $j$  if the following conditions are met:

$i$ -th box is not put into another box;

j-th box doesn't contain any other boxes;

box i is smaller than box j ( $a_i < a_j$ ).

Mishka can put boxes into each other an arbitrary number of times. He wants to minimize the number of visible boxes. A box is called visible iff it is not put into some another box.

Help Mishka to determine the minimum possible number of visible boxes!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     map<int, int> cnt;
10    for (int i = 0; i < n; i++) {
11        int a;
12        cin >> a;
13        cnt[a]++;
14    }
15    int ans = 0;
16    for (auto [a, c] : cnt) {
17        ans = max(ans, c);
18    }
19    cout << ans << "\n";
20    return 0;
21 }
```

## 128: The Modcrab

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vova is again playing some computer game, now an RPG. In the game Vova's character received a quest: to slay the fearsome monster called Modcrab.

After two hours of playing the game Vova has tracked the monster and analyzed its tactics. The Modcrab has  $h_2$  health points and an attack power of  $a_2$ . Knowing that, Vova has decided to buy a lot of strong healing potions and to prepare for battle.

Vova's character has  $h_1$  health points and an attack power of  $a_1$ . Also he has a large supply of healing potions, each of which increases his current amount of health points by  $c_1$  when Vova drinks a potion. All potions are identical to each other. It is guaranteed that  $c_1 > a_2$ .

The battle consists of multiple phases. In the beginning of each phase, Vova can either attack the monster (thus reducing its health by  $a_1$ ) or drink a healing potion (it increases Vova's health by  $c_1$ ; Vova's health can exceed  $h_1$ ). Then, if the battle is not over yet, the Modcrab attacks Vova, reducing his health by  $a_2$ . The battle ends when Vova's (or Modcrab's) health drops to 0 or lower. It is possible that the battle ends in a middle of a phase after Vova's attack.

Of course, Vova wants to win the fight. But also he wants to do it as fast as possible. So he wants to make up a strategy that will allow him to win the fight after the minimum possible number of phases.

Help Vova to make up a strategy! You may assume that Vova never runs out of healing potions, and that he can always win.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int h1, a1, c1;
8     cin >> h1 >> a1 >> c1;
9     int h2, a2;
10    cin >> h2 >> a2;
11    vector<string> ans;
12    while (h2 > 0) {
13        if (a1 >= h2 || h1 > a2) {
14            ans.push_back("STRIKE");
15            h2 -= a1;
16            h1 -= a2;
17        } else {
18            ans.push_back("HEAL");
19            h1 += c1 - a2;
20        }
21    }
22    cout << ans.size() << "\n";
23    for (auto s : ans) {
24        cout << s << "\n";
25    }
26    return 0;
27 }
```

## 129: Hungry Student Problem

- Time limit: 1 second

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Ivan's classes at the university have just finished, and now he wants to go to the local CFK cafe and eat some fried chicken.

CFK sells chicken chunks in small and large portions. A small portion contains 3 chunks; a large one - 7 chunks. Ivan wants to eat exactly  $x$  chunks. Now he wonders whether he can buy exactly this amount of chicken.

Formally, Ivan wants to know if he can choose two non-negative integers  $a$  and  $b$  in such a way that  $a$  small portions and  $b$  large ones contain exactly  $x$  chunks.

Help Ivan to answer this question for several values of  $x$ !

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int x;
6     cin >> x;
7     if (set{1, 2, 4, 5, 8, 11}.count(x)) {
8         cout << "NO\n";
9     } else {
10        cout << "YES\n";
11    }
12 }
13 int main() {
14     ios::sync_with_stdio(false);
15     cin.tie(nullptr);
16     int t;
17     cin >> t;
18     while (t--) {
19         solve();
20     }
21     return 0;
22 }
```

## 130: Find Extra One

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have  $n$  distinct points on a plane, none of them lie on OY axis. Check that there is a point after removal of which the remaining points are located on one side of the OY axis.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     int pos = 0;
10    for (int i = 0; i < n; i++) {
11        int x, y;
12        cin >> x >> y;
13        pos += (x > 0);
14    }
15    if (pos <= 1 || pos >= n - 1) {
16        cout << "Yes\n";
17    } else {
18        cout << "No\n";
19    }
20    return 0;
21 }
```

## sortings

### 131: Building an Aquarium

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You love fish, that's why you have decided to build an aquarium. You have a piece of coral made of  $n$  columns, the  $i$ -th of which is  $a_i$  units tall. Afterwards, you will build a tank around the coral as follows:

Pick an integer  $h \geq 1$  - the height of the tank. Build walls of height  $h$  on either side of the tank.

Then, fill the tank up with water so that the height of each column is  $h$ , unless the coral is taller than  $h$ ; then no water should be added to this column.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, x;
6     cin >> n >> x;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    sort(a.begin(), a.end());
12    i64 sum = 0;
13    i64 ans = 0;
14    for (int i = 0; i < n; i++) {
15        if (i && sum + 1LL * i * (a[i] - a[i - 1]) > x) {
16            ans = a[i - 1] + (x - sum) / i;
17            break;
18        }
19        if (i) {
20            sum += 1LL * i * (a[i] - a[i - 1]);
21        }
22        if (i == n - 1) {
23            ans = a[i] + (x - sum) / n;
24        }
25    }
26    cout << ans << "\n";
27 }
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(nullptr);
31     int t;
32     cin >> t;
33     while (t--) {
34         solve();
35     }
36     return 0;
37 }
```

## 132: 2D Traveling

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Piggy lives on an infinite plane with the Cartesian coordinate system on it.

There are  $n$  cities on the plane, numbered from 1 to  $n$ , and the first  $k$  cities are defined as major cities. The coordinates of the  $i$ -th city are  $(x_i, y_i)$ .

Piggy, as a well-experienced traveller, wants to have a relaxing trip after Zhongkao examination. Currently, he is in city  $a$ , and he wants to travel to city  $b$  by air. You can fly between any two cities, and you can visit several cities in any order while travelling, but the final destination must be city  $b$ .

Because of active trade between major cities, it's possible to travel by plane between them for free. Formally, the price of an air ticket  $f(i, j)$  between two cities  $i$  and  $j$  is defined as follows:

$$f(i, j) = \begin{cases} 0, & \text{if cities } i \text{ and } j \text{ are both major cities} \\ |x_i - x_j| + |y_i - y_j|, & \text{otherwise} \end{cases}$$

Piggy doesn't want to save time, but he wants to save money. So you need to tell him the minimum value of the total cost of all air tickets if he can take any number of flights.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k, a, b;
6     cin >> n >> k >> a >> b;
7     a--, b--;
8     vector<int> x(n), y(n);
9     for (int i = 0; i < n; i++) {
10         cin >> x[i] >> y[i];
11     }
12     i64 ans = 1LL * abs(x[a] - x[b]) + abs(y[a] - y[b]);
13     i64 da = 1E18, db = 1E18;
14     for (int i = 0; i < k; i++) {
15         da = min(da, 1LL * abs(x[a] - x[i]) + abs(y[a] - y[i]));
16         db = min(db, 1LL * abs(x[b] - x[i]) + abs(y[b] - y[i]));
17     }
18     ans = min(ans, da + db);
19     cout << ans << "\n";
20 }
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     int t;
25     cin >> t;
26     while (t--) {
27         solve();
28     }
29     return 0;
30 }
```

### 133: Social Distance

- Time limit: 1.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

$m$  chairs are arranged in a circle sequentially. The chairs are numbered from 0 to  $m - 1$ .  $n$  people want to sit in these chairs. The  $i$ -th of them wants at least  $a[i]$  empty chairs both on his right and left side.

More formally, if the  $i$ -th person sits in the  $j$ -th chair, then no one else should sit in the following chairs:  $(j - a[i]) \bmod m, (j - a[i] + 1) \bmod m, \dots, (j + a[i] - 1) \bmod m, (j + a[i]) \bmod m$ .

Decide if it is possible to sit down for all of them, under the given limitations.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    i64 ans = accumulate(a.begin(), a.end(), 0LL);
12    ans += *max_element(a.begin(), a.end());
13    ans -= *min_element(a.begin(), a.end());
14    ans += n;
15    cout << (ans <= m ? "YES" : "NO") << "\n";
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

### 134: Split Sort

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a permutation<sup>†</sup>  $p_1, p_2, \dots, p_n$  of integers 1 to  $n$ .

You can change the current permutation by applying the following operation several (possibly, zero) times:

choose some  $x$  ( $2 \leq x \leq n$ );

create a new permutation by: first, writing down all elements of  $p$  that are less than  $x$ , without changing their order; second, writing down all elements of  $p$  that are greater than or equal to  $x$ , without changing their order;

first, writing down all elements of  $p$  that are less than  $x$ , without changing their order;

second, writing down all elements of  $p$  that are greater than or equal to  $x$ , without changing their order;

replace  $p$  with the newly created permutation.

For example, if the permutation used to be  $[6, 4, 3, 5, 2, 1]$  and you choose  $x = 4$ , then you will first write down  $[3, 2, 1]$ , then append this with  $[6, 4, 5]$ . So the initial permutation will be replaced by  $[3, 2, 1, 6, 4, 5]$ .

Find the minimum number of operations you need to achieve  $p_i = i$  for  $i = 1, 2, \dots, n$ . We can show that it is always possible to do so.

<sup>†</sup> A permutation of length  $n$  is an array consisting of  $n$  distinct integers from 1 to  $n$  in arbitrary order. For example,  $[2, 3, 1, 5, 4]$  is a permutation, but  $[1, 2, 2]$  is not a permutation (2 appears twice in the array), and  $[1, 3, 4]$  is also not a permutation ( $n = 3$  but there is 4 in the array).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> p(n);
8     for (int i = 0; i < n; i++) {
9         int x;
10        cin >> x;
11        x--;
12        p[x] = i;
13    }
14    int ans = 0;
15    for (int i = 1; i < n; i++) {
16        ans += (p[i - 1] > p[i]);
17    }
18    cout << ans << "\n";
19 }
20 int main() {
21     ios::sync_with_stdio(false);
22     cin.tie(nullptr);
23     int t;
24     cin >> t;
25     while (t--) {
26         solve();
27     }
28     return 0;
29 }
```

### 135: Flower City Fence

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Anya lives in the Flower City. By order of the city mayor, she has to build a fence for herself.

The fence consists of  $n$  planks, each with a height of  $a_i$  meters. According to the order, the heights of the planks must not increase. In other words, it is true that  $a_i \geq a_j$  for all  $i < j$ .

Anya became curious whether her fence is symmetrical with respect to the diagonal. In other words, will she get the same fence if she lays all the planks horizontally in the same order.

For example, for  $n = 5$ ,  $a = [5, 4, 3, 2, 1]$ , the fence is symmetrical. Because if all the planks are laid horizontally, the fence will be  $[5, 4, 3, 2, 1]$ , as shown in the diagram.

On the left is the fence  $[5, 4, 3, 2, 1]$ , on the right is the same fence laid horizontally

But for  $n = 3$ ,  $a = [4, 2, 1]$ , the fence is not symmetrical. Because if all the planks are laid horizontally, the fence will be  $[3, 2, 1, 1]$ , as shown in the diagram.

On the left is the fence  $[4, 2, 1]$ , on the right is the same fence laid horizontally

Help Anya and determine whether her fence is symmetrical.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    for (int i = 0, j = n; i < n; i++) {
12        while (j > 0 && a[j - 1] <= i) {
13            j--;
14        }
15        if (a[i] != j) {
16            cout << "NO\n";
17            return;
18        }
19    }
20    cout << "YES\n";
21 }
```

```

22 int main() {
23     ios::sync_with_stdio(false);
24     cin.tie(nullptr);
25     int t;
26     cin >> t;
27     while (t--) {
28         solve();
29     }
30     return 0;
31 }
```

### 136: Destroyer

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

John is a lead programmer on a destroyer belonging to the space navy of the Confederacy of Independent Operating Systems. One of his tasks is checking if the electronic brains of robots were damaged during battles.

A standard test is to order the robots to form one or several lines, in each line the robots should stand one after another. After that, each robot reports the number of robots standing in front of it in its line.

The  $i$ -th robot reported number  $l_i$ . Unfortunately, John does not know which line each robot stands in, and can't check the reported numbers. Please determine if it is possible to form the lines in such a way that all reported numbers are correct, or not.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> cnt(n + 1);
8     for (int i = 0; i < n; i++) {
9         int x;
10        cin >> x;
11        cnt[min(n, x)]++;
12    }
13    for (int i = 1; i <= n; i++) {
14        if (cnt[i] > cnt[i - 1]) {
15            cout << "NO\n";
16            return;
17        }
18    }
19    cout << "YES\n";
20 }
```

```

21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     int t;
25     cin >> t;
26     while (t--) {
27         solve();
28     }
29     return 0;
30 }
```

### 137: Assembly via Minimums

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Sasha has an array  $a$  of  $n$  integers. He got bored and for all  $i, j$  ( $i < j$ ), he wrote down the minimum value of  $a_i$  and  $a_j$ . He obtained a new array  $b$  of size  $\frac{n \cdot (n - 1)}{2}$ .

For example, if  $a = [2, 3, 5, 1]$ , he would write  $[\min(2, 3), \min(2, 5), \min(2, 1), \min(3, 5), \min(3, 1), \min(5, 1)] = [2, 2, 1, 3, 1, 1]$ .

Then, he randomly shuffled all the elements of the array  $b$ .

Unfortunately, he forgot the array  $a$ , and your task is to restore any possible array  $a$  from which the array  $b$  could have been obtained.

The elements of array  $a$  should be in the range  $[-10^9, 10^9]$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n * (n - 1) / 2);
8     for (int i = 0; i < n * (n - 1) / 2; i++) {
9         cin >> a[i];
10    }
11    sort(a.begin(), a.end());
12    for (int i = 0; i < n - 1; i++) {
13        cout << a[(n - 1 + n - i) * i / 2] << " ";
14    }
15    cout << a.back() << "\n";
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
```

```

20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

**138: Monsters**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp is playing yet another computer game. And yet again, his character is killing some monsters. There are  $n$  monsters, numbered from 1 to  $n$ , and the  $i$ -th of them has  $a_i$  health points initially.

Monocarp's character has an ability that deals  $k$  damage to the monster with the highest current health. If there are several of them, the one with the smaller index is chosen. If a monster's health becomes less than or equal to 0 after Monocarp uses his ability, then it dies.

Monocarp uses his ability until all monsters die. Your task is to determine the order in which monsters will die.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<int> p(n);
12    iota(p.begin(), p.end(), 0);
13    stable_sort(p.begin(), p.end(),
14                [&](int i, int j) {
15                    return (a[i] - 1) % k > (a[j] - 1) % k;
16                });
17    for (int i = 0; i < n; i++) {
18        cout << p[i] + 1 << " \n"[i == n - 1];
19    }
20 }
21 int main() {
22     ios::sync_with_stdio(false);
```

```

23     cin.tie(nullptr);
24     int t;
25     cin >> t;
26     while (t--) {
27         solve();
28     }
29     return 0;
30 }
```

**139: Parity Sort**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have an array of integers  $a$  of length  $n$ . You can apply the following operation to the given array:

Swap two elements  $a_i$  and  $a_j$  such that  $i \neq j$ ,  $a_i$  and  $a_j$  are either both even or both odd.

Determine whether it is possible to sort the array in non-decreasing order by performing the operation any number of times (possibly zero).

For example, let  $a = [7, 10, 1, 3, 2]$ . Then we can perform 3 operations to sort the array:

Swap  $a_3 = 1$  and  $a_1 = 7$ , since 1 and 7 are odd. We get  $a = [1, 10, 7, 3, 2]$ ;

Swap  $a_2 = 10$  and  $a_5 = 2$ , since 10 and 2 are even. We get  $a = [1, 2, 7, 3, 10]$ ;

Swap  $a_4 = 3$  and  $a_3 = 7$ , since 3 and 7 are odd. We get  $a = [1, 2, 3, 7, 10]$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    auto b = a;
12    sort(b.begin(), b.end());
13    for (int i = 0; i < n; i++) {
14        if ((a[i] - b[i]) % 2) {
15            cout << "NO\n";
16            return;
17    }}
```

```

18     }
19     cout << "YES\n";
20 }
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     int t;
25     cin >> t;
26     while (t--) {
27         solve();
28     }
29     return 0;
30 }
```

## 140: Ten Words of Wisdom

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In the game show “Ten Words of Wisdom”, there are  $n$  participants numbered from 1 to  $n$ , each of whom submits one response. The  $i$ -th response is  $a_i$  words long and has quality  $b_i$ . No two responses have the same quality, and at least one response has length at most 10.

The winner of the show is the response which has the highest quality out of all responses that are not longer than 10 words. Which response is the winner?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     int x = 0, q = 0;
8     for (int i = 1; i <= n; i++) {
9         int a, b;
10        cin >> a >> b;
11        if (a <= 10 && b > q) {
12            q = b;
13            x = i;
14        }
15    }
16    cout << x << "\n";
17 }
18 int main() {
19     ios::sync_with_stdio(false);
20     cin.tie(nullptr);
21     int t;
22     cin >> t;
23     while (t--) {
```

```

24         solve();
25     }
26     return 0;
27 }
```

**141: To My Critics**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Suneet has three digits  $a$ ,  $b$ , and  $c$ .

Since math isn't his strongest point, he asks you to determine if you can choose any two digits to make a sum greater or equal to 10.

Output "YES" if there is such a pair, and "NO" otherwise.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int a, b, c;
6     cin >> a >> b >> c;
7     if (max({a + b, b + c, c + a}) >= 10) {
8         cout << "YES\n";
9     } else {
10        cout << "NO\n";
11    }
12 }
13 int main() {
14     ios::sync_with_stdio(false);
15     cin.tie(nullptr);
16     int t;
17     cin >> t;
18     while (t--) {
19         solve();
20     }
21     return 0;
22 }
```

**142: The Man who became a God**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

Kars is tired and resentful of the narrow mindset of his village since they are content with staying where they are and are not trying to become the perfect life form. Being a top-notch inventor, Kars wishes to enhance his body and become the perfect life form. Unfortunately,  $n$  of the villagers have become suspicious of his ideas. The  $i$ -th villager has a suspicion of  $a_i$  on him. Individually each villager is scared of Kars, so they form into groups to be more powerful.

The power of the group of villagers from  $l$  to  $r$  be defined as  $f(l, r)$  where

$$f(l, r) = |a_l - a_{l+1}| + |a_{l+1} - a_{l+2}| + \dots + |a_{r-1} - a_r|.$$

Here  $|x - y|$  is the absolute value of  $x - y$ . A group with only one villager has a power of 0.

Kars wants to break the villagers into exactly  $k$  contiguous subgroups so that the sum of their power is minimized. Formally, he must find  $k - 1$  positive integers  $1 \leq r_1 < r_2 < \dots < r_{k-1} < n$  such that  $f(1, r_1) + f(r_1 + 1, r_2) + \dots + f(r_{k-1} + 1, n)$  is minimised. Help Kars in finding the minimum value of  $f(1, r_1) + f(r_1 + 1, r_2) + \dots + f(r_{k-1} + 1, n)$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<int> d(n - 1);
12    for (int i = 0; i < n - 1; i++) {
13        d[i] = abs(a[i + 1] - a[i]);
14    }
15    sort(d.begin(), d.end());
16    int ans = 0;
17    for (int i = 0; i < n - k; i++) {
18        ans += d[i];
19    }
20    cout << ans << "\n";
21 }
22 int main() {
23     ios::sync_with_stdio(false);
24     cin.tie(nullptr);
25     int t;
26     cin >> t;
27     while (t--) {
28         solve();
29     }

```

```

30     return 0;
31 }
```

### 143: Vika and the Bridge

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In the summer, Vika likes to visit her country house. There is everything for relaxation: comfortable swings, bicycles, and a river.

There is a wooden bridge over the river, consisting of  $n$  planks. It is quite old and unattractive, so Vika decided to paint it. And in the shed, they just found cans of paint of  $k$  colors.

After painting each plank in one of  $k$  colors, Vika was about to go swinging to take a break from work. However, she realized that the house was on the other side of the river, and the paint had not yet completely dried, so she could not walk on the bridge yet.

In order not to spoil the appearance of the bridge, Vika decided that she would still walk on it, but only stepping on planks of the same color. Otherwise, a small layer of paint on her sole will spoil the plank of another color. Vika also has a little paint left, but it will only be enough to repaint one plank of the bridge.

Now Vika is standing on the ground in front of the first plank. To walk across the bridge, she will choose some planks of the same color (after repainting), which have numbers  $1 \leq i_1 < i_2 < \dots < i_m \leq n$  (planks are numbered from 1 from left to right). Then Vika will have to cross  $i_1 - 1, i_2 - i_1 - 1, i_3 - i_2 - 1, \dots, i_m - i_{m-1} - 1, n - i_m$  planks as a result of each of  $m + 1$  steps.

Since Vika is afraid of falling, she does not want to take too long steps. Help her and tell her the minimum possible maximum number of planks she will have to cross in one step, if she can repaint one (or zero) plank a different color while crossing the bridge.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> c(n);
```

```

8     vector<int> lst(k, -1);
9     vector<vector<int>> f(k);
10    for (int i = 0; i < n; i++) {
11        cin >> c[i];
12        c[i]--;
13        f[c[i]].push_back(i - 1 - lst[c[i]]);
14        lst[c[i]] = i;
15    }
16    int ans = n;
17    for (int i = 0; i < k; i++) {
18        f[i].push_back(n - 1 - lst[i]);
19        sort(f[i].begin(), f[i].end(), greater());
20        int res = f[i][0] / 2;
21        if (f[i].size() > 1) {
22            res = max(res, f[i][1]);
23        }
24        ans = min(ans, res);
25    }
26    cout << ans << "\n";
27 }
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(nullptr);
31     int t;
32     cin >> t;
33     while (t--) {
34         solve();
35     }
36     return 0;
37 }
```

## 144: Rudolf and the Another Competition

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Rudolf has registered for a programming competition that will follow the rules of ICPC. The rules imply that for each solved problem, a participant gets 1 point, and also incurs a penalty equal to the number of minutes passed from the beginning of the competition to the moment of solving the problem. In the final table, the participant with the most points is ranked higher, and in case of a tie in points, the participant with the lower penalty is ranked higher.

In total,  $n$  participants have registered for the competition. Rudolf is a participant with index 1. It is known that  $m$  problems will be proposed. And the competition will last  $h$  minutes.

A powerful artificial intelligence has predicted the values  $t_{i,j}$ , which represent the number of minutes it will take for the  $i$ -th participant to solve the  $j$ -th problem.

Rudolf realized that the order of solving problems will affect the final result. For example, if  $h = 120$ , and the times to solve problems are [20, 15, 110], then if Rudolf solves the problems in the order:

3, 1, 2, then he will only solve the third problem and get 1 point and 110 penalty.

1, 2, 3, then he will solve the first problem after 20 minutes from the start, the second one after  $20 + 15 = 35$  minutes, and he will not have time to solve the third one. Thus, he will get 2 points and  $20 + 35 = 55$  penalty.

2, 1, 3, then he will solve the second problem after 15 minutes from the start, the first one after  $15 + 20 = 35$  minutes, and he will not have time to solve the third one. Thus, he will get 2 points and  $15 + 35 = 50$  penalty.

Rudolf became interested in what place he will take in the competition if each participant solves problems in the optimal order based on the predictions of the artificial intelligence. It will be assumed that in case of a tie in points and penalty, Rudolf will take the best place.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m, h;
6     cin >> n >> m >> h;
7     vector<vector<int>> t(n, vector<int>(m));
8     vector<pair<int, i64>> a(n);
9     for (int i = 0; i < n; i++) {
10         for (int j = 0; j < m; j++) {
11             cin >> t[i][j];
12         }
13         sort(t[i].begin(), t[i].end());
14         int c = 0;
15         i64 p = 0;
16         int sum = 0;
17         for (auto x : t[i]) {
18             if (sum + x <= h) {
19                 sum += x;
20                 p += sum;
21                 c++;
22             }
23         }
24         a[i] = {c, -p};
25     }
26     int ans = 1;
27     for (int i = 1; i < n; i++) {
28         ans += (a[0] < a[i]);
29     }
30     cout << ans << "\n";
31 }
32 int main() {
33     ios::sync_with_stdio(false);
34     cin.tie(nullptr);
35     int t;
36     cin >> t;
37     while (t--) {
38         solve();
39     }
40     return 0;

```

41 }

**145: Lamps**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have  $n$  lamps, numbered by integers from 1 to  $n$ . Each lamp  $i$  has two integer parameters  $a_i$  and  $b_i$ .

At each moment each lamp is in one of three states: it may be turned on, turned off, or broken.

Initially all lamps are turned off. In one operation you can select one lamp that is turned off and turn it on (you can't turn on broken lamps). You receive  $b_i$  points for turning lamp  $i$  on. The following happens after each performed operation:

Let's denote the number of lamps that are turned on as  $x$  (broken lamps do not count). All lamps  $i$  such that  $a_i \leq x$  simultaneously break, whether they were turned on or off.

Please note that broken lamps never count as turned on and that after a turned on lamp breaks, you still keep points received for turning it on.

You can perform an arbitrary number of operations.

Find the maximum number of points you can get.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     i64 ans = 0;
8     vector<pair<int, int>> l(n);
9     for (int i = 0; i < n; i++) {
10         int a, b;
11         cin >> a >> b;
12         l[i] = {a, -b};
13     }
14     sort(l.begin(), l.end());
15     int cnt = 0;
16     for (int i = 0, j = 0; i < n; i++) {
17         if (i >= j) {
18             cnt++;
19             ans += -l[i].second;
20         }

```

```

21         while (j < n && l[j].first <= cnt) {
22             j++;
23         }
24         cnt = max(0, i + 1 - j);
25     }
26     cout << ans << "\n";
27 }
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(nullptr);
31     int t;
32     cin >> t;
33     while (t--) {
34         solve();
35     }
36     return 0;
37 }
```

**146: Restore the Weather**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  containing the weather forecast for Berlandia for the last  $n$  days. That is,  $a_i$  - is the estimated air temperature on day  $i$  ( $1 \leq i \leq n$ ).

You are also given an array  $b$  - the air temperature that was actually present on each of the days. However, all the values in array  $b$  are mixed up.

Determine which day was which temperature, if you know that the weather never differs from the forecast by more than  $k$  degrees. In other words, if on day  $i$  the real air temperature was  $c$ , then the equality  $|a_i - c| \leq k$  is always true.

For example, let an array  $a = [1, 3, 5, 3, 9]$  of length  $n = 5$  and  $k = 2$  be given and an array  $b = [2, 5, 11, 2, 4]$ . Then, so that the value of  $b_i$  corresponds to the air temperature on day  $i$ , we can rearrange the elements of the array  $b$  so:  $[2, 2, 5, 4, 11]$ . Indeed:

On the 1st day,  $|a_1 - b_1| = |1 - 2| = 1, 1 \leq 2 = k$  is satisfied;

On the 2nd day  $|a_2 - b_2| = |3 - 2| = 1, 1 \leq 2 = k$  is satisfied;

On the 3rd day,  $|a_3 - b_3| = |5 - 5| = 0, 0 \leq 2 = k$  is satisfied;

On the 4th day,  $|a_4 - b_4| = |3 - 4| = 1, 1 \leq 2 = k$  is satisfied;

On the 5th day,  $|a_5 - b_5| = |9 - 11| = 2, 2 \leq 2 = k$  is satisfied.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> a(n), b(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    for (int i = 0; i < n; i++) {
12        cin >> b[i];
13    }
14    vector<int> o(n), ans(n);
15    iota(o.begin(), o.end(), 0);
16    sort(o.begin(), o.end(), [&](int i, int j) {
17        return a[i] < a[j];
18    });
19    sort(b.begin(), b.end());
20    for (int i = 0; i < n; i++) {
21        ans[o[i]] = b[i];
22    }
23    for (int i = 0; i < n; i++) {
24        cout << ans[i] << " \n"[i == n - 1];
25    }
26 }
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(nullptr);
30     int t;
31     cin >> t;
32     while (t--) {
33         solve();
34     }
35     return 0;
36 }
```

### 147: Counting Orders

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given two arrays  $a$  and  $b$  each consisting of  $n$  integers. All elements of  $a$  are pairwise distinct.

Find the number of ways to reorder  $a$  such that  $a_i > b_i$  for all  $1 \leq i \leq n$ , modulo  $10^9 + 7$ .

Two ways of reordering are considered different if the resulting arrays are different.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 void solve() {
35     int n;
36     cin >> n;
37     vector<int> a(n), b(n);
38     for (int i = 0; i < n; i++) {
39         cin >> a[i];
40     }
41     for (int i = 0; i < n; i++) {
42         cin >> b[i];
43     }
44     sort(a.begin(), a.end());
45     sort(b.begin(), b.end());
46     Z ans = 1;
47     for (int i = 0, j = 0; i < n; i++) {
48         while (j < n && a[i] > b[j]) {
49             j++;
50         }
51         ans *= j - i;
52     }
53     cout << ans << "\n";
54 }
55 int main() {
56     ios::sync_with_stdio(false);
57     cin.tie(nullptr);
58     int t;
59     cin >> t;
60     while (t--) {
61         solve();
62     }
63     return 0;
64 }
```

---

## 148: Helpful Maths

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Xenia the beginner mathematician is a third year student at elementary school. She is now learning the addition operation.

The teacher has written down the sum of multiple numbers. Pupils should calculate the sum. To make the calculation easier, the sum only contains numbers 1, 2 and 3. Still, that isn't enough for Xenia. She is only beginning to count, so she can calculate a sum only if the summands follow in non-decreasing order. For example, she can't calculate sum  $1+3+2+1$  but she can calculate sums  $1+1+2$  and  $3+3$ .

You've got the sum that was written on the board. Rearrange the summands and print the sum in such a way that Xenia can calculate the sum.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 a = list(map(int, input().split('+'))) 
2 a.sort()
3 print('+'.join(map(str, a)))
```

---

## 149: Sort with Step

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Let's define a permutation of length  $n$  as an array  $p$  of length  $n$ , which contains every number from 1 to  $n$  exactly once.

You are given a permutation  $p_1, p_2, \dots, p_n$  and a number  $k$ . You need to sort this permutation in the ascending order. In order to do it, you can repeat the following operation any number of times (possibly, zero):

pick two elements of the permutation  $p_i$  and  $p_j$  such that  $|i - j| = k$ , and swap them.

Unfortunately, some permutations can't be sorted with some fixed numbers  $k$ . For example, it's impossible to sort  $[2, 4, 3, 1]$  with  $k = 2$ .

That's why, before starting the sorting, you can make at most one preliminary exchange:

choose any pair  $p_i$  and  $p_j$  and swap them.

Your task is to:

check whether is it possible to sort the permutation without any preliminary exchanges,

if it's not, check, whether is it possible to sort the permutation using exactly one preliminary exchange.

For example, if  $k = 2$  and permutation is  $[2, 4, 3, 1]$ , then you can make a preliminary exchange of  $p_1$  and  $p_4$ , which will produce permutation  $[1, 4, 3, 2]$ , which is possible to sort with given  $k$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     int cnt = 0;
8     for (int i = 0; i < n; i++) {
9         int x;
10        cin >> x;
11        x--;
12        cnt += (x % k != i % k);
13    }
14    if (cnt > 2) {
15        cout << -1 << "\n";
16        return;
17    }
18    cout << cnt / 2 << "\n";
19 }
20 int main() {
21     ios::sync_with_stdio(false);
22     cin.tie(nullptr);
23     int t;
24     cin >> t;
25     while (t--) {
26         solve();
27     }
28     return 0;
29 }
```

## 150: Karina and Array

- Time limit: 2 seconds

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Karina has an array of  $n$  integers  $a_1, a_2, a_3, \dots, a_n$ . She loves multiplying numbers, so she decided that the beauty of a pair of numbers is their product. And the beauty of an array is the maximum beauty of a pair of adjacent elements in the array.

For example, for  $n = 4$ ,  $a = [3, 5, 7, 4]$ , the beauty of the array is  $\max(3 \cdot 5, 5 \cdot 7, 7 \cdot 4) = \max(15, 35, 28) = 35$ .

Karina wants her array to be as beautiful as possible. In order to achieve her goal, she can remove some elements (possibly zero) from the array. After Karina removes all elements she wants to, the array must contain at least two elements.

Unfortunately, Karina doesn't have enough time to do all her tasks, so she asks you to calculate the maximum beauty of the array that she can get by removing any number of elements (possibly zero).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    sort(a.begin(), a.end());
12    if (n > 4) {
13        a.erase(a.begin() + 2, a.end() - 2);
14        n = 4;
15    }
16    i64 ans = -1E18;
17    for (int i = 0; i < n; i++) {
18        for (int j = i+1; j < n; j++) {
19            ans = max(ans, 1LL * a[i] * a[j]);
20        }
21    }
22    cout << ans << "\n";
23 }
24 int main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
27     int t;
28     cin >> t;
29     while (t--) {
30         solve();
31     }
32     return 0;

```

```
33 }
```

## green

### brute force

#### 151: Good Triples

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Given a non-negative integer number  $n$  ( $n \geq 0$ ). Let's say a triple of non-negative integers  $(a, b, c)$  is good if  $a + b + c = n$ , and  $\text{digsum}(a) + \text{digsum}(b) + \text{digsum}(c) = \text{digsum}(n)$ , where  $\text{digsum}(x)$  is the sum of digits of number  $x$ .

For example, if  $n = 26$ , then the pair  $(4, 12, 10)$  is good, because  $4 + 12 + 10 = 26$ , and  $(4) + (1 + 2) + (1 + 0) = (2 + 6)$ .

Your task is to find the number of good triples for the given number  $n$ . The order of the numbers in a triple matters. For example, the triples  $(4, 12, 10)$  and  $(10, 12, 4)$  are two different triples.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     i64 ans = 1;
6     int n;
7     cin >> n;
8     while (n > 0) {
9         int x = n % 10;
10        ans *= (x + 1) * (x + 2) / 2;
11        n /= 10;
12    }
13    cout << ans << "\n";
14 }
15 int main() {
16     ios::sync_with_stdio(false);
17     cin.tie(nullptr);
18     int t;
19     cin >> t;
20     while (t--) {
21         solve();

```

```

22     }
23     return 0;
24 }
```

## 152: Chtholly's request

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output
- I experienced so many great things.
- You gave me memories like dreams... But I have to leave now...
- One last request, can you...
- Help me solve a Codeforces problem?
- .....
- What?

Chtholly has been thinking about a problem for days:

If a number is palindrome and length of its decimal representation without leading zeros is even, we call it a zcy number. A number is palindrome means when written in decimal representation, it contains no leading zeros and reads the same forwards and backwards. For example 12321 and 1221 are palindromes and 123 and 12451 are not. Moreover, 1221 is zcy number and 12321 is not.

Given integers k and p, calculate the sum of the k smallest zcy numbers and output this sum modulo p.

Unfortunately, Willem isn't good at solving this kind of problems, so he asks you for help!

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
```

```

11     }
12     return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<0>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int k, p;
38     cin >> k >> p;
39     Z::setMod(p);
40     Z ans;
41     for (int i = 1; i <= k; i++) {
42         string s = to_string(i);
43         s += string(s.rbegin(), s.rend());
44         ans += stoll(s);
45     }
46     cout << ans << "\n";
47     return 0;
48 }

```

### 153: Binary String Copying

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a string  $s$  consisting of  $n$  characters 0 and/or 1.

You make  $m$  copies of this string, let the  $i$ -th copy be the string  $t_i$ . Then you perform exactly one operation on each of the copies: for the  $i$ -th copy, you sort its substring  $[l_i; r_i]$  (the substring from the  $l_i$ -th character to the  $r_i$ -th character, both endpoints inclusive). Note that each operation affects only one copy, and each copy is affected by only one operation.

Your task is to calculate the number of different strings among  $t_1, t_2, \dots, t_m$ . Note that the initial string  $s$  should be counted only if at least one of the copies stays the same after the operation.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     string s;
8     cin >> s;
9     set<pair<int, int>> S;
10    vector<int> nxt(n + 1, n), lst(n + 1, -1);
11    for (int i = n - 1; i >= 0; i--) {
12        nxt[i] = s[i] == '1' ? i : nxt[i + 1];
13    }
14    for (int i = 0; i < n; i++) {
15        lst[i + 1] = s[i] == '0' ? i : lst[i];
16    }
17    while (m--) {
18        int l, r;
19        cin >> l >> r;
20        l--;
21        l = nxt[l];
22        r = lst[r];
23        if (l > r) {
24            l = r = -1;
25        }
26        S.emplace(l, r);
27    }
28    cout << S.size() << "\n";
29 }
30 int main() {
31     ios::sync_with_stdio(false);
32     cin.tie(nullptr);
33     int t;
34     cin >> t;
35     while (t--) {
36         solve();
37     }
38     return 0;
39 }
```

### 154: Vampiric Powers, anyone?

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

DIO knows that the Stardust Crusaders have determined his location and will be coming to fight him. To foil their plans he decides to send out some Stand users to fight them. Initially, he summoned  $n$

Stand users with him, the  $i$ -th one having a strength of  $a_i$ . Using his vampiric powers, he can do the following as many times as he wishes:

Let the current number of Stand users be  $m$ .

DIO chooses an index  $i$  ( $1 \leq i \leq m$ ).

Then he summons a new Stand user, with index  $m + 1$  and strength given by:

$$a_{m+1} = a_i \oplus a_{i+1} \oplus \dots \oplus a_m,$$

where the operator  $\oplus$  denotes the bitwise XOR operation.

where the operator  $\oplus$  denotes the bitwise XOR operation.

Now, the number of Stand users becomes  $m + 1$ .

Unfortunately for DIO, by using Hermit Purple's divination powers, the Crusaders know that he is plotting this, and they also know the strengths of the original Stand users. Help the Crusaders find the maximum possible strength of a Stand user among all possible ways that DIO can summon.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<int> s(n + 1);
12    int ans = 0;
13    for (int i = 0; i < n; i++) {
14        s[i + 1] = s[i] ^ a[i];
15    }
16    sort(s.begin(), s.end());
17    s.erase(unique(s.begin(), s.end()), s.end());
18    for (int i = 0; i < s.size(); i++) {
19        for (int j = 0; j < i; j++) {
20            ans = max(ans, s[i] ^ s[j]);
21        }
22    }
23    cout << ans << "\n";
24 }
25 int main() {
26     ios::sync_with_stdio(false);
27     cin.tie(nullptr);
28     int t;
29     cin >> t;
30     while (t--) {

```

```

31     solve();
32 }
33 return 0;
34 }
```

## 155: Tracking Segments

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  consisting of  $n$  zeros. You are also given a set of  $m$  not necessarily different segments. Each segment is defined by two numbers  $l_i$  and  $r_i$  ( $1 \leq l_i \leq r_i \leq n$ ) and represents a subarray  $a_{l_i}, a_{l_i+1}, \dots, a_{r_i}$  of the array  $a$ .

Let's call the segment  $l_i, r_i$  beautiful if the number of ones on this segment is strictly greater than the number of zeros. For example, if  $a = [1, 0, 1, 0, 1]$ , then the segment  $[1, 5]$  is beautiful (the number of ones is 3, the number of zeros is 2), but the segment  $[3, 4]$  is not beautiful (the number of ones is 1, the number of zeros is 1).

You also have  $q$  changes. For each change you are given the number  $1 \leq x \leq n$ , which means that you must assign an element  $a_x$  the value 1.

You have to find the first change after which at least one of  $m$  given segments becomes beautiful, or report that none of them is beautiful after processing all  $q$  changes.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<int> l(m), r(m);
8     for (int i = 0; i < m; i++) {
9         cin >> l[i] >> r[i];
10        l[i]--;
11    }
12    int q;
13    cin >> q;
14    vector<int> x(q);
15    for (int i = 0; i < q; i++) {
16        cin >> x[i];
17        x[i]--;
18    }
19    int lo = 0, hi = q + 1;
20    while (lo < hi) {
```

```

21     int t = (lo + hi) / 2;
22     vector<int> s(n + 1);
23     for (int i = 0; i < t; i++) {
24         s[x[i] + 1] = 1;
25     }
26     for (int i = 1; i <= n; i++) {
27         s[i] += s[i - 1];
28     }
29     bool ok = false;
30     for (int i = 0; i < m; i++) {
31         if (s[r[i]] - s[l[i]] > (r[i] - l[i]) / 2) {
32             ok = true;
33         }
34     }
35     if (ok) {
36         hi = t;
37     } else {
38         lo = t + 1;
39     }
40 }
41 int ans = lo;
42 if (ans > q) {
43     ans = -1;
44 }
45 cout << ans << "\n";
46 }
47 int main() {
48     ios::sync_with_stdio(false);
49     cin.tie(nullptr);
50     int t;
51     cin >> t;
52     while (t--) {
53         solve();
54     }
55     return 0;
56 }
```

## 156: Ice Sculptures

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The Berland University is preparing to celebrate the 256-th anniversary of its founding! A specially appointed Vice Rector for the celebration prepares to decorate the campus. In the center of the campus  $n$  ice sculptures were erected. The sculptures are arranged in a circle at equal distances from each other, so they form a regular  $n$ -gon. They are numbered in clockwise order with numbers from 1 to  $n$ .

The site of the University has already conducted a voting that estimated each sculpture's characteristic of  $t_i$  - the degree of the sculpture's attractiveness. The values of  $t_i$  can be positive, negative or zero.

When the university rector came to evaluate the work, he said that this might be not the perfect arrangement. He suggested to melt some of the sculptures so that:

the remaining sculptures form a regular polygon (the number of vertices should be between 3 and  $n$ ),

the sum of the  $t_i$  values of the remaining sculptures is maximized.

Help the Vice Rector to analyze the criticism - find the maximum value of  $t_i$  sum which can be obtained in this way. It is allowed not to melt any sculptures at all. The sculptures can not be moved.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> t(n);
10    for (int i = 0; i < n; i++) {
11        cin >> t[i];
12    }
13    i64 ans = -1E18;
14    for (int x = 3; x <= n; x++) {
15        if (n % x) {
16            continue;
17        }
18        for (int i = 0; i < n/x; i++) {
19            i64 s = 0;
20            for (int j = i; j < n; j += n/x) {
21                s += t[j];
22            }
23            ans = max(ans, s);
24        }
25    }
26    cout << ans << "\n";
27    return 0;
28 }
```

## 157: Martian Dollar

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

One day Vasya got hold of information on the Martian dollar course in bourles for the next  $n$  days. The buying prices and the selling prices for one dollar on day  $i$  are the same and are equal to  $a_i$ . Vasya has  $b$  bourles. He can buy a certain number of dollars and then sell it no more than once in  $n$  days. According to Martian laws, one can buy only an integer number of dollars. Which maximal sum of money in bourles can Vasya get by the end of day  $n$ ?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, b;
8     cin >> n >> b;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    int ans = b;
14    for (int i = 0; i < n; i++) {
15        for (int j = i+1; j < n; j++) {
16            ans = max(ans, b + (a[j] - a[i]) * (b / a[i]));
17        }
18    }
19    cout << ans << "\n";
20    return 0;
21 }
```

## 158: The Text Splitting

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given the string  $s$  of length  $n$  and the numbers  $p, q$ . Split the string  $s$  to pieces of length  $p$  and  $q$ .

For example, the string “Hello” for  $p = 2, q = 3$  can be split to the two strings “Hel” and “lo” or to the two strings “He” and “llo”.

Note it is allowed to split the string  $s$  to the strings only of length  $p$  or to the strings only of length  $q$  (see the second sample test).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
```

```

7     int n, p, q;
8     cin >> n >> p >> q;
9     string s;
10    cin >> s;
11    for (int i = 0; i*p <= n; i++) {
12        if ((n - i*p) % q == 0) {
13            cout << i + (n - i*p) / q << "\n";
14            int j = 0;
15            while (j < i*p) {
16                cout << s.substr(j, p) << "\n";
17                j += p;
18            }
19            while (j < n) {
20                cout << s.substr(j, q) << "\n";
21                j += q;
22            }
23            return 0;
24        }
25    }
26    cout << -1 << "\n";
27    return 0;
28 }
```

### 159: Tear It Apart

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a string  $s$ , consisting of lowercase Latin letters.

In one operation, you can select several (one or more) positions in it such that no two selected positions are adjacent to each other. Then you remove the letters on the selected positions from the string. The resulting parts are concatenated without changing their order.

What is the smallest number of operations required to make all the letters in  $s$  the same?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     string s;
6     cin >> s;
7     int n = s.size();
8     vector<int> p[26];
9     for (int i = 0; i < n; i++) {
10         p[s[i] - 'a'].push_back(i);
11     }
```

```

12     int ans = n;
13     for (int x = 0; x < 26; x++) {
14         int t = -1;
15         p[x].push_back(n);
16         int res = 0;
17         for (auto i : p[x]) {
18             int len = i - t - 1;
19             if (len) {
20                 res = max(res, __lg(len) + 1);
21             }
22             t = i;
23         }
24         ans = min(ans, res);
25     }
26     cout << ans << "\n";
27 }
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(nullptr);
31     int t;
32     cin >> t;
33     while (t--) {
34         solve();
35     }
36     return 0;
37 }
```

## 160: Fire Again

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: input.txt
- Output file: output.txt

After a terrifying forest fire in Berland a forest rebirth program was carried out. Due to it N rows with M trees each were planted and the rows were so neat that one could map it on a system of coordinates so that the j-th tree in the i-th row would have the coordinates of (i, j). However a terrible thing happened and the young forest caught fire. Now we must find the coordinates of the tree that will catch fire last to plan evacuation.

The burning began in K points simultaneously, which means that initially K trees started to burn. Every minute the fire gets from the burning trees to the ones that aren't burning and that the distance from them to the nearest burning tree equals to 1.

Find the tree that will be the last to start burning. If there are several such trees, output any.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
```

```

3  using i64 = long long;
4  #ifdef ONLINE_JUDGE
5      ifstream fin("input.txt");
6      ofstream fout("output.txt");
7  #else
8      #define fin cin
9      #define fout cout
10 #endif
11 int main() {
12     int n, m, k;
13     fin >> n >> m >> k;
14     vector<int> x(k), y(k);
15     for (int i = 0; i < k; i++) {
16         fin >> x[i] >> y[i];
17     }
18     int X = 1, Y = 1, ans = 0;
19     for (int i = 1; i <= n; i++) {
20         for (int j = 1; j <= m; j++) {
21             int d = 1E9;
22             for (int l = 0; l < k; l++) {
23                 d = min(d, abs(i - x[l]) + abs(j - y[l]));
24             }
25             if (d > ans) {
26                 X = i;
27                 Y = j;
28                 ans = d;
29             }
30         }
31     }
32     fout << X << " " << Y << "\n";
33     return 0;
34 }
```

## 161: Constructive Problem

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

As you know, any problem that does not require the use of complex data structures is considered constructive. You are offered to solve one of such problems.

You are given an array  $a$  of  $n$  non-negative integers. You are allowed to perform the following operation exactly once: choose some non-empty subsegment  $a_l, a_{l+1}, \dots, a_r$  of the array  $a$  and a non-negative integer  $k$ , and assign value  $k$  to all elements of the array on the chosen subsegment.

The task is to find out whether  $\text{MEX}(a)$  can be increased by exactly one by performing such an operation. In other words, if before the operation  $\text{MEX}(a) = m$  held, then after the operation it must hold that  $\text{MEX}(a) = m + 1$ .

Recall that  $\text{MEX}$  of a set of integers  $c_1, c_2, \dots, c_k$  is defined as the smallest non-negative integer  $x$  which does not occur in the set  $c$ .

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     vector<int> f(n + 2);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11         if (a[i] <= n + 1) {
12             f[a[i]] = 1;
13         }
14     }
15     int m = 0;
16     while (f[m]) {
17         m++;
18     }
19     if (!f[m + 1]) {
20         if (m < n) {
21             cout << "Yes\n";
22         } else {
23             cout << "No\n";
24         }
25     }
26     return;
27 }
28 int l = 0, r = n - 1;
29 while (a[l] != m + 1) {
30     l++;
31 }
32 while (a[r] != m + 1) {
33     r--;
34 }
35 f.assign(n + 2, 0);
36 for (int i = l; i <= r; i++) {
37     a[i] = m;
38 }
39 for (auto x : a) {
40     if (x <= n + 1) {
41         f[x] = 1;
42     }
43 }
44 int newm = 0;
45 while (f[newm]) {
46     newm++;
47 }
48 if (newm == m + 1) {
49     cout << "Yes\n";
50 } else {
51     cout << "No\n";
52 }
53 int main() {
54     ios::sync_with_stdio(false);
55     cin.tie(nullptr);
56     int t;
57     cin >> t;
58     while (t--) {
59         solve();
60     }
61 }
```

```

61     return 0;
62 }
```

## 162: Accounting

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A long time ago in some far country lived king Copa. After the recent king's reform, he got so large powers that started to keep the books by himself.

The total income A of his kingdom during 0-th year is known, as well as the total income B during n-th year (these numbers can be negative - it means that there was a loss in the correspondent year).

King wants to show financial stability. To do this, he needs to find common coefficient X - the coefficient of income growth during one year. This coefficient should satisfy the equation:

Surely, the king is not going to do this job by himself, and demands you to find such number X.

It is necessary to point out that the fractional numbers are not used in kingdom's economy. That's why all input numbers as well as coefficient X must be integers. The number X may be zero or negative.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int A, B, n;
8     cin >> A >> B >> n;
9     if (B == 0) {
10         cout << 0 << "\n";
11         return 0;
12     }
13     if (A == 0) {
14         cout << "No solution\n";
15         return 0;
16     }
17     int v = B / A, x;
18     if (v < 0) {
19         v = -v;
20         if (n % 2 == 0) {
21             cout << "No solution\n";
22         }
23     }
```

```

24         x = pow(v, 1. / n) + .5;
25         x = -x;
26     } else {
27         x = pow(v, 1. / n) + .5;
28     }
29     for (int i = 0; i < n; i++) {
30         A *= x;
31     }
32     if (A != B) {
33         cout << "No solution\n";
34     } else {
35         cout << x << "\n";
36     }
37     return 0;
38 }
```

**163: IQ test**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Bob is preparing to pass IQ test. The most frequent task in this test is to find out which one of the given  $n$  numbers differs from the others. Bob observed that one number usually differs from the others in evenness. Help Bob - to check his answers, he needs a program that among the given  $n$  numbers finds one that is different in evenness.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    int cnt[2] {};
11    for (int i = 0; i < n; i++) {
12        cin >> a[i];
13        cnt[a[i] % 2] += 1;
14    }
15    for (int i = 0; i < n; i++) {
16        if (cnt[a[i] % 2] == 1) {
17            cout << i + 1 << "\n";
18        }
19    }
20    return 0;
21 }
```

## 164: Bargaining Table

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Bob wants to put a new bargaining table in his office. To do so he measured the office room thoroughly and drew its plan: Bob's office room is a rectangular room  $n \times m$  meters. Each square meter of the room is either occupied by some furniture, or free. A bargaining table is rectangular, and should be placed so, that its sides are parallel to the office walls. Bob doesn't want to change or rearrange anything, that's why all the squares that will be occupied by the table should be initially free. Bob wants the new table to sit as many people as possible, thus its perimeter should be maximal. Help Bob find out the maximum possible perimeter of a bargaining table for his office.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<string> s(n);
10    for (int i = 0; i < n; i++) {
11        cin >> s[i];
12    }
13    int ans = 0;
14    for (int x1 = 0; x1 < n; x1++) {
15        for (int x2 = x1; x2 < n; x2++) {
16            int len = 0;
17            for (int i = 0; i < m; i++) {
18                bool ok = true;
19                for (int j = x1; j <= x2; j++) {
20                    if (s[j][i] == '1') {
21                        ok = false;
22                    }
23                }
24                if (ok) {
25                    len += 1;
26                    ans = max(ans, (len + x2 - x1 + 1) * 2);
27                } else {
28                    len = 0;
29                }
30            }
31        }
32    }
33    cout << ans << "\n";
34    return 0;

```

```
35 }
```

## 165: Triangle

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

At a geometry lesson Bob learnt that a triangle is called right-angled if it is nondegenerate and one of its angles is right. Bob decided to draw such a triangle immediately: on a sheet of paper he drew three points with integer coordinates, and joined them with segments of straight lines, then he showed the triangle to Peter. Peter said that Bob's triangle is not right-angled, but is almost right-angled: the triangle itself is not right-angled, but it is possible to move one of the points exactly by distance 1 so, that all the coordinates remain integer, and the triangle become right-angled. Bob asks you to help him and find out if Peter tricks him. By the given coordinates of the triangle you should find out if it is right-angled, almost right-angled, or neither of these.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 # struct Point;
6 template<class T>
7 T dot(Point<T> a, Point<T> b) {
8     return a.x * b.x + a.y * b.y;
9 }
10 template<class T>
11 T cross(Point<T> a, Point<T> b) {
12     return a.x * b.y - a.y * b.x;
13 }
14 template<class T>
15 T square(Point<T> p) {
16     return dot(p, p);
17 }
18 template<class T>
19 double length(Point<T> p) {
20     return sqrt(double(square(p)));
21 }
22 long double length(Point<long double> p) {
23     return sqrt(square(p));
24 }
25 template<class T>
26 # struct Line;
27 template<class T>
28 Point<T> rotate(Point<T> a) {
```

```

29     return Point(-a.y, a.x);
30 }
31 template<class T>
32 int sgn(Point<T> a) {
33     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
34 }
35 template<class T>
36 bool pointOnLineLeft(Point<T> p, Line<T> l) {
37     return cross(l.b - l.a, p - l.a) > 0;
38 }
39 template<class T>
40 Point<T> lineIntersection(Line<T> l1, Line<T> l2) {
41     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1
42         .a - l1.b));
43 }
44 template<class T>
45 bool pointOnSegment(Point<T> p, Line<T> l) {
46     return cross(p - l.a, l.b - l.a) == 0 && min(l.a.x, l.b.x) <= p.x && p.x <= max(l.a.x,
47         l.b.x)
48         && min(l.a.y, l.b.y) <= p.y && p.y <= max(l.a.y, l.b.y);
49 }
50 template<class T>
51 bool pointInPolygon(Point<T> a, vector<Point<T>> p) {
52     int n = p.size();
53     for (int i = 0; i < n; i++) {
54         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
55             return true;
56         }
57         int t = 0;
58         for (int i = 0; i < n; i++) {
59             auto u = p[i];
60             auto v = p[(i + 1) % n];
61             if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
62                 t ^= 1;
63             }
64             if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
65                 t ^= 1;
66             }
67         }
68         return t == 1;
69     }
70     // 0 : not intersect
71     // 1 : strictly intersect
72     // 2 : overlap
73     // 3 : intersect at endpoint
74     template<class T>
75     tuple<int, Point<T>, Point<T>> segmentIntersection(Line<T> l1, Line<T> l2) {
76         if (max(l1.a.x, l1.b.x) < min(l2.a.x, l2.b.x)) {
77             return {0, Point<T>(), Point<T>()};
78         }
79         if (min(l1.a.x, l1.b.x) > max(l2.a.x, l2.b.x)) {
80             return {0, Point<T>(), Point<T>()};
81         }
82         if (max(l1.a.y, l1.b.y) < min(l2.a.y, l2.b.y)) {
83             return {0, Point<T>(), Point<T>()};
84         }
85         if (min(l1.a.y, l1.b.y) > max(l2.a.y, l2.b.y)) {
86             return {0, Point<T>(), Point<T>()};
87         }
88         if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
89             if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
90                 return {0, Point<T>(), Point<T>()};
91             } else {

```

```

91         int maxx1 = max(l1.a.x, l1.b.x);
92         int minx1 = min(l1.a.x, l1.b.x);
93         int maxy1 = max(l1.a.y, l1.b.y);
94         int miny1 = min(l1.a.y, l1.b.y);
95         int maxx2 = max(l2.a.x, l2.b.x);
96         int minx2 = min(l2.a.x, l2.b.x);
97         int maxy2 = max(l2.a.y, l2.b.y);
98         int miny2 = min(l2.a.y, l2.b.y);
99         Point<T> p1(max(minx1, minx2), max(miny1, miny2));
100        Point<T> p2(min(maxx1, maxx2), min(maxy1, maxy2));
101        if (!pointOnSegment(p1, l1)) {
102            swap(p1.y, p2.y);
103        }
104        if (p1 == p2) {
105            return {3, p1, p2};
106        } else {
107            return {2, p1, p2};
108        }
109    }
110
111    auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
112    auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
113    auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
114    auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
115    if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) || (cp3 < 0
116        && cp4 < 0)) {
117        return {0, Point<T>(), Point<T>()};
118    }
119    Point p = lineIntersection(l1, l2);
120    if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
121        return {1, p, p};
122    } else {
123        return {3, p, p};
124    }
125    template<class T>
126    bool segmentInPolygon(Line<T> l, vector<Point<T>> p) {
127        int n = p.size();
128        if (!pointInPolygon(l.a, p)) {
129            return false;
130        }
131        if (!pointInPolygon(l.b, p)) {
132            return false;
133        }
134        for (int i = 0; i < n; i++) {
135            auto u = p[i];
136            auto v = p[(i + 1) % n];
137            auto w = p[(i + 2) % n];
138            auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
139            if (t == 1) {
140                return false;
141            }
142            if (t == 0) {
143                continue;
144            }
145            if (t == 2) {
146                if (pointOnSegment(v, l) && v != l.a && v != l.b) {
147                    if (cross(v - u, w - v) > 0) {
148                        return false;
149                    }
150                }
151            } else {
152                if (p1 != u && p1 != v) {
153                    if (pointOnLineLeft(l.a, Line(v, u)))

```

```

154             || pointOnLineLeft(l.b, Line(v, u))) {
155         return false;
156     }
157 } else if (p1 == v) {
158     if (l.a == v) {
159         if (pointOnLineLeft(u, l)) {
160             if (pointOnLineLeft(w, l)
161                 && pointOnLineLeft(w, Line(u, v))) {
162                 return false;
163             }
164         } else {
165             if (pointOnLineLeft(w, l)
166                 || pointOnLineLeft(w, Line(u, v))) {
167                 return false;
168             }
169         }
170     } else if (l.b == v) {
171         if (pointOnLineLeft(u, Line(l.b, l.a))) {
172             if (pointOnLineLeft(w, Line(l.b, l.a))
173                 && pointOnLineLeft(w, Line(u, v))) {
174                 return false;
175             }
176         } else {
177             if (pointOnLineLeft(w, Line(l.b, l.a))
178                 || pointOnLineLeft(w, Line(u, v))) {
179                 return false;
180             }
181         }
182     } else {
183         if (pointOnLineLeft(u, l)) {
184             if (pointOnLineLeft(w, Line(l.b, l.a))
185                 || pointOnLineLeft(w, Line(u, v))) {
186                 return false;
187             }
188         } else {
189             if (pointOnLineLeft(w, l)
190                 || pointOnLineLeft(w, Line(u, v))) {
191                 return false;
192             }
193         }
194     }
195 }
196 }
197 }
198 return true;
199 }
200 template<class T>
201 vector<Point<T>> hp(vector<Line<T>> lines) {
202     sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
203         auto d1 = l1.b - l1.a;
204         auto d2 = l2.b - l2.a;
205         if (sgn(d1) != sgn(d2)) {
206             return sgn(d1) == 1;
207         }
208         return cross(d1, d2) > 0;
209     });
210     deque<Line<T>> ls;
211     deque<Point<T>> ps;
212     for (auto l : lines) {
213         if (ls.empty()) {
214             ls.push_back(l);
215             continue;
216         }
217         while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {

```

```

218         ps.pop_back();
219         ls.pop_back();
220     }
221     while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
222         ps.pop_front();
223         ls.pop_front();
224     }
225     if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
226         if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
227             if (!pointOnLineLeft(ls.back().a, l)) {
228                 assert(ls.size() == 1);
229                 ls[0] = l;
230             }
231             continue;
232         }
233         return {};
234     }
235     ps.push_back(lineIntersection(ls.back(), l));
236     ls.push_back(l);
237 }
238 while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
239     ps.pop_back();
240     ls.pop_back();
241 }
242 if (ls.size() <= 2) {
243     return {};
244 }
245 ps.push_back(lineIntersection(ls[0], ls.back()));
246 return vector(ps.begin(), ps.end());
247 }
248 const int dx[] = {0, 0, -1, 1};
249 const int dy[] = {-1, 1, 0, 0};
250 bool check(auto p) {
251     if (cross(p[1] - p[0], p[2] - p[0]) == 0) {
252         return false;
253     }
254     for (int i = 0; i < 3; i++) {
255         if (dot(p[(i + 1) % 3] - p[i], p[(i + 2) % 3] - p[i]) == 0) {
256             return true;
257         }
258     }
259     return false;
260 }
261 int main() {
262     ios::sync_with_stdio(false);
263     cin.tie(nullptr);
264     array<Point<int>, 3> p;
265     for (int i = 0; i < 3; i++) {
266         cin >> p[i].x >> p[i].y;
267     }
268     if (check(p)) {
269         cout << "RIGHT\n";
270         return 0;
271     }
272     for (int i = 0; i < 3; i++) {
273         for (int j = 0; j < 4; j++) {
274             auto q = p;
275             q[i].x += dx[j];
276             q[i].y += dy[j];
277             if (check(q)) {
278                 cout << "ALMOST\n";
279                 return 0;
280             }
281         }
282     }

```

```

282     }
283     cout << "NEITHER\n";
284     return 0;
285 }
```

## 166: Make It Permutation

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have an integer array  $a$  of length  $n$ . There are two kinds of operations you can make.

Remove an integer from  $a$ . This operation costs  $c$ .

Insert an arbitrary positive integer  $x$  to any position of  $a$  (to the front, to the back, or between any two consecutive elements). This operation costs  $d$ .

You want to make the final array a permutation of any positive length. Please output the minimum cost of doing that. Note that you can make the array empty during the operations, but the final array must contain at least one integer.

A permutation of length  $n$  is an array consisting of  $n$  distinct integers from 1 to  $n$  in arbitrary order. For example,  $[2, 3, 1, 5, 4]$  is a permutation, but  $[1, 2, 2]$  is not a permutation (2 appears twice in the array), and  $[1, 3, 4]$  is also not a permutation ( $n = 3$  but there is 4 in the array).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, c, d;
6     cin >> n >> c >> d;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    sort(a.begin(), a.end());
12    i64 ans = 1LL * c * n + d;
13    int t = 0;
14    for (int i = 0; i < n; i++) {
15        if (i == 0 || a[i] != a[i - 1]) {
16            t += 1;
17        }
18        ans = min(ans, 1LL * c * n + 1LL * d * a[i] - 1LL * (c + d) * t);
19    }
20    cout << ans << "\n";
21 }
```

```

22 int main() {
23     ios::sync_with_stdio(false);
24     cin.tie(nullptr);
25     int t;
26     cin >> t;
27     while (t--) {
28         solve();
29     }
30     return 0;
31 }
```

## 167: Pull Your Luck

- Time limit: 1 second
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

While James is gone on business, Vesper takes her time and explores what the legendary Casino Royale has to offer to people who are fond of competitive programming.

Her attention was grabbed by the very new “Pull Your Luck” roulette which functions in a pretty peculiar way. The roulette’s wheel consists of  $n$  sectors number from 0 to  $n - 1$ . There is no ball and the winning sector is determined by a static arrow pointing to one of the sectors. Sectors’ indexes go in the natural order and the wheel always spins in the direction of indexes increment. That means that sector  $i + 1$  goes right after sector  $i$  for all  $i$  from 0 to  $n - 2$ , and sector 0 goes right after sector  $n - 1$ .

After a bet is made, the player is allowed to pull the triggering handle herself and cause the wheel to spin. If the player’s initial pull is made with the force equal to positive integer  $f$ , the wheel will spin for  $f$  seconds. During the first second it will advance  $f$  sectors, the next second it will advance  $f - 1$  sectors, then  $f - 2$  sectors, and so on until it comes to a complete stop. After the wheel comes to a complete stop, the sector which the arrow is pointing to is the winning one.

The roulette’s arrow currently points at sector  $x$ . Vesper knows that she can pull the handle with any integer force from 1 to  $p$  inclusive. Note that it is not allowed to pull the handle with force 0, i. e. not pull it at all. The biggest prize is awarded if the winning sector is 0. Now Vesper wonders if she can make sector 0 win by pulling the triggering handle exactly once?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, x, p;
6     cin >> n >> x >> p;
```

```

7     for (int i = 1; i <= min(p, 2 * n); i++) {
8         if ((x + 1LL * i * (i + 1) / 2) % n == 0) {
9             cout << "Yes\n";
10            return;
11        }
12    }
13    cout << "No\n";
14 }
15 int main() {
16     ios::sync_with_stdio(false);
17     cin.tie(nullptr);
18     int t;
19     cin >> t;
20     while (t--) {
21         solve();
22     }
23     return 0;
24 }
```

### 168: Unforgivable Curse (hard version)

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is a complex version of the problem. This version has no additional restrictions on the number  $k$ .

The chief wizard of the Wizengamot once caught the evil wizard Drahnyrt, but the evil wizard has returned and wants revenge on the chief wizard. So he stole spell  $s$  from his student Harry.

The spell - is a  $n$ -length string of lowercase Latin letters.

Drahnyrt wants to replace spell with an unforgivable curse - string  $t$ .

Dragirt, using ancient magic, can swap letters at a distance  $k$  or  $k + 1$  in spell as many times as he wants. In other words, Drahnyrt can change letters in positions  $i$  and  $j$  in spell  $s$  if  $|i - j| = k$  or  $|i - j| = k + 1$ .

For example, if  $\$k = 3$ ,  $s = \$\text{"talant"}$  and  $\$t = \$\text{"atltna"}$ , Drahnyrt can act as follows:

swap the letters at positions 1 and 4 to get spell “aaltnt”.

swap the letters at positions 2 and 6 to get spell “atltna”.

You are given spells  $s$  and  $t$ . Can Drahnyrt change spell  $s$  to  $t$ ?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     string s, t;
8     cin >> s >> t;
9     int cnt[26] {};
10    for (int i = 0; i < n; i++) {
11        if (i - k < 0 && i + k >= n) {
12            if (s[i] != t[i]) {
13                cout << "NO\n";
14                return;
15            }
16        } else {
17            cnt[s[i] - 'a'] += 1;
18            cnt[t[i] - 'a'] -= 1;
19        }
20    }
21    for (int i = 0; i < 26; i++) {
22        if (cnt[i] != 0) {
23            cout << "NO\n";
24            return;
25        }
26    }
27    cout << "YES\n";
28 }
29 int main() {
30     ios::sync_with_stdio(false);
31     cin.tie(nullptr);
32     int t;
33     cin >> t;
34     while (t--) {
35         solve();
36     }
37     return 0;
38 }
```

### 169: Unforgivable Curse (easy version)

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an easy version of the problem. In this version,  $k$  is always 3.

The chief wizard of the Wizengamot once caught the evil wizard Drahnyrt, but the evil wizard has returned and wants revenge on the chief wizard. So he stole spell  $s$  from his student Harry.

The spell  $s$  is a  $n$ -length string of lowercase Latin letters.

Drahnyrt wants to replace spell  $s$  with an unforgivable curse  $t$ .

Drahhyrt, using ancient magic, can swap letters at a distance  $k$  or  $k + 1$  in spell as many times as he wants. In this version of the problem, you can swap letters at a distance of 3 or 4. In other words, Drahhyrt can change letters in positions  $i$  and  $j$  in spell  $s$  if  $|i - j| = 3$  or  $|i - j| = 4$ .

For example, if  $\$s = \$$  “talant” and  $\$t = \$$  “atltnt”, Drahhyrt can act as follows:

swap the letters at positions 1 and 4 to get spell “aaltnnt”.

swap the letters at positions 2 and 6 to get spell “atltnt”.

You are given spells  $s$  and  $t$ . Can Drahhyrt change spell  $s$  to  $t$ ?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     string s, t;
8     cin >> s >> t;
9     int cnt[26] {};
10    for (int i = 0; i < n; i++) {
11        if (i - k < 0 && i + k >= n) {
12            if (s[i] != t[i]) {
13                cout << "NO\n";
14                return;
15            }
16        } else {
17            cnt[s[i] - 'a'] += 1;
18            cnt[t[i] - 'a'] -= 1;
19        }
20    }
21    for (int i = 0; i < 26; i++) {
22        if (cnt[i] != 0) {
23            cout << "NO\n";
24            return;
25        }
26    }
27    cout << "YES\n";
28 }
29 int main() {
30     ios::sync_with_stdio(false);
31     cin.tie(nullptr);
32     int t;
33     cin >> t;
34     while (t--) {
35         solve();
36     }
37     return 0;
38 }
```

## 170: Same Count One

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

ChthollyNotaSeniorious received a special gift from AquaMoon:  $n$  binary arrays of length  $m$ . AquaMoon tells him that in one operation, he can choose any two arrays and any position  $pos$  from 1 to  $m$ , and swap the elements at positions  $pos$  in these arrays.

He is fascinated with this game, and he wants to find the minimum number of operations needed to make the numbers of 1s in all arrays the same. He has invited you to participate in this interesting game, so please try to find it!

If it is possible, please output specific exchange steps in the format described in the output section. Otherwise, please output  $-1$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     int tot = 0;
8     vector<vector<int>> a(n, vector<int>(m));
9     vector<int> sum(n);
10    for (int i = 0; i < n; i++) {
11        for (int j = 0; j < m; j++) {
12            cin >> a[i][j];
13            tot += a[i][j];
14            sum[i] += a[i][j];
15        }
16    }
17    if (tot % n != 0) {
18        cout << -1 << "\n";
19        return;
20    }
21    vector<tuple<int, int, int>> ans;
22    for (int j = 0; j < m; j++) {
23        vector<int> zero, one;
24        for (int i = 0; i < n; i++) {
25            if (a[i][j] == 0 && sum[i] < tot / n) {
26                zero.push_back(i);
27            }
28            if (a[i][j] == 1 && sum[i] > tot / n) {
29                one.push_back(i);
30            }
31        }
32        for (int k = 0; k < min(zero.size(), one.size()); k++) {
33            int i0 = zero[k];
34            int i1 = one[k];

```

```

35         a[i0][j] = 1;
36         a[i1][j] = 0;
37         sum[i0]++;
38         sum[i1]--;
39         ans.emplace_back(i0, i1, j);
40     }
41 }
42 cout << ans.size() << "\n";
43 for (auto [x, y, z] : ans) {
44     cout << x + 1 << " " << y + 1 << " " << z + 1 << "\n";
45 }
46 }
47 int main() {
48     ios::sync_with_stdio(false);
49     cin.tie(nullptr);
50     int t;
51     cin >> t;
52     while (t--) {
53         solve();
54     }
55     return 0;
56 }
```

## constructive algorithms

### 171: Jumping Through Segments

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Polycarp is designing a level for a game. The level consists of  $n$  segments on the number line, where the  $i$ -th segment starts at the point with coordinate  $l_i$  and ends at the point with coordinate  $r_i$ .

The player starts the level at the point with coordinate 0. In one move, they can move to any point that is within a distance of no more than  $k$ . After their  $i$ -th move, the player must land within the  $i$ -th segment, that is, at a coordinate  $x$  such that  $l_i \leq x \leq r_i$ . This means:

After the first move, they must be inside the first segment (from  $l_1$  to  $r_1$ );

After the second move, they must be inside the second segment (from  $l_2$  to  $r_2$ );

...

After the  $n$ -th move, they must be inside the  $n$ -th segment (from  $l_n$  to  $r_n$ ).

The level is considered completed if the player reaches the  $n$ -th segment, following the rules described above. After some thought, Polycarp realized that it is impossible to complete the level with some values of  $k$ .

Polycarp does not want the level to be too easy, so he asks you to determine the minimum integer  $k$  with which it is possible to complete the level.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> l(n), r(n);
8     for (int i = 0; i < n; i++) {
9         cin >> l[i] >> r[i];
10    }
11    int ans = *ranges::partition_point(ranges::iota_view(0, int(1E9) + 1),
12                                         [&](int k) {
13                                             int L = 0, R = 0;
14                                             for (int i = 0; i < n; i++) {
15                                                 L -= k;
16                                                 R += k;
17                                                 L = max(L, l[i]);
18                                                 R = min(R, r[i]);
19                                                 if (L > R) {
20                                                     return true;
21                                                 }
22                                             }
23                                         return false;
24                                     });
25     cout << ans << "\n";
26 }
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(nullptr);
30     int t;
31     cin >> t;
32     while (t--) {
33         solve();
34     }
35     return 0;
36 }
```

## 172: ABBC or BACB

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a string  $s$  made up of characters A and B. Initially you have no coins. You can perform two types of operations:

Pick a substring<sup>†</sup> AB, change it to BC, and get a coin.

Pick a substring<sup>†</sup> BA, change it to CB, and get a coin.

<sup>†</sup> A substring of length 2 is a sequence of two adjacent characters of a string.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     string s;
6     cin >> s;
7     int ans = 0;
8     int n = s.size();
9     for (int i = 0, j = -1; i <= n; i++) {
10         if (i == n || s[i] == 'C') {
11             int min = n;
12             for (int x = j + 1, y = j; x <= i; x++) {
13                 if (x < i && s[x] == 'A') {
14                     ans++;
15                 } else {
16                     min = min(min, x - y - 1);
17                     y = x;
18                 }
19             }
20             ans -= min;
21         }
22     }
23     cout << ans << "\n";
24 }
25 int main() {
26     ios::sync_with_stdio(false);
27     cin.tie(nullptr);
28     int t;
29     cin >> t;
30     while (t--) {
31         solve();
32     }
33     return 0;
34 }
```

### 173: Salyg1n and the MEX Game

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an interactive problem!

salyg1n gave Alice a set  $S$  of  $n$  distinct integers  $s_1, s_2, \dots, s_n$  ( $0 \leq s_i \leq 10^9$ ). Alice decided to play a game with this set against Bob. The rules of the game are as follows:

Players take turns, with Alice going first.

In one move, Alice adds one number  $x$  ( $0 \leq x \leq 10^9$ ) to the set  $S$ . The set  $S$  must not contain the number  $x$  at the time of the move.

In one move, Bob removes one number  $y$  from the set  $S$ . The set  $S$  must contain the number  $y$  at the time of the move. Additionally, the number  $y$  must be strictly smaller than the last number added by Alice.

The game ends when Bob cannot make a move or after  $2 \cdot n + 1$  moves (in which case Alice's move will be the last one).

The result of the game is  $\text{MEX } \dagger(S)$  ( $S$  at the end of the game).

Alice aims to maximize the result, while Bob aims to minimize it.

Let  $R$  be the result when both players play optimally. In this problem, you play as Alice against the jury program playing as Bob. Your task is to implement a strategy for Alice such that the result of the game is always at least  $R$ .

$\dagger$  MEX of a set of integers  $c_1, c_2, \dots, c_k$  is defined as the smallest non-negative integer  $x$  which does not occur in the set  $c$ . For example,  $\text{MEX}(\{0, 1, 2, 4\}) = 3$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> cnt(n + 1);
8     for (int i = 0; i < n; i++) {
9         int x;
10        cin >> x;
11        if (x <= n) {
12            cnt[x]++;
13        }
14    }
15    int mex = 0;
16    while (cnt[mex]) {
17        mex++;
18    }
19    while (true) {
20        cout << mex << endl;
21        int y;
22        cin >> y;
23        if (y == -1) {
24            break;
25        }
26        mex = y;
27    }
28 }
29 int main() {

```

```

30     ios::sync_with_stdio(false);
31     cin.tie(nullptr);
32     int t;
33     cin >> t;
34     while (t--) {
35         solve();
36     }
37     return 0;
38 }
```

### 174: Madoka and Childish Pranks

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Madoka as a child was an extremely capricious girl, and one of her favorite pranks was drawing on her wall. According to Madoka's memories, the wall was a table of  $n$  rows and  $m$  columns, consisting only of zeroes and ones. The coordinate of the cell in the  $i$ -th row and the  $j$ -th column ( $1 \leq i \leq n$ ,  $1 \leq j \leq m$ ) is  $(i, j)$ .

One day she saw a picture “Mahou Shoujo Madoka Magica” and decided to draw it on her wall. Initially, the Madoka's table is a table of size  $n \times m$  filled with zeroes. Then she applies the following operation any number of times:

Madoka selects any rectangular subtable of the table and paints it in a chess coloring (the upper left corner of the subtable always has the color 0). Note that some cells may be colored several times. In this case, the final color of the cell is equal to the color obtained during the last repainting.

For better understanding of the statement, we recommend you to read the explanation of the first test.

Help Madoka and find some sequence of no more than  $n \cdot m$  operations that allows you to obtain the picture she wants, or determine that this is impossible.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<string> s(n);
```

```

8     for (int i = 0; i < n; i++) {
9         cin >> s[i];
10    }
11    if (s[0][0] == '1') {
12        cout << -1 << "\n";
13        return;
14    }
15    cout << n * m << "\n";
16    for (int i = n - 1; i >= 0; i--) {
17        for (int j = m - 1; j >= 0; j--) {
18            if (s[i][j] == '1') {
19                if (i) {
20                    cout << i << " " << j + 1 << " " << i + 1 << " " << j + 1 << "\n";
21                } else {
22                    cout << i + 1 << " " << j << " " << i + 1 << " " << j + 1 << "\n";
23                }
24            } else {
25                cout << i + 1 << " " << j + 1 << " " << i + 1 << " " << j + 1 << "\n";
26            }
27        }
28    }
29 }
30 int main() {
31     ios::sync_with_stdio(false);
32     cin.tie(nullptr);
33     int t;
34     cin >> t;
35     while (t--) {
36         solve();
37     }
38     return 0;
39 }
```

### 175: Two-Colored Dominoes

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There is an  $n \times m$  board divided into cells. There are also some dominoes on this board. Each domino covers two adjacent cells (that is, two cells that share a side), and no two dominoes overlap.

Piet thinks that this board is too boring and it needs to be painted. He will paint the cells of the dominoes black and white. He calls the painting beautiful if all of the following conditions hold:

for each domino, one of its cells is painted white and the other is painted black;

for each row, the number of black cells in this row equals the number of white cells in this row;

for each column, the number of black cells in this column equals the number of white cells in this column.

Note that the cells that are not covered by dominoes are not painted at all, they are counted as neither black nor white.

Help Piet produce a beautiful painting or tell that it is impossible.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<string> s(n);
8     for (int i = 0; i < n; i++) {
9         cin >> s[i];
10    }
11    vector<vector<array<int, 2>>> adj(n + m);
12    for (int i = 0; i < n; i++) {
13        for (int j = 0; j < m; j++) {
14            if (s[i][j] == 'U') {
15                adj[i].push_back({i + 1, j});
16                adj[i + 1].push_back({i, j});
17            }
18            if (s[i][j] == 'L') {
19                adj[n + j].push_back({n + j + 1, i});
20                adj[n + j + 1].push_back({n + j, i});
21            }
22        }
23    }
24    for (int i = 0; i < n + m; i++) {
25        if (adj[i].size() % 2) {
26            cout << "-1\n";
27            return;
28        }
29    }
30    auto dfs = [&](auto self, int x, int y, int z) -> void {
31        while (!adj[x].empty()) {
32            auto [i, j] = adj[x].back();
33            adj[x].pop_back();
34            if ((x < n ? s[x][j] : s[j][x - n]) == '.') {
35                continue;
36            }
37            if (x < n) {
38                s[x][j] = s[i][j] = '.';
39            } else {
40                s[j][x - n] = s[j][i - n] = '.';
41            }
42            self(self, i, x, j);
43        }
44        if (y != -1) {
45            if (x < n) {
46                s[x][z] = 'W';
47                s[y][z] = 'B';
48            } else {
49                s[z][x - n] = 'W';
50                s[z][y - n] = 'B';
51            }
52        }
53    };
54    for (int i = 0; i < n + m; i++) {

```

```

55         dfs(dfs, i, -1, -1);
56     }
57     for (int i = 0; i < n; i++) {
58         cout << s[i] << "\n";
59     }
60 }
61 int main() {
62     ios::sync_with_stdio(false);
63     cin.tie(nullptr);
64     int t;
65     cin >> t;
66     while (t--) {
67         solve();
68     }
69     return 0;
70 }
```

## 176: Kolya and Movie Theatre

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Recently, Kolya found out that a new movie theatre is going to be opened in his city soon, which will show a new movie every day for  $n$  days. So, on the day with the number  $1 \leq i \leq n$ , the movie theatre will show the premiere of the  $i$ -th movie. Also, Kolya found out the schedule of the movies and assigned the entertainment value to each movie, denoted by  $a_i$ .

However, the longer Kolya stays without visiting a movie theatre, the larger the decrease in entertainment value of the next movie. That decrease is equivalent to  $d \cdot cnt$ , where  $d$  is a predetermined value and  $cnt$  is the number of days since the last visit to the movie theatre. It is also known that Kolya managed to visit another movie theatre a day before the new one opened - the day with the number 0. So if we visit the movie theatre the first time on the day with the number  $i$ , then  $cnt$  - the number of days since the last visit to the movie theatre will be equal to  $i$ .

For example, if  $d = 2$  and  $a = [3, 2, 5, 4, 6]$ , then by visiting movies with indices 1 and 3,  $cnt$  value for the day 1 will be equal to  $1 - 0 = 1$  and  $cnt$  value for the day 3 will be  $3 - 1 = 2$ , so the total entertainment value of the movies will be  $a_1 - d \cdot 1 + a_3 - d \cdot 2 = 3 - 2 \cdot 1 + 5 - 2 \cdot 2 = 2$ .

Unfortunately, Kolya only has time to visit at most  $m$  movies. Help him create a plan to visit the cinema in such a way that the total entertainment value of all the movies he visits is maximized.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m, d;
6     cin >> n >> m >> d;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    i64 ans = 0;
12    i64 sum = 0;
13    priority_queue<int, vector<int>, greater<>> h;
14    for (int i = 0; i < n; i++) {
15        if (a[i] > 0) {
16            h.push(a[i]);
17            sum += a[i];
18        }
19        if (h.size() > m) {
20            sum -= h.top();
21            h.pop();
22        }
23        ans = max(ans, sum - 1LL * (i + 1) * d);
24    }
25    cout << ans << "\n";
26 }
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(nullptr);
30     int t;
31     cin >> t;
32     while (t--) {
33         solve();
34     }
35     return 0;
36 }
```

## 177: Row Major

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The row-major order of an  $r \times c$  grid of characters  $A$  is the string obtained by concatenating all the rows, i.e.

$$A_{11}A_{12}\dots A_{1c}A_{21}A_{22}\dots A_{2c}\dots A_{r1}A_{r2}\dots A_{rc}.$$

A grid of characters  $A$  is bad if there are some two adjacent cells (cells sharing an edge) with the same character.

You are given a positive integer  $n$ . Consider all strings  $s$  consisting of only lowercase Latin letters such that they are not the row-major order of any bad grid. Find any string with the minimum number of

distinct characters among all such strings of length  $n$ .

It can be proven that at least one such string exists under the constraints of the problem.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     int t = 1;
8     while (n % t == 0) {
9         t++;
10    }
11    for (int i = 0; i < n; i++) {
12        cout << char('a' + i % t);
13    }
14    cout << "\n";
15 }
16 int main() {
17     ios::sync_with_stdio(false);
18     cin.tie(nullptr);
19     int t;
20     cin >> t;
21     while (t--) {
22         solve();
23     }
24     return 0;
25 }
```

## 178: Insert Zero and Invert Prefix

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have a sequence  $a_1, a_2, \dots, a_n$  of length  $n$ , each element of which is either 0 or 1, and a sequence  $b$ , which is initially empty.

You are going to perform  $n$  operations. On each of them you will increase the length of  $b$  by 1.

On the  $i$ -th operation you choose an integer  $p$  between 0 and  $i - 1$ . You insert 0 in the sequence  $b$  on position  $p + 1$  (after the first  $p$  elements), and then you invert the first  $p$  elements of  $b$ .

More formally: let's denote the sequence  $b$  before the  $i$ -th ( $1 \leq i \leq n$ ) operation as  $b_1, b_2, \dots, b_{i-1}$ . On the  $i$ -th operation you choose an integer  $p$  between 0 and  $i - 1$  and replace  $b$  with  $\overline{b_1}, \overline{b_2}, \dots, \overline{b_p}, 0, b_{p+1}, b_{p+2}, \dots, b_{i-1}$ . Here,  $\overline{x}$  denotes the binary inversion. Hence,  $\overline{0} = 1$  and  $\overline{1} = 0$ .

You can find examples of operations in the Notes section.

Determine if there exists a sequence of operations that makes  $b$  equal to  $a$ . If such sequence of operations exists, find it.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    if (a.back() != 0) {
12        cout << "NO\n";
13        return;
14    }
15    cout << "YES\n";
16    vector<int> ans;
17    for (int i = 0, j = -1; i < n; i++) {
18        if (a[i] == 0) {
19            ans.push_back(i - j - 1);
20            for (int k = j + 1; k < i; k++) {
21                ans.push_back(0);
22            }
23            j = i;
24        }
25    }
26    reverse(ans.begin(), ans.end());
27    for (int i = 0; i < n; i++) {
28        cout << ans[i] << " \n"[i == n - 1];
29    }
30 }
31 int main() {
32     ios::sync_with_stdio(false);
33     cin.tie(nullptr);
34     int t;
35     cin >> t;
36     while (t--) {
37         solve();
38     }
39     return 0;
40 }
```

## 179: No Prime Differences

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given integers  $n$  and  $m$ . Fill an  $n$  by  $m$  grid with the integers 1 through  $n \cdot m$ , in such a way that for any two adjacent cells in the grid, the absolute difference of the values in those cells is not a prime number. Two cells in the grid are considered adjacent if they share a side.

It can be shown that under the given constraints, there is always a solution.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<vector<int>> a(n, vector<int>(m));
8     if (m % 2 == 0) {
9         for (int i = 0; i < n; i++) {
10            for (int j = 0; j < m; j++) {
11                a[i][j] = i * m + j + 1;
12            }
13        }
14    } else {
15        for (int i = 0; i < n; i++) {
16            for (int j = 0; j < m; j++) {
17                a[i][j] = i * m + (i + j) % m + 1;
18            }
19        }
20    }
21    for (int i = 0; i < n; i++) {
22        for (int j = 0; j < m; j++) {
23            cout << a[i][j] << " \n"[j == m - 1];
24        }
25    }
26 }
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(nullptr);
30     int t;
31     cin >> t;
32     while (t--) {
33         solve();
34     }
35     return 0;
36 }
```

## 180: Bracket Coloring

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

A regular bracket sequence is a bracket sequence that can be transformed into a correct arithmetic

expression by inserting characters “1” and “+” between the original characters of the sequence. For example:

the bracket sequences “()()” and “((())” are regular (the resulting expressions are: “(1)+(1)” and “((1+1)+1)”);

the bracket sequences “)”,“(” and “)” are not.

A bracket sequence is called beautiful if one of the following conditions is satisfied:

it is a regular bracket sequence;

if the order of the characters in this sequence is reversed, it becomes a regular bracket sequence.

For example, the bracket sequences “()()”, “((())”, “)))))(((), “))()((” are beautiful.

You are given a bracket sequence  $s$ . You have to color it in such a way that:

every bracket is colored into one color;

for every color, there is at least one bracket colored into that color;

for every color, if you write down the sequence of brackets having that color in the order they appear, you will get a beautiful bracket sequence.

Color the given bracket sequence  $s$  into the minimum number of colors according to these constraints, or report that it is impossible.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     string s;
8     cin >> s;
9     int c = count(s.begin(), s.end(), '(');
10    if (2 * c != n) {
11        cout << -1 << "\n";
12        return;
13    }
14    vector<int> ans(n, 2);
15    vector<int> a;
16    for (int i = 0; i < n; i++) {
17        if (s[i] == '(') {
18            a.push_back(i);
19        } else if (!a.empty()) {
20            ans[i] = 1;
21            ans[a.back()] = 1;
22            a.pop_back();
23        }
}

```

```

24     }
25     if (count(ans.begin(), ans.end(), 2)) {
26         a.clear();
27         ans.assign(n, 2);
28         for (int i = 0; i < n; i++) {
29             if (s[i] == '1') {
30                 a.push_back(i);
31             } else if (!a.empty()) {
32                 ans[i] = 1;
33                 ans[a.back()] = 1;
34                 a.pop_back();
35             }
36         }
37     }
38     cout << *max_element(ans.begin(), ans.end()) << "\n";
39     for (int i = 0; i < n; i++) {
40         cout << ans[i] << " \n"[i == n - 1];
41     }
42 }
43 int main() {
44     ios::sync_with_stdio(false);
45     cin.tie(nullptr);
46     int t;
47     cin >> t;
48     while (t--) {
49         solve();
50     }
51     return 0;
52 }
```

## 181: Flipper

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a permutation  $p$  of length  $n$ .

A permutation is an array consisting of  $n$  distinct integers from 1 to  $n$  in any order. For example,  $\{2, 3, 1, 5, 4\}$  is a permutation, while  $\{1, 2, 2\}$  is not (since 2 appears twice), and  $\{1, 3, 4\}$  is also not a permutation (as  $n = 3$ , but the array contains 4).

To the permutation  $p$ , you need to apply the following operation exactly once:

First you choose a segment  $[l, r]$  ( $1 \leq l \leq r \leq n$ , a segment is a continuous sequence of numbers  $\{p_l, p_{l+1}, \dots, p_{r-1}, p_r\}$ ) and reverse it. Reversing a segment means swapping pairs of numbers  $(p_l, p_r)$ ,  $(p_{l+1}, p_{r-1}), \dots, (p_{l+i}, p_{r-i})$  (where  $l + i \leq r - i$ ).

Then you swap the prefix and suffix:  $[r + 1, n]$  and  $[1, l - 1]$  (note that these segments may be empty).

For example, given  $n = 5, p = \{2, 3, 1, 5, 4\}$ , if you choose the segment  $[l = 2, r = 3]$ , after reversing the segment  $p = \{2, 1, 3, 5, 4\}$ , then you swap the segments  $[4, 5]$  and  $[1, 1]$ . Thus,  $p = \{5, 4, 1, 3, 2\}$ . It

can be shown that this is the maximum possible result for the given permutation.

You need to output the lexicographically maximum permutation that can be obtained by applying the operation described exactly once.

A permutation  $a$  is lexicographically greater than permutation  $b$  if there exists an  $i$  ( $1 \leq i \leq n$ ) such that  $a_j = b_j$  for  $1 \leq j < i$  and  $a_i > b_i$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> p(n);
8     for (int i = 0; i < n; i++) {
9         cin >> p[i];
10    }
11    vector ans(p.rbegin(), p.rend());
12    int x = find(p.begin(), p.end(), n) - p.begin();
13    if (x == 0 && n > 1) {
14        x = find(p.begin(), p.end(), n - 1) - p.begin();
15    }
16    for (int i = 0; i < n; i++) {
17        auto q = p;
18        reverse(q.begin() + i, q.end());
19        rotate(q.begin(), q.begin() + i, q.end());
20        ans = max(ans, q);
21    }
22    for (int i = 0; i < x; i++) {
23        auto q = p;
24        reverse(q.begin() + i, q.begin() + x);
25        rotate(q.begin(), q.begin() + i, q.end());
26        rotate(q.begin(), q.begin() + x - i, q.end() - i);
27        ans = max(ans, q);
28    }
29    for (int i = 0; i < n; i++) {
30        cout << ans[i] << " \n"[i == n - 1];
31    }
32 }
33 int main() {
34     ios::sync_with_stdio(false);
35     cin.tie(nullptr);
36     int t;
37     cin >> t;
38     while (t--) {
39         solve();
40     }
41     return 0;
42 }
```

**182: Find Color**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Not so long ago as a result of combat operations the main Berland place of interest - the magic clock - was damaged. The cannon's balls made several holes in the clock, that's why the residents are concerned about the repair. The magic clock can be represented as an infinite Cartesian plane, where the origin corresponds to the clock center. The clock was painted two colors as is shown in the picture:

The picture shows only the central part of the clock. This coloring naturally extends to infinity.

The balls can be taken to be points on the plane. Your task is to find the color of the area, damaged by the given ball.

All the points located on the border of one of the areas have to be considered painted black.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int x, y;
8     cin >> x >> y;
9     if (!x || !y) {
10         cout << "black\n";
11     } else {
12         int z = x * x + y * y;
13         int w = sqrt(z);
14         if (w * w == z || ((w % 2) ^ (x < 0) ^ (y < 0)) == 0) {
15             cout << "black\n";
16         } else {
17             cout << "white\n";
18         }
19     }
20     return 0;
21 }
```

**183: Li Hua and Chess**

- Time limit: 1 second
- Memory limit: 256 megabytes

- Input file: standard input
- Output file: standard output

This is an interactive problem.

Li Ming and Li Hua are playing a game. Li Hua has a chessboard of size  $n \times m$ . Denote  $(r, c)$  ( $1 \leq r \leq n, 1 \leq c \leq m$ ) as the cell on the  $r$ -th row from the top and on the  $c$ -th column from the left. Li Ming put a king on the chessboard and Li Hua needs to guess its position.

Li Hua can ask Li Ming no more than 3 questions. In each question, he can choose a cell and ask the minimum steps needed to move the king to the chosen cell. Each question is independent, which means the king doesn't actually move.

A king can move from  $(x, y)$  to  $(x', y')$  if and only if  $\max\{|x - x'|, |y - y'|\} = 1$  (shown in the following picture).

The position of the king is chosen before the interaction.

Suppose you were Li Hua, please solve this problem.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int query(int r, int c) {
5     cout << "? " << r << " " << c << endl;
6     int ans;
7     cin >> ans;
8     return ans;
9 }
10 void solve() {
11     int n, m;
12     cin >> n >> m;
13     int a = query(1, 1);
14     int b = query(n, m);
15     int x, y;
16     if (a + b == n - 1) {
17         x = 1 + a;
18         y = 1 + query(x, 1);
19     } else if (a + b == m - 1) {
20         y = 1 + a;
21         x = 1 + query(1, y);
22     } else {
23         if (1 + a <= n && m - b >= 1 && query(1 + a, m - b) == 0) {
24             x = 1 + a;
25             y = m - b;
26         } else {
27             x = n - b;
28             y = 1 + a;
29         }
30     }
31     cout << "! " << x << " " << y << endl;
32 }
33 int main() {

```

```

34     ios::sync_with_stdio(false);
35     cin.tie(nullptr);
36     int t;
37     cin >> t;
38     while (t--) {
39         solve();
40     }
41     return 0;
42 }
```

### 184: Search in Parallel

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Suppose you have  $n$  boxes. The  $i$ -th box contains infinitely many balls of color  $i$ . Sometimes you need to get a ball with some specific color; but you're too lazy to do it yourself.

You have bought two robots to retrieve the balls for you. Now you have to program them. In order to program the robots, you have to construct two lists  $[a_1, a_2, \dots, a_k]$  and  $[b_1, b_2, \dots, b_{n-k}]$ , where the list  $a$  represents the boxes assigned to the first robot, and the list  $b$  represents the boxes assigned to the second robot. Every integer from 1 to  $n$  must be present in exactly one of these lists.

When you request a ball with color  $x$ , the robots work as follows. Each robot looks through the boxes that were assigned to that robot, in the order they appear in the list. The first robot spends  $s_1$  seconds analyzing the contents of a box; the second robot spends  $s_2$ . As soon as one of the robots finds the box with balls of color  $x$  (and analyzes its contents), the search ends. The search time is the number of seconds from the beginning of the search until one of the robots finishes analyzing the contents of the  $x$ -th box. If a robot analyzes the contents of all boxes assigned to it, it stops searching.

For example, suppose  $s_1 = 2$ ,  $s_2 = 3$ ,  $a = [4, 1, 5, 3, 7]$ ,  $b = [2, 6]$ . If you request a ball with color 3, the following happens:

initially, the first robot starts analyzing the box 4, and the second robot starts analyzing the box 2;

at the end of the 2-nd second, the first robot finishes analyzing the box 4. It is not the box you need, so the robot continues with the box 1;

at the end of the 3-rd second, the second robot finishes analyzing the box 2. It is not the box you need, so the robot continues with the box 6;

at the end of the 4-th second, the first robot finishes analyzing the box 1. It is not the box you need, so the robot continues with the box 5;

at the end of the 6-th second, the first robot finishes analyzing the box 5. It is not the box you need, so the robot continues with the box 3. At the same time, the second robot finishes analyzing the box 6. It is not the box you need, and the second robot has analyzed all the boxes in its list, so that robot stops searching;

at the end of the 8-th second, the first robot finishes analyzing the box 3. It is the box you need, so the search ends;

so, the search time is 8 seconds.

You know that you are going to request a ball of color 1  $r_1$  times, a ball of color 2  $r_2$  times, and so on. You want to construct the lists  $a$  and  $b$  for the robots in such a way that the total search time over all requests is the minimum possible.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, s1, s2;
6     cin >> n >> s1 >> s2;
7     vector<int> r(n);
8     for (int i = 0; i < n; i++) {
9         cin >> r[i];
10    }
11    vector<int> p(n);
12    iota(p.begin(), p.end(), 0);
13    sort(p.begin(), p.end(), [&](int i, int j) {
14        return r[i] > r[j];
15    });
16    vector<int> a, b;
17    for (auto i : p) {
18        if ((a.size() + 1) * s1 < (b.size() + 1) * s2) {
19            a.push_back(i + 1);
20        } else {
21            b.push_back(i + 1);
22        }
23    }
24    cout << a.size();
25    for (auto x : a) {
26        cout << " " << x;
27    }
28    cout << "\n";
29    cout << b.size();
30    for (auto x : b) {
31        cout << " " << x;
32    }
33    cout << "\n";
34 }
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int t;

```

```

39     cin >> t;
40     while (t--) {
41         solve();
42     }
43     return 0;
44 }
```

### 185: Umka and a Long Flight

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The girl Umka loves to travel and participate in math olympiads. One day she was flying by plane to the next olympiad and out of boredom explored a huge checkered sheet of paper.

Denote the  $n$ -th Fibonacci number as  $F_n = \begin{cases} 1, & n = 0; \\ 1, & n = 1; \\ F_{n-2} + F_{n-1}, & n \geq 2. \end{cases}$

A checkered rectangle with a height of  $F_n$  and a width of  $F_{n+1}$  is called a Fibonacci rectangle of order  $n$ .

Umka has a Fibonacci rectangle of order  $n$ . Someone colored a cell in it at the intersection of the row  $x$  and the column  $y$ .

It is necessary to cut this rectangle exactly into  $n + 1$  squares in such way that

the painted cell was in a square with a side of 1;

there was at most one pair of squares with equal sides;

the side of each square was equal to a Fibonacci number.

Will Umka be able to cut this rectangle in that way?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 i64 F[50];
5 void solve() {
6     int n;
7     i64 x, y;
```

```

8     cin >> n >> x >> y;
9     while (n > 0) {
10         if (y <= F[n - 1]) {
11             } else if (y > F[n]) {
12                 y -= F[n];
13             } else {
14                 cout << "NO\n";
15                 return;
16             }
17             swap(x, y);
18             n -= 1;
19         }
20         cout << "YES\n";
21     }
22 int main() {
23     ios::sync_with_stdio(false);
24     cin.tie(nullptr);
25     F[0] = F[1] = 1;
26     for (int i = 2; i < 50; i++) {
27         F[i] = F[i - 2] + F[i - 1];
28     }
29     int t;
30     cin >> t;
31     while (t--) {
32         solve();
33     }
34     return 0;
35 }
```

## 186: Shocking Arrangement

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a_1, a_2, \dots, a_n$  consisting of integers such that  $a_1 + a_2 + \dots + a_n = 0$ .

You have to rearrange the elements of the array  $a$  so that the following condition is satisfied:

$$\max_{1 \leq l \leq r \leq n} |a_l + a_{l+1} + \dots + a_r| < \max(a_1, a_2, \dots, a_n) - \min(a_1, a_2, \dots, a_n),$$

where  $|x|$  denotes the absolute value of  $x$ .

More formally, determine if there exists a permutation  $p_1, p_2, \dots, p_n$  that for the array  $a_{p_1}, a_{p_2}, \dots, a_{p_n}$ , the condition above is satisfied, and find the corresponding array.

Recall that the array  $p_1, p_2, \dots, p_n$  is called a permutation if for each integer  $x$  from 1 to  $n$  there is exactly one  $i$  from 1 to  $n$  such that  $p_i = x$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    if (a == vector(n, 0)) {
12        cout << "NO\n";
13        return;
14    }
15    cout << "YES\n";
16    i64 s = 0;
17    int i = 0, j = n - 1;
18    sort(a.begin(), a.end());
19    vector<int> ans;
20    while (i <= j) {
21        if (s <= 0) {
22            s += a[j];
23            ans.push_back(a[j--]);
24        } else {
25            s += a[i];
26            ans.push_back(a[i++]);
27        }
28    }
29    for (int i = 0; i < n; i++) {
30        cout << ans[i] << " \n"[i == n - 1];
31    }
32 }
33 int main() {
34     ios::sync_with_stdio(false);
35     cin.tie(nullptr);
36     int t;
37     cin >> t;
38     while (t--) {
39         solve();
40     }
41     return 0;
42 }
```

## 187: Sum on Subarrays

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

For an array  $a = [a_1, a_2, \dots, a_n]$ , let's denote its subarray  $a[l, r]$  as the array  $[a_l, a_{l+1}, \dots, a_r]$ .

For example, the array  $a = [1, -3, 1]$  has 6 non-empty subarrays:

$a[1, 1] = [1];$

$a[1, 2] = [1, -3];$

$a[1, 3] = [1, -3, 1];$

$a[2, 2] = [-3];$

$a[2, 3] = [-3, 1];$

$a[3, 3] = [1].$

You are given two integers  $n$  and  $k$ . Construct an array  $a$  consisting of  $n$  integers such that:

all elements of  $a$  are from  $-1000$  to  $1000$ ;

$a$  has exactly  $k$  subarrays with positive sums;

the rest  $\frac{(n+1) \cdot n}{2} - k$  subarrays of  $a$  have negative sums.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     int x = 0;
8     while ((x + 1) * (x + 2) / 2 <= k) {
9         x += 1;
10    }
11    vector<int> a(n);
12    for (int i = 0; i < n; i++) {
13        if (i < x) {
14            a[i] = 2;
15        } else if (i == x) {
16            a[i] = -2 * x - 1 + 2 * (k - x * (x + 1) / 2);
17        } else {
18            a[i] = -1000;
19        }
20        cout << a[i] << " \n"[i == n - 1];
21    }
22 }
23 int main() {
24     ios::sync_with_stdio(false);
25     cin.tie(nullptr);
26     int t;
27     cin >> t;
28     while (t--) {
29         solve();
30     }
31     return 0;
32 }
```

## 188: Sequence Master

- Time limit: 1 second
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

For some positive integer  $m$ , YunQian considers an array  $q$  of  $2m$  (possibly negative) integers good, if and only if for every possible subsequence of  $q$  that has length  $m$ , the product of the  $m$  elements in the subsequence is equal to the sum of the  $m$  elements that are not in the subsequence. Formally, let  $U = \{1, 2, \dots, 2m\}$ . For all sets  $S \subseteq U$  such that  $|S| = m$ ,  $\prod_{i \in S} q_i = \sum_{i \in U \setminus S} q_i$ .

Define the distance between two arrays  $a$  and  $b$  both of length  $k$  to be  $\sum_{i=1}^k |a_i - b_i|$ .

You are given a positive integer  $n$  and an array  $p$  of  $2n$  integers.

Find the minimum distance between  $p$  and  $q$  over all good arrays  $q$  of length  $2n$ . It can be shown for all positive integers  $n$ , at least one good array exists. Note that you are not required to construct the array  $q$  that achieves this minimum distance.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> p(2 * n);
8     for (int i = 0; i < 2 * n; i++) {
9         cin >> p[i];
10    }
11    if (n == 1) {
12        cout << abs(p[0] - p[1]) << "\n";
13        return;
14    }
15    i64 ans = 0;
16    for (int i = 0; i < 2 * n; i++) {
17        ans += abs(p[i]);
18    }
19    if (n == 2) {
20        i64 res = 0;
21        for (int i = 0; i < 2 * n; i++) {
22            res += abs(p[i] - 2);
23        }
24        ans = min(ans, res);
25    }
26    if (n % 2 == 0) {
27        i64 res = 0;
28        for (int i = 0; i < 2 * n; i++) {
29            res += abs(p[i] + 1);
30        }
31    }
32 }
```

```

30         }
31     for (int i = 0; i < 2 * n; i++) {
32         ans = min(ans, res - abs(p[i] + 1) + abs(p[i] - n));
33     }
34 }
35 cout << ans << "\n";
36 }
37 int main() {
38     ios::sync_with_stdio(false);
39     cin.tie(nullptr);
40     int t;
41     cin >> t;
42     while (t--) {
43         solve();
44     }
45     return 0;
46 }
```

### 189: Chris and Magic Square

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

ZS the Coder and Chris the Baboon arrived at the entrance of Udayland. There is a  $n \times n$  magic grid on the entrance which is filled with integers. Chris noticed that exactly one of the cells in the grid is empty, and to enter Udayland, they need to fill a positive integer into the empty cell.

Chris tried filling in random numbers but it didn't work. ZS the Coder realizes that they need to fill in a positive integer such that the numbers in the grid form a magic square. This means that he has to fill in a positive integer so that the sum of the numbers in each row of the grid (), each column of the grid (), and the two long diagonals of the grid (the main diagonal - and the secondary diagonal - ) are equal.

Chris doesn't know what number to fill in. Can you help Chris find the correct positive integer to fill in or determine that it is impossible?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<i64> a(n, vector<i64>(n));
10    int r, c;
11    for (int i = 0; i < n; i++) {
```

```

12         for (int j = 0; j < n; j++) {
13             cin >> a[i][j];
14             if (a[i][j] == 0) {
15                 r = i, c = j;
16             }
17         }
18     }
19     if (n == 1) {
20         cout << 1 << "\n";
21         return 0;
22     }
23     i64 sum = accumulate(a[(r + 1) % n].begin(), a[(r + 1) % n].end(), 0LL);
24     i64 csum = accumulate(a[r].begin(), a[r].end(), 0LL);
25     i64 x = sum - csum;
26     if (x <= 0) {
27         cout << -1 << "\n";
28         return 0;
29     }
30     a[r][c] = x;
31     for (int i = 0; i < n; i++) {
32         if (accumulate(a[i].begin(), a[i].end(), 0LL) != sum) {
33             cout << -1 << "\n";
34             return 0;
35         }
36         i64 sc = 0;
37         for (int j = 0; j < n; j++) {
38             sc += a[j][i];
39         }
40         if (sc != sum) {
41             cout << -1 << "\n";
42             return 0;
43         }
44     }
45     i64 s1 = 0, s2 = 0;
46     for (int i = 0; i < n; i++) {
47         s1 += a[i][i];
48         s2 += a[i][n - 1 - i];
49     }
50     if (s1 != sum || s2 != sum) {
51         cout << -1 << "\n";
52         return 0;
53     }
54     cout << x << "\n";
55     return 0;
56 }
```

## 190: The Very Beautiful Blanket

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Kirill wants to weave the very beautiful blanket consisting of  $n \times m$  of the same size square patches of some colors. He matched some non-negative integer to each color. Thus, in our problem, the blanket can be considered a  $B$  matrix of size  $n \times m$  consisting of non-negative integers.

Kirill considers that the blanket is very beautiful, if for each submatrix  $A$  of size  $4 \times 4$  of the matrix  $B$  is true:

$$A_{11} \oplus A_{12} \oplus A_{21} \oplus A_{22} = A_{33} \oplus A_{34} \oplus A_{43} \oplus A_{44},$$

$$A_{13} \oplus A_{14} \oplus A_{23} \oplus A_{24} = A_{31} \oplus A_{32} \oplus A_{41} \oplus A_{42},$$

where  $\oplus$  means bitwise exclusive OR

Kirill asks you to help her weave a very beautiful blanket, and as colorful as possible!

He gives you two integers  $n$  and  $m$ .

Your task is to generate a matrix  $B$  of size  $n \times m$ , which corresponds to a very beautiful blanket and in which the number of different numbers maximized.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     cout << n * m << "\n";
8     for (int i = 0; i < n; i++) {
9         for (int j = 0; j < m; j++) {
10             cout << (i << 10) + j << " \n"[j == m - 1];
11         }
12     }
13 }
14 int main() {
15     ios::sync_with_stdio(false);
16     cin.tie(nullptr);
17     int t;
18     cin >> t;
19     while (t--) {
20         solve();
21     }
22     return 0;
23 }
```

## dp

### 191: Anji's Binary Tree

- Time limit: 2.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Keksic keeps getting left on seen by Anji. Through a mutual friend, he's figured out that Anji really likes binary trees and decided to solve her problem in order to get her attention.

Anji has given Keksic a binary tree with  $n$  vertices. Vertex 1 is the root and does not have a parent. All other vertices have exactly one parent. Each vertex can have up to 2 children, a left child, and a right child. For each vertex, Anji tells Keksic index of both its left and its right child or tells him that they do not exist.

Additionally, each of the vertices has a letter  $s_i$  on it, which is either 'U', 'L' or 'R'.

Keksic begins his journey on the root, and in each move he does the following:

If the letter on his current vertex is 'U', he moves to its parent. If it doesn't exist, he does nothing.

If the letter on his current vertex is 'L', he moves to its left child. If it doesn't exist, he does nothing.

If the letter on his current vertex is 'R', he moves to its right child. If it doesn't exist, he does nothing.

You are interested in the minimal number of operations he needs to do before his journey, such that when he starts his journey, he will reach a leaf at some point. A leaf is a vertex that has no children. It does not matter which leaf he reaches. Note that it does not matter whether he will stay in the leaf, he just needs to move to it. Additionally, note that it does not matter how many times he needs to move before reaching a leaf.

Help Keksic solve Anji's tree so that he can win her heart, and make her come to aak.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     string s;
8     cin >> s;
9     vector<int> l(n), r(n);
10    for (int i = 0; i < n; i++) {
11        cin >> l[i] >> r[i];
12        l[i]--, r[i]--;
13    }
14    auto dfs = [&](auto self, int x) -> int {
15        if (l[x] == -1 && r[x] == -1) {
16            return 0;
17        }
18        int ans = n;
19        if (l[x] != -1) {
20            ans = self(self, l[x]) + (s[x] != 'L');
21        }
22        if (r[x] != -1) {
23            ans = min(ans, self(self, r[x]) + (s[x] != 'R'));
24        }
25    return ans;
}

```

```

26     };
27     int ans = dfs(dfs, 0);
28     cout << ans << "\n";
29 }
30 int main() {
31     ios::sync_with_stdio(false);
32     cin.tie(nullptr);
33     int t;
34     cin >> t;
35     while (t--) {
36         solve();
37     }
38     return 0;
39 }
```

## 192: Block Sequence

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Given a sequence of integers  $a$  of length  $n$ .

A sequence is called beautiful if it has the form of a series of blocks, each starting with its length, i.e., first comes the length of the block, and then its elements. For example, the sequences [3, 3, 4, 5, 2, 6, 1] and [1, 8, 4, 5, 2, 6, 1] are beautiful (different blocks are colored differently), while [1], [1, 4, 3], [3, 2, 1] are not.

In one operation, you can remove any element from the sequence. What is the minimum number of operations required to make the given sequence beautiful?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<int> dp(n + 1, n);
12    dp[n] = 0;
13    for (int i = n - 1; i >= 0; i--) {
14        dp[i] = dp[i + 1] + 1;
15        if (i + a[i] < n) {
```

```

16         dp[i] = min(dp[i], dp[i + a[i] + 1]);
17     }
18 }
19 cout << dp[0] << "\n";
20 }
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     int t;
25     cin >> t;
26     while (t--) {
27         solve();
28     }
29     return 0;
30 }
```

### 194: Game on Permutation

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Alice and Bob are playing a game. They have a permutation  $p$  of size  $n$  (a permutation of size  $n$  is an array of size  $n$  where each element from 1 to  $n$  occurs exactly once). They also have a chip, which can be placed on any element of the permutation.

Alice and Bob make alternating moves: Alice makes the first move, then Bob makes the second move, then Alice makes the third move, and so on. During the first move, Alice chooses any element of the permutation and places the chip on that element. During each of the next moves, the current player has to move the chip to any element that is simultaneously to the left and strictly less than the current element (i. e. if the chip is on the  $i$ -th element, it can be moved to the  $j$ -th element if  $j < i$  and  $p_j < p_i$ ). If a player cannot make a move (it is impossible to move the chip according to the rules of the game), that player wins the game.

Let's say that the  $i$ -th element of the permutation is lucky if the following condition holds:

if Alice places the chip on the  $i$ -th element during her first move, she can win the game no matter how Bob plays (i. e. she has a winning strategy).

You have to calculate the number of lucky elements in the permutation.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
```

```

2  using namespace std;
3  using i64 = long long;
4  void solve() {
5      int n;
6      cin >> n;
7      vector<int> p(n);
8      for (int i = 0; i < n; i++) {
9          cin >> p[i];
10         p[i]--;
11     }
12     int minlose = n;
13     int min = n;
14     int ans = 0;
15     for (int i = 0; i < n; i++) {
16         int win = 0;
17         if (p[i] < min) {
18             min = p[i];
19             win = 1;
20         } else {
21             win = (minlose < p[i]);
22         }
23         if (!win) {
24             ans += 1;
25             minlose = min(minlose, p[i]);
26         }
27     }
28     cout << ans << "\n";
29 }
30 int main() {
31     ios::sync_with_stdio(false);
32     cin.tie(nullptr);
33     int t;
34     cin >> t;
35     while (t--) {
36         solve();
37     }
38     return 0;
39 }
```

## 195: Nastya and Potions

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Alchemist Nastya loves mixing potions. There are a total of  $n$  types of potions, and one potion of type  $i$  can be bought for  $c_i$  coins.

Any kind of potions can be obtained in no more than one way, by mixing from several others. The potions used in the mixing process will be consumed. Moreover, no potion can be obtained from itself through one or more mixing processes.

As an experienced alchemist, Nastya has an unlimited supply of  $k$  types of potions  $p_1, p_2, \dots, p_k$ , but she doesn't know which one she wants to obtain next. To decide, she asks you to find, for each

$1 \leq i \leq n$ , the minimum number of coins she needs to spend to obtain a potion of type  $i$  next.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> c(n);
8     for (int i = 0; i < n; i++) {
9         cin >> c[i];
10    }
11    vector<bool> have(n);
12    for (int i = 0; i < k; i++) {
13        int x;
14        cin >> x;
15        x--;
16        have[x] = true;
17    }
18    vector<bool> vis(n);
19    vector<vector<int>> ing(n);
20    for (int i = 0; i < n; i++) {
21        int m;
22        cin >> m;
23        while (m--) {
24            int x;
25            cin >> x;
26            x--;
27            ing[i].push_back(x);
28        }
29    }
30    auto rec = [&](auto self, int x) -> int {
31        if (vis[x])
32            return c[x];
33        if (have[x]) {
34            c[x] = 0;
35        }
36        if (!ing[x].empty()) {
37            i64 res = 0;
38            for (auto y : ing[x]) {
39                res += self(self, y);
40            }
41            c[x] = min(1LL * c[x], res);
42        }
43        vis[x] = true;
44        return c[x];
45    };
46    for (int i = 0; i < n; i++) {
47        cout << rec(rec, i) << " \n"[i == n - 1];
48    }
49 }
50 }
51 int main() {
52     ios::sync_with_stdio(false);
53     cin.tie(nullptr);
54     int t;
55     cin >> t;

```

```

56     while (t--) {
57         solve();
58     }
59     return 0;
60 }
```

**196: Strong Password**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp finally got the courage to register on ForceCoders. He came up with a handle but is still thinking about the password.

He wants his password to be as strong as possible, so he came up with the following criteria:

the length of the password should be exactly  $m$ ;

the password should only consist of digits from 0 to 9;

the password should not appear in the password database (given as a string  $s$ ) as a subsequence (not necessarily contiguous).

Monocarp also came up with two strings of length  $m$ :  $l$  and  $r$ , both consisting only of digits from 0 to 9. He wants the  $i$ -th digit of his password to be between  $l_i$  and  $r_i$ , inclusive.

Does there exist a password that fits all criteria?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     string s;
6     cin >> s;
7     int m;
8     cin >> m;
9     string l, r;
10    cin >> l >> r;
11    int p = 0;
12    for (int i = 0; i < m; i++) {
13        int q = p;
14        for (char x = l[i]; x <= r[i]; x++) {
15            if (s.find(x, p) == -1) {
16                cout << "YES\n";
17                return;
18            }
19        }
20    }
21 }
```

```

18         }
19         q = max(q, int(s.find(x, p)) + 1);
20     }
21     p = q;
22 }
23 cout << "NO\n";
24 }
25 int main() {
26     ios::sync_with_stdio(false);
27     cin.tie(nullptr);
28     int t;
29     cin >> t;
30     while (t--) {
31         solve();
32     }
33     return 0;
34 }
```

### 197: Tenzing and Balls

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Tenzing has  $n$  balls arranged in a line. The color of the  $i$ -th ball from the left is  $a_i$ .

Tenzing can do the following operation any number of times:

select  $i$  and  $j$  such that  $1 \leq i < j \leq |a|$  and  $a_i = a_j$ ,

remove  $a_i, a_{i+1}, \dots, a_j$  from the array (and decrease the indices of all elements to the right of  $a_j$  by  $j - i + 1$ ).

Tenzing wants to know the maximum number of balls he can remove.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10        a[i]--;
11    }
12    vector<int> dp(n + 1), f(n, -1E9);
13    for (int i = 0; i < n; i++) {
14        dp[i + 1] = max(dp[i], f[a[i]] + i + 1);
15        f[a[i]] = max(f[a[i]], dp[i] - i);
```

```

16      }
17      cout << dp[n] << "\n";
18  }
19 int main() {
20     ios::sync_with_stdio(false);
21     cin.tie(nullptr);
22     int t;
23     cin >> t;
24     while (t--) {
25         solve();
26     }
27     return 0;
28 }
```

## 198: Copil Copac Draws Trees

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Copil Copac is given a list of  $n - 1$  edges describing a tree of  $n$  vertices. He decides to draw it using the following algorithm:

Step 0: Draws the first vertex (vertex 1). Go to step 1.

Step 1: For every edge in the input, in order: if the edge connects an already drawn vertex  $u$  to an undrawn vertex  $v$ , he will draw the undrawn vertex  $v$  and the edge. After checking every edge, go to step 2.

Step 2: If all the vertices are drawn, terminate the algorithm. Else, go to step 1.

The number of readings is defined as the number of times Copil Copac performs step 1.

Find the number of readings needed by Copil Copac to draw the tree.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<vector<pair<int, int>>> adj(n);
8     vector<int> f(n), e(n);
9     e[0] = n;
10    for (int i = 0; i < n - 1; i++) {
11        int u, v;
12        cin >> u >> v;
13        u--, v--;
```

```

14         adj[u].emplace_back(v, i);
15         adj[v].emplace_back(u, i);
16     }
17     function<void(int, int)> dfs = [&](int x, int p) {
18         for (auto [y, i] : adj[x]) {
19             if (y == p) {
20                 continue;
21             }
22             e[y] = i;
23             f[y] = f[x] + (i < e[x]);
24             dfs(y, x);
25         }
26     };
27     dfs(0, -1);
28     auto ans = *max_element(f.begin(), f.end());
29     cout << ans << "\n";
30 }
31 int main() {
32     ios::sync_with_stdio(false);
33     cin.tie(nullptr);
34     int t;
35     cin >> t;
36     while (t--) {
37         solve();
38     }
39     return 0;
40 }
```

## 199: Round Table Knights

- Time limit: 0.5 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  knights sitting at the Round Table at an equal distance from each other. Each of them is either in a good or in a bad mood.

Merlin, the wizard predicted to King Arthur that the next month will turn out to be particularly fortunate if the regular polygon can be found. On all vertices of the polygon knights in a good mood should be located. Otherwise, the next month will bring misfortunes.

A convex polygon is regular if all its sides have same length and all his angles are equal. In this problem we consider only regular polygons with at least 3 vertices, i. e. only nondegenerated.

On a picture below some examples of such polygons are present. Green points mean knights in a good mood. Red points mean ones in a bad mood.

King Arthur knows the knights' moods. Help him find out if the next month will be fortunate or not.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    vector<bool> check(n+1);
14    for (int i = 3; i <= n; i++) {
15        if (check[i]) {
16            continue;
17        }
18        if (n % i) {
19            continue;
20        }
21        for (int j = i; j <= n; j += i) {
22            check[j] = true;
23        }
24        for (int k = 0; k < n/i; k++) {
25            bool ok = true;
26            for (int x = k; x < n; x += n/i) {
27                if (!a[x]) {
28                    ok = false;
29                }
30            }
31            if (ok) {
32                cout << "YES\n";
33                return 0;
34            }
35        }
36    }
37    cout << "NO\n";
38    return 0;
39 }
```

## 200: Anfisa the Monkey

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Anfisa the monkey learns to type. She is yet unfamiliar with the “space” key and can only type in lower-case Latin letters. Having typed for a fairly long line, Anfisa understood that it would be great to divide what she has written into  $k$  lines not shorter than  $a$  and not longer than  $b$ , for the text to resemble human speech more. Help Anfisa.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int k, a, b;
8     cin >> k >> a >> b;
9     string s;
10    cin >> s;
11    int n = s.size();
12    if (n < a * k || n > b * k) {
13        cout << "No solution\n";
14        return 0;
15    }
16    for (int i = 0, j = 0; i < k; i++) {
17        int t = min(b, n - j - (k-i-1) * a);
18        cout << s.substr(j, t) << "\n";
19        j += t;
20    }
21    return 0;
22 }
```

## 201: Living Sequence

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In Japan, the number 4 reads like death, so Bob decided to build a live sequence. Living sequence  $a$  contains all natural numbers that do not contain the digit 4.  $a = [1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, \dots]$ .

For example, the number 1235 is part of the sequence  $a$ , but the numbers 4321, 443 are not part of the sequence  $a$ .

Bob realized that he does not know how to quickly search for a particular number by the position  $k$  in the sequence, so he asks for your help.

For example, if Bob wants to find the number at position  $k = 4$  (indexing from 1), you need to answer  $a_k = 5$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     i64 n;
6     cin >> n;
7     string s;
8     while (n) {
9         int x = n % 9;
10        s += '0' + (x < 4 ? x : x + 1);
11        n /= 9;
12    }
13    reverse(s.begin(), s.end());
14    cout << s << "\n";
15 }
16 int main() {
17     ios::sync_with_stdio(false);
18     cin.tie(nullptr);
19     int t;
20     cin >> t;
21     while (t--) {
22         solve();
23     }
24     return 0;
25 }
```

## 202: Hard problem

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vasiliy is fond of solving different tasks. Today he found one he wasn't able to solve himself, so he asks you to help.

Vasiliy is given  $n$  strings consisting of lowercase English letters. He wants them to be sorted in lexicographical order (as in the dictionary), but he is not allowed to swap any of them. The only operation he is allowed to do is to reverse any of them (first character becomes last, second becomes one before last and so on).

To reverse the  $i$ -th string Vasiliy has to spent  $c_i$  units of energy. He is interested in the minimum amount of energy he has to spent in order to have strings sorted in lexicographical order.

String A is lexicographically smaller than string B if it is shorter than B ( $|A| < |B|$ ) and is its prefix, or if none of them is a prefix of the other and at the first position where they differ character in A is smaller than the character in B.

For the purpose of this problem, two equal strings nearby do not break the condition of sequence being sorted lexicographically.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 1E18;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n;
9     cin >> n;
10    vector<int> c(n);
11    for (int i = 0; i < n; i++) {
12        cin >> c[i];
13    }
14    vector<array<string, 2>> s(n);
15    for (int i = 0; i < n; i++) {
16        cin >> s[i][0];
17        s[i][1] = string(s[i][0].rbegin(), s[i][0].rend());
18    }
19    array<i64, 2> dp{0, c[0]};
20    for (int i = 1; i < n; i++) {
21        array<i64, 2> g{inf, inf};
22        for (int x = 0; x < 2; x++) {
23            for (int y = 0; y < 2; y++) {
24                if (s[i - 1][x] <= s[i][y]) {
25                    g[y] = min(g[y], dp[x] + c[i] * y);
26                }
27            }
28        }
29        dp = g;
30    }
31    auto ans = min(dp[0], dp[1]);
32    if (ans == inf) {
33        ans = -1;
34    }
35    cout << ans << "\n";
36    return 0;
37 }
```

## 203: Serval and Toxel's Arrays

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Toxel likes arrays. Before traveling to the Paldea region, Serval gave him an array  $a$  as a gift. This array has  $n$  pairwise distinct elements.

In order to get more arrays, Toxel performed  $m$  operations with the initial array. In the  $i$ -th operation, he modified the  $p_i$ -th element of the  $(i - 1)$ -th array to  $v_i$ , resulting in the  $i$ -th array (the initial array  $a$

is numbered as 0). During modifications, Toxel guaranteed that the elements of each array are still pairwise distinct after each operation.

Finally, Toxel got  $m + 1$  arrays and denoted them as  $A_0 = a, A_1, \dots, A_m$ . For each pair  $(i, j)$  ( $0 \leq i < j \leq m$ ), Toxel defines its value as the number of distinct elements of the concatenation of  $A_i$  and  $A_j$ . Now Toxel wonders, what is the sum of the values of all pairs? Please help him to calculate the answer.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<int> f(n + m);
8     vector<int> a(n);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11         a[i]--;
12         f[a[i]] += m + 1;
13     }
14     for (int i = 0; i < m; i++) {
15         int p, v;
16         cin >> p >> v;
17         p--, v--;
18         f[a[p]] -= m - i;
19         a[p] = v;
20         f[a[p]] += m - i;
21     }
22     i64 ans = 1LL * m * (m + 1) * n;
23     for (int i = 0; i < n + m; i++) {
24         ans -= 1LL * f[i] * (f[i] - 1) / 2;
25     }
26     cout << ans << "\n";
27 }
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(nullptr);
31     int t;
32     cin >> t;
33     while (t--) {
34         solve();
35     }
36     return 0;
37 }
```

## 204: Ice and Fire

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

Little09 and his friends are playing a game. There are  $n$  players, and the temperature value of the player  $i$  is  $i$ .

The types of environment are expressed as 0 or 1. When two players fight in a specific environment, if its type is 0, the player with a lower temperature value in this environment always wins; if it is 1, the player with a higher temperature value in this environment always wins. The types of the  $n - 1$  environments form a binary string  $s$  with a length of  $n - 1$ .

If there are  $x$  players participating in the game, there will be a total of  $x - 1$  battles, and the types of the  $x - 1$  environments will be the first  $x - 1$  characters of  $s$ . While there is more than one player left in the tournament, choose any two remaining players to fight. The player who loses will be eliminated from the tournament. The type of the environment of battle  $i$  is  $s_i$ .

For each  $x$  from 2 to  $n$ , answer the following question: if all players whose temperature value does not exceed  $x$  participate in the game, how many players have a chance to win?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     string s;
8     cin >> s;
9     int suf = 0;
10    for (int x = 2; x <= n; x++) {
11        if (x > 2 && s[x - 2] != s[x - 3]) {
12            suf = 1;
13        } else {
14            suf++;
15        }
16        cout << x - suf << " \n"[x == n];
17    }
18 }
19 int main() {
20     ios::sync_with_stdio(false);
21     cin.tie(nullptr);
22     int t;
23     cin >> t;
24     while (t--) {
25         solve();
26     }
27     return 0;
28 }
```

## 205: Remove the Bracket

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

RSJ has a sequence  $a$  of  $n$  integers  $a_1, a_2, \dots, a_n$  and an integer  $s$ . For each of  $a_2, a_3, \dots, a_{n-1}$ , he chose a pair of non-negative integers  $x_i$  and  $y_i$  such that  $x_i + y_i = a_i$  and  $(x_i - s) \cdot (y_i - s) \geq 0$ .

Now he is interested in the value

$$F = a_1 \cdot x_2 + a_2 \cdot y_2 + a_3 \cdot x_3 + a_4 \cdot y_3 + \dots + a_{n-2} \cdot x_{n-1} + a_{n-1} \cdot y_{n-1}.$$

Please help him find the minimum possible value  $F$  he can get by choosing  $x_i$  and  $y_i$  optimally. It can be shown that there is always at least one valid way to choose them.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 1E18;
5 void solve() {
6     int n, s;
7     cin >> n >> s;
8     vector<int> a(n), x(n);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11         if (a[i] <= s) x[i] = 0;
12         else x[i] = s;
13     }
14     x[0] = 0, x[n - 1] = a[n - 1];
15     array<i64, 2> dp{inf, inf};
16     dp[0] = 0;
17     for (int i = 1; i < n; i++) {
18         array<i64, 2> g{inf, inf};
19         for (int j = 0; j < 2; j++) {
20             for (int k = 0; k < 2; k++) {
21                 g[k] = min(g[k], dp[j] + 1LL * (j ? x[i - 1] : a[i - 1] - x[i - 1]) * (k ?
22                     a[i] - x[i] : x[i]));
23             }
24         }
25         dp = g;
26     }
27     cout << dp[0] << "\n";
28 }
29 int main() {
30     ios::sync_with_stdio(false);
31     cin.tie(nullptr);
32     int t;
33     cin >> t;
34     while (t--) {

```

```

34     solve();
35 }
36 return 0;
37 }
```

## 206: Playoff

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

$2^n$  teams participate in a playoff tournament. The tournament consists of  $2^n - 1$  games. They are held as follows: in the first phase of the tournament, the teams are split into pairs: team 1 plays against team 2, team 3 plays against team 4, and so on (so,  $2^{n-1}$  games are played in that phase). When a team loses a game, it is eliminated, and each game results in elimination of one team (there are no ties). After that, only  $2^{n-1}$  teams remain. If only one team remains, it is declared the champion; otherwise, the second phase begins, where  $2^{n-2}$  games are played: in the first one of them, the winner of the game “1 vs 2” plays against the winner of the game “3 vs 4”, then the winner of the game “5 vs 6” plays against the winner of the game “7 vs 8”, and so on. This process repeats until only one team remains.

The skill level of the  $i$ -th team is  $p_i$ , where  $p$  is a permutation of integers  $1, 2, \dots, 2^n$  (a permutation is an array where each element from 1 to  $2^n$  occurs exactly once).

You are given a string  $s$  which consists of  $n$  characters. These characters denote the results of games in each phase of the tournament as follows:

if  $s_i$  is equal to 0, then during the  $i$ -th phase (the phase with  $2^{n-i}$  games), in each match, the team with the lower skill level wins;

if  $s_i$  is equal to 1, then during the  $i$ -th phase (the phase with  $2^{n-i}$  games), in each match, the team with the higher skill level wins.

Let's say that an integer  $x$  is winning if it is possible to find a permutation  $p$  such that the team with skill  $x$  wins the tournament. Find all winning integers.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
```

```

6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     string s;
10    cin >> s;
11    int lt = 0, gt = 0;
12    for (int i = 0; i < n; i++) {
13        if (s[i] == '1') lt += lt + 1;
14        else gt += gt + 1;
15    }
16    for (int i = lt + 1; i <= (1 << n) - gt; i++) {
17        cout << i << "\n"[i == n - gt];
18    }
19    return 0;
20 }
```

## 207: Hamiltonian Wall

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Sir Monocarp Hamilton is planning to paint his wall. The wall can be represented as a grid, consisting of 2 rows and  $m$  columns. Initially, the wall is completely white.

Monocarp wants to paint a black picture on the wall. In particular, he wants cell  $(i, j)$  (the  $j$ -th cell in the  $i$ -th row) to be colored black, if  $c_{i,j} = \text{'B}'$ , and to be left white, if  $c_{i,j} = \text{'W}'$ . Additionally, he wants each column to have at least one black cell, so, for each  $j$ , the following constraint is satisfied:  $c_{1,j}, c_{2,j}$  or both of them will be equal to 'B'.

In order for the picture to turn out smooth, Monocarp wants to place down a paint brush in some cell  $(x_1, y_1)$  and move it along the path  $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)$  so that:

for each  $i$ ,  $(x_i, y_i)$  and  $(x_{i+1}, y_{i+1})$  share a common side;

all black cells appear in the path exactly once;

white cells don't appear in the path.

Determine if Monocarp can paint the wall.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
```

```

4 void solve() {
5     int n;
6     cin >> n;
7     string s[2];
8     cin >> s[0] >> s[1];
9     for (int t = 0; t < 2; t++) {
10         int x = t;
11         int ok = 1;
12         for (int i = 0; i < n; i++) {
13             if (s[x][i] != 'B') ok = 0;
14             if (s[!x][i] == 'B') x ^= 1;
15         }
16         if (ok) {
17             cout << "YES\n";
18             return;
19         }
20     }
21     cout << "NO\n";
22 }
23 int main() {
24     ios::sync_with_stdio(false);
25     cin.tie(nullptr);
26     int t;
27     cin >> t;
28     while (t--) solve();
29     return 0;
30 }
```

## 208: Hossam and Friends

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Hossam makes a big party, and he will invite his friends to the party.

He has  $n$  friends numbered from 1 to  $n$ . They will be arranged in a queue as follows:  $1, 2, 3, \dots, n$ .

Hossam has a list of  $m$  pairs of his friends that don't know each other. Any pair not present in this list are friends.

A subsegment of the queue starting from the friend  $a$  and ending at the friend  $b$  is  $[a, a+1, a+2, \dots, b]$ . A subsegment of the queue is called good when all pairs of that segment are friends.

Hossam wants to know how many pairs  $(a, b)$  there are ( $1 \leq a \leq b \leq n$ ), such that the subsegment starting from the friend  $a$  and ending at the friend  $b$  is good.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
```

```

3  using i64 = long long;
4  void solve() {
5      int n, m;
6      cin >> n >> m;
7      vector<int> r(n, n - 1);
8      for (int i = 0; i < m; i++) {
9          int x, y;
10         cin >> x >> y;
11         x--, y--;
12         if (x > y) {
13             swap(x, y);
14         }
15         r[x] = min(r[x], y - 1);
16     }
17     i64 ans = 0;
18     for (int i = n - 1; i >= 0; i--) {
19         if (i < n - 1) {
20             r[i] = min(r[i], r[i + 1]);
21         }
22         ans += r[i] - i + 1;
23     }
24     cout << ans << "\n";
25 }
26 int main() {
27     ios::sync_with_stdio(false);
28     cin.tie(nullptr);
29     int t;
30     cin >> t;
31     while (t--) {
32         solve();
33     }
34     return 0;
35 }
```

## 209: The Humanoid

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  astronauts working on some space station. An astronaut with the number  $i$  ( $1 \leq i \leq n$ ) has power  $a_i$ .

An evil humanoid has made his way to this space station. The power of this humanoid is equal to  $h$ . Also, the humanoid took with him two green serums and one blue serum.

In one second , a humanoid can do any of three actions:

to absorb an astronaut with power strictly less humanoid power;

to use green serum, if there is still one left;

to use blue serum, if there is still one left.

When an astronaut with power  $a_i$  is absorbed, this astronaut disappears, and power of the humanoid increases by  $\lfloor \frac{a_i}{2} \rfloor$ , that is, an integer part of  $\frac{a_i}{2}$ . For example, if a humanoid absorbs an astronaut with power 4, its power increases by 2, and if a humanoid absorbs an astronaut with power 7, its power increases by 3.

After using the green serum, this serum disappears, and the power of the humanoid doubles, so it increases by 2 times.

After using the blue serum, this serum disappears, and the power of the humanoid triples, so it increases by 3 times.

The humanoid is wondering what the maximum number of astronauts he will be able to absorb if he acts optimally.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, h;
6     cin >> n >> h;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    sort(a.begin(), a.end());
12    array<array<i64, 2>, 3> dp {};
13    dp[2][1] = h;
14    int ans = 0;
15    for (int i = 0; i < n; i++) {
16        for (int x = 2; x >= 0; x--) {
17            for (int y = 1; y >= 0; y--) {
18                if (x) {
19                    dp[x - 1][y] = max(dp[x - 1][y], 2 * dp[x][y]);
20                }
21                if (y) {
22                    dp[x][y - 1] = max(dp[x][y - 1], 3 * dp[x][y]);
23                }
24                if (dp[x][y] > a[i]) {
25                    dp[x][y] += a[i] / 2;
26                    ans = i + 1;
27                }
28            }
29        }
30    }
31    cout << ans << "\n";
32 }
33 int main() {
34     ios::sync_with_stdio(false);
35     cin.tie(nullptr);
36     int t;
37     cin >> t;
38     while (t--) {

```

```

39         solve();
40     }
41     return 0;
42 }
```

## 210: Zero-Sum Prefixes

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The score of an array  $v_1, v_2, \dots, v_n$  is defined as the number of indices  $i$  ( $1 \leq i \leq n$ ) such that  $v_1 + v_2 + \dots + v_i = 0$ .

You are given an array  $a_1, a_2, \dots, a_n$  of length  $n$ . You can perform the following operation multiple times:

select an index  $i$  ( $1 \leq i \leq n$ ) such that  $a_i = 0$ ;

then replace  $a_i$  by an arbitrary integer.

What is the maximum possible score of  $a$  that can be obtained by performing a sequence of such operations?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    i64 s = 0;
12    int ans = 0;
13    int max = 0;
14    int first = 1;
15    map<i64, int> cnt;
16    for (int i = 0; i < n; i++) {
17        if (a[i] == 0) {
18            if (first) {
19                first = 0;
20                ans += cnt[0];
21            } else {
22                ans += max;
23            }
24            max = 0;
25            cnt.clear();
```

```

26         }
27         s += a[i];
28         max = max(max, ++cnt[s]);
29     }
30     if (first) {
31         first = 0;
32         ans += cnt[0];
33     } else {
34         ans += max;
35     }
36     cout << ans << "\n";
37 }
38 int main() {
39     ios::sync_with_stdio(false);
40     cin.tie(nullptr);
41     int t;
42     cin >> t;
43     while (t--) {
44         solve();
45     }
46     return 0;
47 }
```

**greedy****211: Romantic Glasses**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Iulia has  $n$  glasses arranged in a line. The  $i$ -th glass has  $a_i$  units of juice in it. Iulia drinks only from odd-numbered glasses, while her date drinks only from even-numbered glasses.

To impress her date, Iulia wants to find a contiguous subarray of these glasses such that both Iulia and her date will have the same amount of juice in total if only the glasses in this subarray are considered. Please help her to do that.

More formally, find out if there exists two indices  $l, r$  such that  $1 \leq l \leq r \leq n$ , and  $a_l + a_{l+2} + a_{l+4} + \dots + a_r = a_{l+1} + a_{l+3} + \dots + a_{r-1}$  if  $l$  and  $r$  have the same parity and  $a_l + a_{l+2} + a_{l+4} + \dots + a_{r-1} = a_{l+1} + a_{l+3} + \dots + a_r$  otherwise.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
```

```

6     cin >> n;
7     vector<i64> s(n + 1);
8     set<i64> set{0LL};
9     for (int i = 0; i < n; i++) {
10         int a;
11         cin >> a;
12         s[i + 1] = s[i] + (i % 2 ? 1 : -1) * a;
13         set.insert(s[i + 1]);
14     }
15     if (set.size() != n + 1) {
16         cout << "YES\n";
17     } else {
18         cout << "NO\n";
19     }
20 }
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     int t;
25     cin >> t;
26     while (t--) {
27         solve();
28     }
29     return 0;
30 }
```

## 212: Watering an Array

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have an array of integers  $a_1, a_2, \dots, a_n$  of length  $n$ . On the  $i$ -th of the next  $d$  days you are going to do exactly one of the following two actions:

Add 1 to each of the first  $b_i$  elements of the array  $a$  (i.e., set  $a_j := a_j + 1$  for each  $1 \leq j \leq b_i$ ).

Count the elements which are equal to their position (i.e., the  $a_j = j$ ). Denote the number of such elements as  $c$ . Then, you add  $c$  to your score, and reset the entire array  $a$  to a 0-array of length  $n$  (i.e., set  $[a_1, a_2, \dots, a_n] := [0, 0, \dots, 0]$ ).

Your score is equal to 0 in the beginning. Note that on each day you should perform exactly one of the actions above: you cannot skip a day or perform both actions on the same day.

What is the maximum score you can achieve at the end?

Since  $d$  can be quite large, the sequence  $b$  is given to you in the compressed format:

You are given a sequence of integers  $v_1, v_2, \dots, v_k$ . The sequence  $b$  is a concatenation of infinitely many copies of  $v$ :  $b = [v_1, v_2, \dots, v_k, v_1, v_2, \dots, v_k, \dots]$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k, d;
6     cin >> n >> k >> d;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<int> v(k);
12    for (int i = 0; i < k; i++) {
13        cin >> v[i];
14    }
15    int ans = 0;
16    for (int i = 0; i < d && i < 2 * n + 1; i++) {
17        int res = 0;
18        for (int j = 0; j < n; j++) {
19            res += (a[j] == j + 1);
20        }
21        ans = max(ans, res + (d - 1 - i) / 2);
22        int b = v[i % k];
23        for (int j = 0; j < b; j++) {
24            a[j] += 1;
25        }
26    }
27    cout << ans << "\n";
28 }
29 int main() {
30     ios::sync_with_stdio(false);
31     cin.tie(nullptr);
32     int t;
33     cin >> t;
34     while (t--) {
35         solve();
36     }
37     return 0;
38 }
```

### 213: Heavy Intervals

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have  $n$  intervals  $[l_1, r_1], [l_2, r_2], \dots, [l_n, r_n]$ , such that  $l_i < r_i$  for each  $i$ , and all the endpoints of the intervals are distinct.

The  $i$ -th interval has weight  $c_i$  per unit length. Therefore, the weight of the  $i$ -th interval is  $c_i \cdot (r_i - l_i)$ .

You don't like large weights, so you want to make the sum of weights of the intervals as small as possible. It turns out you can perform all the following three operations:

rearrange the elements in the array  $l$  in any order;

rearrange the elements in the array  $r$  in any order;

rearrange the elements in the array  $c$  in any order.

However, after performing all of the operations, the intervals must still be valid (i.e., for each  $i$ ,  $l_i < r_i$  must hold).

What's the minimum possible sum of weights of the intervals after performing the operations?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<pair<int, int>> a;
8     for (int i = 0; i < n; i++) {
9         int l;
10        cin >> l;
11        a.emplace_back(l, 1);
12    }
13    for (int i = 0; i < n; i++) {
14        int r;
15        cin >> r;
16        a.emplace_back(r, -1);
17    }
18    sort(a.begin(), a.end());
19    vector<int> c(n);
20    for (int i = 0; i < n; i++) {
21        cin >> c[i];
22    }
23    sort(c.begin(), c.end());
24    i64 ans = 0;
25    vector<int> len, stk;
26    for (auto [x, t] : a) {
27        if (t == 1) {
28            stk.push_back(x);
29        } else {
30            assert(!stk.empty());
31            len.push_back(x - stk.back());
32            stk.pop_back();
33        }
34    }
35    sort(len.begin(), len.end(), greater());
36    for (int i = 0; i < n; i++) {
37        ans += 1LL * c[i] * len[i];
38    }
39    cout << ans << "\n";
40 }
41 int main() {
42     ios::sync_with_stdio(false);
43     cin.tie(nullptr);
44     int t;
45     cin >> t;
46     while (t--) {
47         solve();

```

```

48     }
49     return 0;
50 }
```

## 214: Game with Marbles (Hard Version)

- Time limit: 3.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The easy and hard versions of this problem differ only in the constraints on the number of test cases and  $n$ . In the hard version, the number of test cases does not exceed  $10^4$ , and the sum of values of  $n$  over all test cases does not exceed  $2 \cdot 10^5$ . Furthermore, there are no additional constraints on  $n$  in a single test case.

Recently, Alice and Bob were given marbles of  $n$  different colors by their parents. Alice has received  $a_1$  marbles of color 1,  $a_2$  marbles of color 2, ...,  $a_n$  marbles of color  $n$ . Bob has received  $b_1$  marbles of color 1,  $b_2$  marbles of color 2, ...,  $b_n$  marbles of color  $n$ . All  $a_i$  and  $b_i$  are between 1 and  $10^9$ .

After some discussion, Alice and Bob came up with the following game: players take turns, starting with Alice. On their turn, a player chooses a color  $i$  such that both players have at least one marble of that color. The player then discards one marble of color  $i$ , and their opponent discards all marbles of color  $i$ . The game ends when there is no color  $i$  such that both players have at least one marble of that color.

The score in the game is the difference between the number of remaining marbles that Alice has and the number of remaining marbles that Bob has at the end of the game. In other words, the score in the game is equal to  $(A - B)$ , where  $A$  is the number of marbles Alice has and  $B$  is the number of marbles Bob has at the end of the game. Alice wants to maximize the score, while Bob wants to minimize it.

Calculate the score at the end of the game if both players play optimally.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n), b(n);
8     for (int i = 0; i < n; i++) {
```

```

9      cin >> a[i];
10     }
11     for (int i = 0; i < n; i++) {
12         cin >> b[i];
13     }
14     vector<int> p(n);
15     iota(p.begin(), p.end(), 0);
16     sort(p.begin(), p.end(),
17           [&](int i, int j) {
18               return a[i] + b[i] > a[j] + b[j];
19           });
20     i64 ans = 0;
21     for (int i = 0; i < n; i++) {
22         int j = p[i];
23         if (i % 2 == 0) {
24             ans += a[j] - 1;
25         } else {
26             ans -= b[j] - 1;
27         }
28     }
29     cout << ans << "\n";
30 }
31 int main() {
32     ios::sync_with_stdio(false);
33     cin.tie(nullptr);
34     int t;
35     cin >> t;
36     while (t--) {
37         solve();
38     }
39     return 0;
40 }
```

## 215: Game with Marbles (Easy Version)

- Time limit: 3.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The easy and hard versions of this problem differ only in the constraints on the number of test cases and  $n$ . In the easy version, the number of test cases does not exceed  $10^3$ , and  $n$  does not exceed 6.

Recently, Alice and Bob were given marbles of  $n$  different colors by their parents. Alice has received  $a_1$  marbles of color 1,  $a_2$  marbles of color 2, ...,  $a_n$  marbles of color  $n$ . Bob has received  $b_1$  marbles of color 1,  $b_2$  marbles of color 2, ...,  $b_n$  marbles of color  $n$ . All  $a_i$  and  $b_i$  are between 1 and  $10^9$ .

After some discussion, Alice and Bob came up with the following game: players take turns, starting with Alice. On their turn, a player chooses a color  $i$  such that both players have at least one marble of that color. The player then discards one marble of color  $i$ , and their opponent discards all marbles of color  $i$ . The game ends when there is no color  $i$  such that both players have at least one marble of that color.

The score in the game is the difference between the number of remaining marbles that Alice has and the number of remaining marbles that Bob has at the end of the game. In other words, the score in the game is equal to  $(A - B)$ , where  $A$  is the number of marbles Alice has and  $B$  is the number of marbles Bob has at the end of the game. Alice wants to maximize the score, while Bob wants to minimize it.

Calculate the score at the end of the game if both players play optimally.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n), b(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    for (int i = 0; i < n; i++) {
12        cin >> b[i];
13    }
14    vector<int> p(n);
15    iota(p.begin(), p.end(), 0);
16    sort(p.begin(), p.end(),
17          [&](int i, int j) {
18              return a[i] + b[i] > a[j] + b[j];
19          });
20    i64 ans = 0;
21    for (int i = 0; i < n; i++) {
22        int j = p[i];
23        if (i % 2 == 0) {
24            ans += a[j] - 1;
25        } else {
26            ans -= b[j] - 1;
27        }
28    }
29    cout << ans << "\n";
30 }
31 int main() {
32     ios::sync_with_stdio(false);
33     cin.tie(nullptr);
34     int t;
35     cin >> t;
36     while (t--) {
37         solve();
38     }
39     return 0;
40 }
```

## 216: Game with Multiset

- Time limit: 1.5 seconds

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In this problem, you are initially given an empty multiset. You have to process two types of queries:

ADD  $x$  - add an element equal to  $2^x$  to the multiset;

GET  $w$  - say whether it is possible to take the sum of some subset of the current multiset and get a value equal to  $w$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int m;
8     cin >> m;
9     array<int, 30> cnt{};
10    while (m--) {
11        int t, v;
12        cin >> t >> v;
13        if (t == 1) {
14            cnt[v] += 1;
15        } else {
16            for (int i = 29; i >= 0; i--) {
17                int x = min(v >> i, cnt[i]);
18                v -= x << i;
19            }
20            if (v == 0) {
21                cout << "YES\n";
22            } else {
23                cout << "NO\n";
24            }
25        }
26    }
27    return 0;
28 }
```

## 217: Largest Subsequence

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Given is a string  $s$  of length  $n$ . In one operation you can select the lexicographically largest<sup>†</sup> subsequence of string  $s$  and cyclic shift it to the right<sup>‡</sup>.

Your task is to calculate the minimum number of operations it would take for  $s$  to become sorted, or report that it never reaches a sorted state.

<sup>†</sup>A string  $a$  is lexicographically smaller than a string  $b$  if and only if one of the following holds:

$a$  is a prefix of  $b$ , but  $a \neq b$ ;

In the first position where  $a$  and  $b$  differ, the string  $a$  has a letter that appears earlier in the alphabet than the corresponding letter in  $b$ .

<sup>‡</sup>By cyclic shifting the string  $t_1 t_2 \dots t_m$  to the right, we get the string  $t_m t_1 \dots t_{m-1}$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     string s;
8     cin >> s;
9     vector<int> a;
10    for (int i = n - 1; i >= 0; i--) {
11        if (a.empty() || s[i] >= s[a.back()]) {
12            a.push_back(i);
13        }
14    }
15    reverse(a.begin(), a.end());
16    int j = 0;
17    while (j < a.size() && s[a[0]] == s[a[j]]) {
18        j += 1;
19    }
20    int ans = a.size() - j;
21    for (int i = 0; i < a.size() - 1 - j; i++) {
22        swap(s[a[i]], s[a[a.size() - 1 - i]]);
23    }
24    if (!is_sorted(s.begin(), s.end())) {
25        ans = -1;
26    }
27    cout << ans << "\n";
28 }
29 int main() {
30     ios::sync_with_stdio(false);
31     cin.tie(nullptr);
32     int t;
33     cin >> t;
34     while (t--) {
35         solve();
36     }
37     return 0;
38 }
```

## 218: Add, Divide and Floor

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an integer array  $a_1, a_2, \dots, a_n$  ( $0 \leq a_i \leq 10^9$ ). In one operation, you can choose an integer  $x$  ( $0 \leq x \leq 10^{18}$ ) and replace  $a_i$  with  $\lfloor \frac{a_i+x}{2} \rfloor$  ( $\lfloor y \rfloor$  denotes rounding  $y$  down to the nearest integer) for all  $i$  from 1 to  $n$ . Pay attention to the fact that all elements of the array are affected on each operation.

Print the smallest number of operations required to make all elements of the array equal.

If the number of operations is less than or equal to  $n$ , then print the chosen  $x$  for each operation. If there are multiple answers, print any of them.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    int min = *min_element(a.begin(), a.end());
12    int max = *max_element(a.begin(), a.end());
13    vector<int> ans;
14    while (min < max) {
15        int x = min % 2;
16        ans.push_back(x);
17        min = (min + x) / 2;
18        max = (max + x) / 2;
19    }
20    cout << ans.size() << "\n";
21    if (ans.size() <= n) {
22        for (auto x : ans) {
23            cout << x << " ";
24        }
25        cout << "\n";
26    }
27 }
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(nullptr);
31     int t;
32     cin >> t;
33     while (t--) {

```

```

34     solve();
35 }
36 return 0;
37 }
```

## 219: Theofanis' Nightmare

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Theofanis easily gets obsessed with problems before going to sleep and often has nightmares about them. To deal with his obsession he visited his doctor, Dr. Emix.

In his latest nightmare, he has an array  $a$  of size  $n$  and wants to divide it into non-empty subarrays<sup>†</sup> such that every element is in exactly one of the subarrays.

For example, the array  $[1, -3, 7, -6, 2, 5]$  can be divided to  $[1][-3, 7][-6, 2][5]$ .

The Cypriot value of such division is equal to  $\sum_{i=1}^k i \cdot \text{sum}_i$  where  $k$  is the number of subarrays that we divided the array into and  $\text{sum}_i$  is the sum of the  $i$ -th subarray.

The Cypriot value of this division of the array  $[1][-3, 7][-6, 2][5] = 1 \cdot 1 + 2 \cdot (-3+7) + 3 \cdot (-6+2) + 4 \cdot 5 = 17$ .

Theofanis is wondering what is the maximum Cypriot value of any division of the array.

<sup>†</sup> An array  $b$  is a subarray of an array  $a$  if  $b$  can be obtained from  $a$  by deletion of several (possibly, zero or all) elements from the beginning and several (possibly, zero or all) elements from the end. In particular, an array is a subarray of itself.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<i64> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    for (int i = n - 2; i >= 0; i--) {
12        a[i] += a[i + 1];
13    }
14    i64 ans = a[0];
15    for (int i = 1; i < n; i++) {
```

```

16         ans += max(0LL, a[i]);
17     }
18     cout << ans << "\n";
19 }
20 int main() {
21     ios::sync_with_stdio(false);
22     cin.tie(nullptr);
23     int t;
24     cin >> t;
25     while (t--) {
26         solve();
27     }
28     return 0;
29 }
```

## 220: Insert and Equalize

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an integer array  $a_1, a_2, \dots, a_n$ , all its elements are distinct.

First, you are asked to insert one more integer  $a_{n+1}$  into this array.  $a_{n+1}$  should not be equal to any of  $a_1, a_2, \dots, a_n$ .

Then, you will have to make all elements of the array equal. At the start, you choose a positive integer  $x$  ( $x > 0$ ). In one operation, you add  $x$  to exactly one element of the array. Note that  $x$  is the same for all operations.

What's the smallest number of operations it can take you to make all elements equal, after you choose  $a_{n+1}$  and  $x$ ?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    if (n == 1) {
12        cout << 1 << "\n";
13    }
14 }
```

```

14     }
15     int g = 0;
16     for (int i = 1; i < n; i++) {
17         g = gcd(g, a[i] - a[i - 1]);
18     }
19     int max = *max_element(a.begin(), a.end());
20     set<int> s;
21     vector<int> cnt(n + 1);
22     i64 ans = 0;
23     int mex = 0;
24     for (int i = 0; i < n; i++) {
25         a[i] = (max - a[i]) / g;
26         if (a[i] <= n) {
27             cnt[a[i]] += 1;
28         }
29         ans += a[i];
30     }
31     while (cnt[mex]) {
32         mex++;
33     }
34     ans += mex;
35     cout << ans << "\n";
36 }
37 int main() {
38     ios::sync_with_stdio(false);
39     cin.tie(nullptr);
40     int t;
41     cin >> t;
42     while (t--) {
43         solve();
44     }
45     return 0;
46 }
```

## 221: Alex's whims

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Tree is a connected graph without cycles. It can be shown that any tree of  $n$  vertices has exactly  $n - 1$  edges.

Leaf is a vertex in the tree with exactly one edge connected to it.

Distance between two vertices  $u$  and  $v$  in a tree is the minimum number of edges that must be passed to come from vertex  $u$  to vertex  $v$ .

Alex's birthday is coming up, and Timofey would like to gift him a tree of  $n$  vertices. However, Alex is a very moody boy. Every day for  $q$  days, he will choose an integer, denoted by the integer chosen on the  $i$ -th day by  $d_i$ . If on the  $i$ -th day there are not two leaves in the tree at a distance exactly  $d_i$ , Alex will be disappointed.

Timofey decides to gift Alex a designer so that he can change his tree as he wants. Timofey knows that Alex is also lazy (a disaster, not a human being), so at the beginning of every day, he can perform no more than one operation of the following kind:

Choose vertices  $u$ ,  $v_1$ , and  $v_2$  such that there is an edge between  $u$  and  $v_1$  and no edge between  $u$  and  $v_2$ . Then remove the edge between  $u$  and  $v_1$  and add an edge between  $u$  and  $v_2$ . This operation cannot be performed if the graph is no longer a tree after it.

Somehow Timofey managed to find out all the  $d_i$ . After that, he had another brilliant idea - just in case, make an instruction manual for the set, one that Alex wouldn't be disappointed.

Timofey is not as lazy as Alex, but when he saw the integer  $n$ , he quickly lost the desire to develop the instruction and the original tree, so he assigned this task to you. It can be shown that a tree and a sequence of operations satisfying the described conditions always exist.

Here is an example of an operation where vertices were selected:  $u - 6$ ,  $v_1 - 1$ ,  $v_2 - 4$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, q;
6     cin >> n >> q;
7     int u = n;
8     int v1 = n - 1;
9     for (int i = 1; i < n; i++) {
10         cout << i << " " << i + 1 << "\n";
11     }
12     while (q--) {
13         int d;
14         cin >> d;
15         int v2 = d;
16         if (v1 == v2) {
17             cout << "-1 -1 -1\n";
18         } else {
19             cout << u << " " << v1 << " " << v2 << "\n";
20             v1 = v2;
21         }
22     }
23 }
24 int main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
27     int t;
28     cin >> t;
29     while (t--) {
30         solve();
31     }
32     return 0;
33 }
```

## 222: Queue Sort

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vlad found an array  $a$  of  $n$  integers and decided to sort it in non-decreasing order.

To do this, Vlad can apply the following operation any number of times:

Extract the first element of the array and insert it at the end;

Swap that element with the previous one until it becomes the first or until it becomes strictly greater than the previous one.

Note that both actions are part of the operation, and for one operation, you must apply both actions.

For example, if you apply the operation to the array [4, 3, 1, 2, 6, 4], it will change as follows:

[4, 3, 1, 2, 6, 4];

[3, 1, 2, 6, 4, 4];

[3, 1, 2, 6, 4, 4];

[3, 1, 2, 4, 6, 4].

Vlad doesn't have time to perform all the operations, so he asks you to determine the minimum number of operations required to sort the array or report that it is impossible.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    auto it = min_element(a.begin(), a.end());
12    if (is_sorted(it, a.end())) {
13        cout << it - a.begin() << "\n";
14    } else {
15        cout << -1 << "\n";
16    }
17 }
18 int main() {

```

```

19     ios::sync_with_stdio(false);
20     cin.tie(nullptr);
21     int t;
22     cin >> t;
23     while (t--) {
24         solve();
25     }
26     return 0;
27 }
```

## 223: Milena and Admirer

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Milena has received an array of integers  $a_1, a_2, \dots, a_n$  of length  $n$  from a secret admirer. She thinks that making it non-decreasing should help her identify the secret admirer.

She can use the following operation to make this array non-decreasing:

Select an element  $a_i$  of array  $a$  and an integer  $x$  such that  $1 \leq x < a_i$ . Then, replace  $a_i$  by two elements  $x$  and  $a_i - x$  in array  $a$ . New elements ( $x$  and  $a_i - x$ ) are placed in the array  $a$  in this order instead of  $a_i$ . More formally, let  $a_1, a_2, \dots, a_i, \dots, a_k$  be an array  $a$  before the operation. After the operation, it becomes equal to  $a_1, a_2, \dots, a_{i-1}, x, a_i - x, a_{i+1}, \dots, a_k$ . Note that the length of  $a$  increases by 1 on each operation.

More formally, let  $a_1, a_2, \dots, a_i, \dots, a_k$  be an array  $a$  before the operation. After the operation, it becomes equal to  $a_1, a_2, \dots, a_{i-1}, x, a_i - x, a_{i+1}, \dots, a_k$ . Note that the length of  $a$  increases by 1 on each operation.

Milena can perform this operation multiple times (possibly zero). She wants you to determine the minimum number of times she should perform this operation to make array  $a$  non-decreasing.

An array  $x_1, x_2, \dots, x_k$  of length  $k$  is called non-decreasing if  $x_i \leq x_{i+1}$  for all  $1 \leq i < k$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
```

```

10      }
11      int lst = a[n - 1];
12      i64 ans = 0;
13      for (int i = n - 2; i >= 0; i--) {
14          int t = (a[i] - 1) / lst + 1;
15          ans += t - 1;
16          lst = a[i] / t;
17      }
18      cout << ans << "\n";
19  }
20 int main() {
21     ios::sync_with_stdio(false);
22     cin.tie(nullptr);
23     int t;
24     cin >> t;
25     while (t--) {
26         solve();
27     }
28     return 0;
29 }
```

## 224: Smilo and Monsters

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A boy called Smilo is playing a new game! In the game, there are  $n$  hordes of monsters, and the  $i$ -th horde contains  $a_i$  monsters. The goal of the game is to destroy all the monsters. To do this, you have two types of attacks and a combo counter  $x$ , initially set to 0:

The first type: you choose a number  $i$  from 1 to  $n$ , such that there is at least one monster left in the horde with the number  $i$ . Then, you kill one monster from horde number  $i$ , and the combo counter  $x$  increases by 1.

The second type: you choose a number  $i$  from 1 to  $n$ , such that there are at least  $x$  monsters left in the horde with number  $i$ . Then, you use an ultimate attack and kill  $x$  monsters from the horde with number  $i$ . After that,  $x$  is reset to zero.

Your task is to destroy all of the monsters, meaning that there should be no monsters left in any of the hordes. Smilo wants to win as quickly as possible, so he wants to the minimum number of attacks required to win the game.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
```

```

3  using i64 = long long;
4  void solve() {
5      int n;
6      cin >> n;
7      vector<int> a(n);
8      for (int i = 0; i < n; i++) {
9          cin >> a[i];
10     }
11     sort(a.begin(), a.end());
12     i64 sum = accumulate(a.begin(), a.end(), 0LL);
13     i64 s = sum / 2;
14     i64 ans = sum - s;
15     for (int i = n - 1; i >= 0; i--) {
16         if (s > 0) {
17             s -= a[i];
18             ans += 1;
19         }
20     }
21     cout << ans << "\n";
22 }
23 int main() {
24     ios::sync_with_stdio(false);
25     cin.tie(nullptr);
26     int t;
27     cin >> t;
28     while (t--) {
29         solve();
30     }
31     return 0;
32 }
```

## 225: Dances (Easy version)

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the easy version of the problem. The only difference is that in this version  $m = 1$ .

You are given two arrays of integers  $a_1, a_2, \dots, a_n$  and  $b_1, b_2, \dots, b_n$ . Before applying any operations, you can reorder the elements of each array as you wish. Then, in one operation, you will perform both of the following actions, if the arrays are not empty:

Choose any element from array  $a$  and remove it (all remaining elements are shifted to a new array  $a$ ),

Choose any element from array  $b$  and remove it (all remaining elements are shifted to a new array  $b$ ).

Let  $k$  be the final size of both arrays. You need to find the minimum number of operations required to satisfy  $a_i < b_i$  for all  $1 \leq i \leq k$ .

This problem was too easy, so the problem author decided to make it more challenging. You are also given a positive integer  $m$ . Now, you need to find the sum of answers to the problem for  $m$  pairs of arrays  $(c[i], b)$ , where  $1 \leq i \leq m$ . Array  $c[i]$  is obtained from  $a$  as follows:

$c[i]_1 = i$ ,  
 $c[i]_j = a_j$ , for  $2 \leq j \leq n$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<int> a(n), b(n);
8     for (int i = 1; i < n; i++) {
9         cin >> a[i];
10    }
11    for (int i = 0; i < n; i++) {
12        cin >> b[i];
13    }
14    a[0] = 1;
15    sort(a.begin(), a.end());
16    sort(b.begin(), b.end());
17    int ans = 0;
18    for (int i = 0, j = 0; i < n; i++) {
19        while (j < n && a[i] >= b[j]) {
20            j++;
21        }
22        ans = max(ans, j - i);
23    }
24    cout << ans << "\n";
25 }
26 int main() {
27     ios::sync_with_stdio(false);
28     cin.tie(nullptr);
29     int t;
30     cin >> t;
31     while (t--) {
32         solve();
33     }
34     return 0;
35 }
```

## 226: Iva & Pav

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Iva and Pav are a famous Serbian competitive programming couple. In Serbia, they call Pav “papuca” and that’s why he will make all of Iva’s wishes come true.

Iva gave Pav an array  $a$  of  $n$  elements.

Let's define  $f(l, r) = a_l \& a_{l+1} \& \dots \& a_r$  (here  $\&$  denotes the bitwise AND operation).

Note that  $f(l, r)$  is not defined when  $l > r$ .

Iva also gave Pav  $q$  queries.

Each query consists of 2 numbers,  $k$  and  $l$ , and she wants Pav to find the largest index  $r$  ( $l \leq r \leq n$ ), such that  $f(l, r) \geq k$ .

Pav wants to solve this problem fast because he doesn't want to upset Iva. He needs your help.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<vector<int>> nxt(n + 1, vector<int>(30, -1));
12    for (int i = n - 1; i >= 0; i--) {
13        nxt[i] = nxt[i + 1];
14        for (int j = 0; j < 30; j++) {
15            if (~a[i] >> j & 1) {
16                nxt[i][j] = i;
17            }
18        }
19    }
20    int q;
21    cin >> q;
22    while (q--) {
23        int l, k;
24        cin >> l >> k;
25        l--;
26        int ans = l;
27        int res = n;
28        for (int i = 29; i >= 0; i--) {
29            if (k >> i & 1) {
30                res = min(res, nxt[l][i]);
31            } else {
32                ans = max(ans, min(res, nxt[l][i]));
33            }
34        }
35        ans = max(ans, res);
36        if (ans <= l) {
37            ans = -1;
38        }
39        cout << ans << " ";
40    }
41 }
42 int main() {
43     ios::sync_with_stdio(false);
44     cin.tie(nullptr);

```

```

45     int t;
46     cin >> t;
47     while (t--) {
48         solve();
49     }
50     return 0;
51 }
```

## 227: Reverse Madness

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a string  $s$  of length  $n$ , containing lowercase Latin letters.

Next you will be given a positive integer  $k$  and two arrays,  $l$  and  $r$  of length  $k$ .

It is guaranteed that the following conditions hold for these 2 arrays:

$$l_1 = 1;$$

$$r_k = n;$$

$l_i \leq r_i$ , for each positive integer  $i$  such that  $1 \leq i \leq k$ ;

$l_i = r_{i-1} + 1$ , for each positive integer  $i$  such that  $2 \leq i \leq k$ ;

Now you will be given a positive integer  $q$  which represents the number of modifications you need to do on  $s$ .

Each modification is defined with one positive integer  $x$ :

Find an index  $i$  such that  $l_i \leq x \leq r_i$  (notice that such  $i$  is unique).

Let  $a = \min(x, r_i + l_i - x)$  and let  $b = \max(x, r_i + l_i - x)$ .

Reverse the substring of  $s$  from index  $a$  to index  $b$ .

Reversing the substring  $[a, b]$  of a string  $s$  means to make  $s$  equal to  $s_1, s_2, \dots, s_{a-1}, s_b, s_{b-1}, \dots, s_{a+1}, s_a, s_{b+1}, s_{b+2}, \dots$

Print  $s$  after the last modification is finished.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
```

```

5     int n, k;
6     cin >> n >> k;
7     string s;
8     cin >> s;
9     vector<int> l(k), r(k);
10    for (int i = 0; i < k; i++) {
11        cin >> l[i];
12        l[i]--;
13    }
14    for (int i = 0; i < k; i++) {
15        cin >> r[i];
16        r[i]--;
17    }
18    int q;
19    cin >> q;
20    vector<int> f(n);
21    while (q--) {
22        int x;
23        cin >> x;
24        x--;
25        f[x] ^= 1;
26    }
27    for (int i = 0; i < k; i++) {
28        int rev = 0;
29        for (int j = l[i]; j <= l[i] + r[i] - j; j++) {
30            rev ^= f[j] ^ f[l[i] + r[i] - j];
31            if (rev) {
32                swap(s[j], s[l[i] + r[i] - j]);
33            }
34        }
35    }
36    cout << s << "\n";
37 }
38 int main() {
39     ios::sync_with_stdio(false);
40     cin.tie(nullptr);
41     int t;
42     cin >> t;
43     while (t--) {
44         solve();
45     }
46     return 0;
47 }
```

## 228: Card Game

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  cards stacked in a deck. Initially,  $a_i$  is written on the  $i$ -th card from the top. The value written on a card does not change.

You will play a game. Initially your score is 0. In each turn, you can do one of the following operations:

Choose an odd<sup>†</sup> positive integer  $i$ , which is not greater than the number of cards left in the deck.

Remove the  $i$ -th card from the top of the deck and add the number written on the card to your score. The remaining cards will be reindexed starting from the top.

Choose an even<sup>‡</sup> positive integer  $i$ , which is not greater than the number of cards left in the deck. Remove the  $i$ -th card from the top of the deck. The remaining cards will be reindexed starting from the top.

End the game. You can end the game whenever you want, you do not have to remove all cards from the initial deck.

What is the maximum score you can get when the game ends?

<sup>†</sup> An integer  $i$  is odd, if there exists an integer  $k$  such that  $i = 2k + 1$ .

<sup>‡</sup> An integer  $i$  is even, if there exists an integer  $k$  such that  $i = 2k$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    i64 ans = 0;
12    for (int i = 2; i < n; i++) {
13        ans += max(0, a[i]);
14    }
15    ans += max(0, a[0] + max(0, n > 1 ? a[1] : 0));
16    cout << ans << "\n";
17 }
18 int main() {
19     ios::sync_with_stdio(false);
20     cin.tie(nullptr);
21     int t;
22     cin >> t;
23     while (t--) {
24         solve();
25     }
26     return 0;
27 }
```

## 229: Sets and Union

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

You have  $n$  sets of integers  $S_1, S_2, \dots, S_n$ . We call a set  $S$  attainable, if it is possible to choose some (possibly, none) of the sets  $S_1, S_2, \dots, S_n$  so that  $S$  is equal to their union<sup>†</sup>. If you choose none of  $S_1, S_2, \dots, S_n$ , their union is an empty set.

Find the maximum number of elements in an attainable  $S$  such that  $S \neq S_1 \cup S_2 \cup \dots \cup S_n$ .

<sup>†</sup> The union of sets  $A_1, A_2, \dots, A_k$  is defined as the set of elements present in at least one of these sets. It is denoted by  $A_1 \cup A_2 \cup \dots \cup A_k$ . For example,  $\{2, 4, 6\} \cup \{2, 3\} \cup \{3, 6, 7\} = \{2, 3, 4, 6, 7\}$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     i64 Or = 0;
8     vector<i64> a(n);
9     int ans = 0;
10    for (int i = 0; i < n; i++) {
11        int k;
12        cin >> k;
13        while (k--) {
14            int x;
15            cin >> x;
16            a[i] |= 1LL << x;
17        }
18        Or |= a[i];
19    }
20    for (int i = 1; i <= 50; i++) {
21        if (Or >> i & 1) {
22            i64 v = 0;
23            for (int j = 0; j < n; j++) {
24                if (~a[j] >> i & 1) {
25                    v |= a[j];
26                }
27            }
28            ans = max(ans, __builtin_popcountll(v));
29        }
30    }
31    cout << ans << "\n";
32 }
33 int main() {
34     ios::sync_with_stdio(false);
35     cin.tie(nullptr);
36     int t;
37     cin >> t;
38     while (t--) {
39         solve();
40     }
41     return 0;
42 }
```

## 230: Make it Alternating

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a binary string  $s$ . A binary string is a string consisting of characters 0 and/or 1.

You can perform the following operation on  $s$  any number of times (even zero):

choose an integer  $i$  such that  $1 \leq i \leq |s|$ , then erase the character  $s_i$ .

You have to make  $s$  alternating, i. e. after you perform the operations, every two adjacent characters in  $s$  should be different.

Your goal is to calculate two values:

the minimum number of operations required to make  $s$  alternating;

the number of different shortest sequences of operations that make  $s$  alternating. Two sequences of operations are different if in at least one operation, the chosen integer  $i$  is different in these two sequences.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;

```

```

26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 void solve() {
35     string s;
36     cin >> s;
37     int n = s.size();
38     Z ways = 1;
39     int ans = 0;
40     for (int l = 0, r = 0; l < n; l = r) {
41         while (r < n && s[l] == s[r]) {
42             r++;
43         }
44         ans += r - l - 1;
45         ways *= r - l;
46     }
47     for (int i = 1; i <= ans; i++) {
48         ways *= i;
49     }
50     cout << ans << " " << ways << "\n";
51 }
52 int main() {
53     ios::sync_with_stdio(false);
54     cin.tie(nullptr);
55     int t;
56     cin >> t;
57     while (t--) {
58         solve();
59     }
60     return 0;
61 }
```

## implementation

### 231: Anonymous Informant

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $b_1, b_2, \dots, b_n$ .

An anonymous informant has told you that the array  $b$  was obtained as follows: initially, there existed an array  $a_1, a_2, \dots, a_n$ , after which the following two-component operation was performed  $k$  times:

A fixed point<sup>†</sup>  $x$  of the array  $a$  was chosen.

Then, the array  $a$  was cyclically shifted to the left<sup>‡</sup> exactly  $x$  times.

As a result of  $k$  such operations, the array  $b_1, b_2, \dots, b_n$  was obtained. You want to check if the words of the anonymous informant can be true or if they are guaranteed to be false.

<sup>†</sup>A number  $x$  is called a fixed point of the array  $a_1, a_2, \dots, a_n$  if  $1 \leq x \leq n$  and  $a_x = x$ .

<sup>‡</sup>A cyclic left shift of the array  $a_1, a_2, \dots, a_n$  is the array  $a_2, \dots, a_n, a_1$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    int x = n - 1;
12    for (int i = 0; i < min(n, k); i++) {
13        if (a[x] > n) {
14            cout << "No\n";
15            return;
16        }
17        x = (x - a[x] + n) % n;
18    }
19    cout << "Yes\n";
20 }
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     int t;
25     cin >> t;
26     while (t--) {
27         solve();
28     }
29     return 0;
30 }
```

## 232: Decreasing String

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Recall that string  $a$  is lexicographically smaller than string  $b$  if  $a$  is a prefix of  $b$  (and  $a \neq b$ ), or there exists an index  $i$  ( $1 \leq i \leq \min(|a|, |b|)$ ) such that  $a_i < b_i$ , and for any index  $j$  ( $1 \leq j < i$ )  $a_j = b_j$ .

Consider a sequence of strings  $s_1, s_2, \dots, s_n$ , each consisting of lowercase Latin letters. String  $s_1$  is given explicitly, and all other strings are generated according to the following rule: to obtain the string

$s_i$ , a character is removed from string  $s_{i-1}$  in such a way that string  $s_i$  is lexicographically minimal.

For example, if  $s_1 = dacb$ , then string  $s_2 = acb$ , string  $s_3 = ab$ , string  $s_4 = a$ .

After that, we obtain the string  $S = s_1 + s_2 + \dots + s_n$  ( $S$  is the concatenation of all strings  $s_1, s_2, \dots, s_n$ ).

You need to output the character in position  $pos$  of the string  $S$  (i. e. the character  $S_{pos}$ ).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     string s;
6     cin >> s;
7     int n = s.size();
8     i64 pos;
9     cin >> pos;
10    pos--;
11    int x, y;
12    for (int i = 0; i < n; i++) {
13        int len = n - i;
14        if (pos < len) {
15            x = i;
16            y = pos;
17            break;
18        }
19        pos -= len;
20    }
21    string t;
22    for (auto c : s) {
23        while (x > 0 && !t.empty() && c < t.back()) {
24            t.pop_back();
25            x--;
26        }
27        t += c;
28    }
29    cout << t[y];
30 }
31 int main() {
32     ios::sync_with_stdio(false);
33     cin.tie(nullptr);
34     cout << fixed << setprecision(10);
35     int t;
36     cin >> t;
37     while (t--) {
38         solve();
39     }
40     cout << "\n";
41     return 0;
42 }
```

## 233: Fill in the Matrix

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There is an empty matrix  $M$  of size  $n \times m$ .

Zhongkao examination is over, and Daniel would like to do some puzzle games. He is going to fill in the matrix  $M$  using permutations of length  $m$ . That is, each row of  $M$  must be a permutation of length  $m$ <sup>†</sup>.

Define the value of the  $i$ -th column in  $M$  as  $v_i = \text{MEX}(M_{1,i}, M_{2,i}, \dots, M_{n,i})$ <sup>‡</sup>. Since Daniel likes diversity, the beauty of  $M$  is  $s = \text{MEX}(v_1, v_2, \dots, v_m)$ .

You have to help Daniel fill in the matrix  $M$  and maximize its beauty.

<sup>†</sup> A permutation of length  $m$  is an array consisting of  $m$  distinct integers from 0 to  $m - 1$  in arbitrary order. For example,  $[1, 2, 0, 4, 3]$  is a permutation, but  $[0, 1, 1]$  is not a permutation (1 appears twice in the array), and  $[0, 1, 3]$  is also not a permutation ( $m - 1 = 2$  but there is 3 in the array).

<sup>‡</sup> The MEX of an array is the smallest non-negative integer that does not belong to the array. For example,  $\text{MEX}(2, 2, 1) = 0$  because 0 does not belong to the array, and  $\text{MEX}(0, 3, 1, 2) = 4$  because 0, 1, 2 and 3 appear in the array, but 4 does not.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     cout << (m == 1 ? 0 : m == 2 ? 2 : min(n + 1, m)) << "\n";
8     for (int i = 0; i < n; i++) {
9         vector<int> a(m);
10        iota(a.begin(), a.end(), 0);
11        int p = max(1, m - i - 1);
12        reverse(a.begin(), a.begin() + p);
13        reverse(a.begin() + p, a.end());
14        for (int j = 0; j < m; j++) {
15            cout << a[j] << " \n"[j == m - 1];
16        }
17    }
18 }
19 int main() {
20     ios::sync_with_stdio(false);
21     cin.tie(nullptr);
22     int t;
23     cin >> t;

```

```

24     while (t--) {
25         solve();
26     }
27     return 0;
28 }
```

## 234: Dominant Character

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Ashish has a string  $s$  of length  $n$  containing only characters ‘a’, ‘b’ and ‘c’.

He wants to find the length of the smallest substring, which satisfies the following conditions:

Length of the substring is at least 2

‘a’ occurs strictly more times in this substring than ‘b’

‘a’ occurs strictly more times in this substring than ‘c’

Ashish is busy planning his next Codeforces round. Help him solve the problem.

A string  $a$  is a substring of a string  $b$  if  $a$  can be obtained from  $b$  by deletion of several (possibly, zero or all) characters from the beginning and several (possibly, zero or all) characters from the end.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     string s;
8     cin >> s;
9     int ans = n + 1;
10    for (int i = 0; i < n; i++) {
11        int cnt[3] {};
12        for (int j = i; j <= i + 6 && j < n; j++) {
13            cnt[s[j] - 'a']++;
14            if (j > i && cnt[0] > cnt[1] && cnt[0] > cnt[2]) {
15                ans = min(ans, j - i + 1);
16            }
17        }
18    }
19    if (ans > n) {
20        ans = -1;
21    }
22 }
```

```

21     }
22     cout << ans << "\n";
23 }
24 int main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
27     int t;
28     cin >> t;
29     while (t--) {
30         solve();
31     }
32     return 0;
33 }
```

## 235: Game of Ball Passing

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Daniel is watching a football team playing a game during their training session. They want to improve their passing skills during that session.

The game involves  $n$  players, making multiple passes towards each other. Unfortunately, since the balls were moving too fast, after the session Daniel is unable to know how many balls were involved during the game. The only thing he knows is the number of passes delivered by each player during all the session.

Find the minimum possible amount of balls that were involved in the game.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    i64 sum = accumulate(a.begin(), a.end(), 0LL);
12    int max = *max_element(a.begin(), a.end());
13    int ans;
14    if (max == 0) {
15        ans = 0;
16    } else if (2 * max <= sum + 1) {
```

```

17     ans = 1;
18 } else {
19     ans = 2 * max - sum;
20 }
21 cout << ans << "\n";
22 }
23 int main() {
24     ios::sync_with_stdio(false);
25     cin.tie(nullptr);
26     int t;
27     cin >> t;
28     while (t--) {
29         solve();
30     }
31     return 0;
32 }
```

## 236: Queries for the Array

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp had an array  $a$  consisting of integers. Initially, this array was empty.

Monocarp performed three types of queries to this array:

choose an integer and append it to the end of the array. Each time Monocarp performed a query of this type, he wrote out a character +;

remove the last element from the array. Each time Monocarp performed a query of this type, he wrote out a character -. Monocarp never performed this query on an empty array;

check if the array is sorted in non-descending order, i.e.  $a_1 \leq a_2 \leq \dots \leq a_k$ , where  $k$  is the number of elements in the array currently. Every array with less than 2 elements is considered sorted. If the array was sorted by the time Monocarp was performing that query, he wrote out a character 1. Otherwise, he wrote out a character 0.

You are given a sequence  $s$  of  $q$  characters 0, 1, + and/or -. These are the characters that were written out by Monocarp, given in the exact order he wrote them out.

You have to check if this sequence is consistent, i. e. it was possible for Monocarp to perform the queries so that the sequence of characters he wrote out is exactly  $s$ .

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
```

```

2  using namespace std;
3  using i64 = long long;
4  void solve() {
5      string s;
6      cin >> s;
7      vector<int> a;
8      int z = 0;
9      for (auto c : s) {
10          if (c == '+') {
11              a.push_back(-1);
12          } else if (c == '-') {
13              z -= (a.back() == 0);
14              int t = a.back();
15              a.pop_back();
16              if (t == 1 && !a.empty()) {
17                  a.back() = t;
18              }
19          } else if (c == '1') {
20              if (z) {
21                  cout << "NO\n";
22                  return;
23              }
24              if (!a.empty()) {
25                  a.back() = 1;
26              }
27          } else {
28              if (a.size() <= 1) {
29                  cout << "NO\n";
30                  return;
31              }
32              if (a.back() == 1) {
33                  cout << "NO\n";
34                  return;
35              }
36              if (a.back() == -1) {
37                  a.back() = 0;
38                  z++;
39              }
40          }
41      }
42      cout << "YES\n";
43  }
44 int main() {
45     ios::sync_with_stdio(false);
46     cin.tie(nullptr);
47     int t;
48     cin >> t;
49     while (t--) {
50         solve();
51     }
52     return 0;
53 }
```

### 237: Petya and Catacombs

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A very brave explorer Petya once decided to explore Paris catacombs. Since Petya is not really experienced, his exploration is just walking through the catacombs.

Catacombs consist of several rooms and bidirectional passages between some pairs of them. Some passages can connect a room to itself and since the passages are built on different depths they do not intersect each other. Every minute Petya arbitrary chooses a passage from the room he is currently in and then reaches the room on the other end of the passage in exactly one minute. When he enters a room at minute  $i$ , he makes a note in his logbook with number  $t_i$ :

If Petya has visited this room before, he writes down the minute he was in this room last time;

Otherwise, Petya writes down an arbitrary non-negative integer strictly less than current minute  $i$ .

Initially, Petya was in one of the rooms at minute 0, he didn't write down number  $t_0$ .

At some point during his wandering Petya got tired, threw out his logbook and went home. Vasya found his logbook and now he is curious: what is the minimum possible number of rooms in Paris catacombs according to Petya's logbook?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> cnt(n);
10    for (int i = 0; i < n; i++) {
11        int t;
12        cin >> t;
13        cnt[t]++;
14    }
15    int ans = n;
16    for (int i = 1; i < n; i++) {
17        if (cnt[i] > 0) {
18            ans--;
19        }
20    }
21    cout << ans << "\n";
22    return 0;
23 }
```

## 238: K-Dominant Character

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

You are given a string  $s$  consisting of lowercase Latin letters. Character  $c$  is called  $k$ -dominant iff each substring of  $s$  with length at least  $k$  contains this character  $c$ .

You have to find minimum  $k$  such that there exists at least one  $k$ -dominant character.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     string s;
8     cin >> s;
9     const int n = s.size();
10    vector<int> p[26];
11    for (int i = 0; i < 26; i++) {
12        p[i].push_back(-1);
13    }
14    for (int i = 0; i < n; i++) {
15        p[s[i] - 'a'].push_back(i);
16    }
17    for (int i = 0; i < 26; i++) {
18        p[i].push_back(n);
19    }
20    int ans = n;
21    for (int i = 0; i < 26; i++) {
22        int res = 0;
23        for (int j = 1; j < p[i].size(); j++) {
24            res = max(res, p[i][j] - p[i][j - 1]);
25        }
26        ans = min(ans, res);
27    }
28    cout << ans << "\n";
29    return 0;
30 }
```

### 239: Prefix Permutation Sums

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Your friends have an array of  $n$  elements, calculated its array of prefix sums and passed it to you, accidentally losing one element during the transfer. Your task is to find out if the given array can matches permutation.

A permutation of  $n$  elements is an array of  $n$  numbers from 1 to  $n$  such that each number occurs exactly one times in it.

The array of prefix sums of the array  $a$  - is such an array  $b$  that  $b_i = \sum_{j=1}^i a_j, 1 \leq i \leq n$ .

For example, the original permutation was [1, 5, 2, 4, 3]. Its array of prefix sums - [1, 6, 8, 12, 15]. Having lost one element, you can get, for example, arrays [6, 8, 12, 15] or [1, 6, 8, 15].

It can also be shown that the array [1, 2, 100] does not correspond to any permutation.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<i64> a(n);
8     for (int i = 1; i < n; i++) {
9         cin >> a[i];
10    }
11    if (a[n - 1] > 1LL * n * (n + 1) / 2) {
12        cout << "NO\n";
13        return;
14    }
15    vector<int> cnt(2 * n + 1);
16    for (int i = 1; i < n; i++) {
17        if (a[i] - a[i - 1] > 2 * n) {
18            cout << "NO\n";
19            return;
20        }
21        cnt[a[i] - a[i - 1]]++;
22    }
23    if (a[n - 1] < 1LL * n * (n + 1) / 2) {
24        for (int i = 1; i <= 2 * n; i++) {
25            if (cnt[i] > (i <= n)) {
26                cout << "NO\n";
27                return;
28            }
29        }
30        cout << "YES\n";
31        return;
32    }
33    int x = find(cnt.begin() + 1, cnt.end(), 0) - cnt.begin();
34    int y;
35    for (int i = 1; i <= 2 * n; i++) {
36        if (cnt[i] > (i <= n)) {
37            y = i;
38            break;
39        }
40    }
41    if (y <= x) {
42        cout << "NO\n";
43        return;
44    }
45    cnt[y]--;

```

```

46     cnt[x]++;
47     cnt[y - x]++;
48     for (int i = 1; i <= n; i++) {
49         if (cnt[i] != 1) {
50             cout << "NO\n";
51             return;
52         }
53     }
54     cout << "YES\n";
55 }
56 int main() {
57     ios::sync_with_stdio(false);
58     cin.tie(nullptr);
59     int t;
60     cin >> t;
61     while (t--) {
62         solve();
63     }
64     return 0;
65 }
```

## 240: The Morning Star

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A compass points directly toward the morning star. It can only point in one of eight directions: the four cardinal directions (N, S, E, W) or some combination (NW, NE, SW, SE). Otherwise, it will break.

There are  $n$  distinct points with integer coordinates on a plane. How many ways can you put a compass at one point and the morning star at another so that the compass does not break?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     map<int, int> f[4];
8     i64 ans = 0;
9     for (int i = 0; i < n; i++) {
10         int x, y;
11         cin >> x >> y;
12         ans += f[0][x]++;
13         ans += f[1][y]++;
14         ans += f[2][x + y]++;
15         ans += f[3][x - y]++;
16     }
17     cout << ans * 2 << "\n";
```

```

18 }
19 int main() {
20     ios::sync_with_stdio(false);
21     cin.tie(nullptr);
22     int t;
23     cin >> t;
24     while (t--) {
25         solve();
26     }
27     return 0;
28 }
```

## 241: We Were Both Children

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Mihai and Slavic were looking at a group of  $n$  frogs, numbered from 1 to  $n$ , all initially located at point 0. Frog  $i$  has a hop length of  $a_i$ .

Each second, frog  $i$  hops  $a_i$  units forward. Before any frogs start hopping, Slavic and Mihai can place exactly one trap in a coordinate in order to catch all frogs that will ever pass through the corresponding coordinate.

However, the children can't go far away from their home so they can only place a trap in the first  $n$  points (that is, in a point with a coordinate between 1 and  $n$ ) and the children can't place a trap in point 0 since they are scared of frogs.

Can you help Slavic and Mihai find out what is the maximum number of frogs they can catch using a trap?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> cnt(n + 1);
8     for (int i = 0; i < n; i++) {
9         int a;
10        cin >> a;
11        if (a <= n) {
12            cnt[a]++;
13        }
14    }
15    for (int i = n; i >= 1; i--) {
```

```

16         for (int j = 2 * i; j <= n; j += i) {
17             cnt[j] += cnt[i];
18         }
19     }
20     auto ans = *max_element(cnt.begin(), cnt.end());
21     cout << ans << "\n";
22 }
23 int main() {
24     ios::sync_with_stdio(false);
25     cin.tie(nullptr);
26     int t;
27     cin >> t;
28     while (t--) {
29         solve();
30     }
31     return 0;
32 }
```

## 242: Particles

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have discovered  $n$  mysterious particles on a line with integer charges of  $c_1, \dots, c_n$ . You have a device that allows you to perform the following operation:

Choose a particle and remove it from the line. The remaining particles will shift to fill in the gap that is created. If there were particles with charges  $x$  and  $y$  directly to the left and right of the removed particle, they combine into a single particle of charge  $x + y$ .

For example, if the line of particles had charges of  $[-3, 1, 4, -1, 5, -9]$ , performing the operation on the 4th particle will transform the line into  $[-3, 1, 9, -9]$ .

If we then use the device on the 1st particle in this new line, the line will turn into  $[1, 9, -9]$ .

You will perform operations until there is only one particle left. What is the maximum charge of this remaining particle that you can obtain?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> c(n);
8     for (int i = 0; i < n; i++) {
```

```

9      cin >> c[i];
10     }
11     i64 ans = -1E18;
12     for (int t = 0; t < 2; t++) {
13         i64 sum = 0;
14         int mx = -1E9;
15         for (int i = t; i < n; i += 2) {
16             if (c[i] > 0) {
17                 sum += c[i];
18             }
19             mx = max(mx, c[i]);
20         }
21         if (sum == 0) {
22             sum = mx;
23         }
24         ans = max(ans, sum);
25     }
26     cout << ans << "\n";
27 }
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(nullptr);
31     int t;
32     cin >> t;
33     while (t--) {
34         solve();
35     }
36     return 0;
37 }
```

### 243: Rudolf and Snowflakes (simple version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is a simple version of the problem. The only difference is that in this version  $n \leq 10^6$ .

One winter morning, Rudolf was looking thoughtfully out the window, watching the falling snowflakes. He quickly noticed a certain symmetry in the configuration of the snowflakes. And like a true mathematician, Rudolf came up with a mathematical model of a snowflake.

He defined a snowflake as an undirected graph constructed according to the following rules:

Initially, the graph has only one vertex.

Then, more vertices are added to the graph. The initial vertex is connected by edges to  $k$  new vertices ( $k > 1$ ).

Each vertex that is connected to only one other vertex is connected by edges to  $k$  more new vertices. This step should be done at least once.

The smallest possible snowflake for  $k = 4$  is shown in the figure.

After some mathematical research, Rudolf realized that such snowflakes may not have any number of vertices. Help Rudolf check if a snowflake with  $n$  vertices can exist.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     i64 n;
6     cin >> n;
7     for (int i = 2; i <= 60; i++) {
8         int d;
9         if (i == 2) {
10             d = sqrt(n);
11         } else if (i == 3) {
12             d = cbrt(n);
13         } else {
14             d = pow(n, 1.0 / i);
15         }
16         if (d == 1) {
17             continue;
18         }
19         i64 res = 0;
20         i64 p = 1;
21         for (int t = 0; t <= i; t++) {
22             if (t) {
23                 p *= d;
24             }
25             res += p;
26         }
27         if (res == n) {
28             cout << "YES\n";
29             return;
30         }
31     }
32     cout << "NO\n";
33 }
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     cout << fixed << setprecision(10);
38     int t;
39     cin >> t;
40     while (t--) {
41         solve();
42     }
43     return 0;
44 }
```

## 244: LuoTianyi and the Show

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

There are  $n$  people taking part in a show about VOCALOID. They will sit in the row of seats, numbered 1 to  $m$  from left to right.

The  $n$  people come and sit in order. Each person occupies a seat in one of three ways:

Sit in the seat next to the left of the leftmost person who is already sitting, or if seat 1 is taken, then leave the show. If there is no one currently sitting, sit in seat  $m$ .

Sit in the seat next to the right of the rightmost person who is already sitting, or if seat  $m$  is taken, then leave the show. If there is no one currently sitting, sit in seat 1.

Sit in the seat numbered  $x_i$ . If this seat is taken, then leave the show.

Now you want to know what is the maximum number of people that can take a seat, if you can let people into the show in any order?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> m >> n;
7     int cnt1 = 0, cnt2 = 0;
8     vector<int> a;
9     for (int i = 0; i < m; i++) {
10         int x;
11         cin >> x;
12         if (x == -1) {
13             cnt1++;
14         } else if (x == -2) {
15             cnt2++;
16         } else {
17             a.push_back(x);
18         }
19     }
20     sort(a.begin(), a.end());
21     a.erase(unique(a.begin(), a.end()), a.end());
22     int ans = 0;
23     ans = max(ans, cnt1 + int(a.size()));
24     ans = max(ans, cnt2 + int(a.size()));
25     ans = min(ans, n);
26     for (int i = 0; i < a.size(); i++) {
27         int l = min(a[i] - 1, i + cnt1);
28         int r = min(n - a[i], int(a.size()) - 1 - i + cnt2);
29         ans = max(ans, l + 1 + r);
30     }
31     cout << ans << "\n";
32 }
33 int main() {
34     ios::sync_with_stdio(false);
35     cin.tie(nullptr);
36     int t;
37     cin >> t;

```

```

38     while (t--) {
39         solve();
40     }
41     return 0;
42 }
```

## 245: Hits Different

- Time limit: 2.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In a carnival game, there is a huge pyramid of cans with 2023 rows, numbered in a regular pattern as shown.

If can  $9^2$  is hit initially, then all cans colored red in the picture above would fall.

You throw a ball at the pyramid, and it hits a single can with number  $n^2$ . This causes all cans that are stacked on top of this can to fall (that is, can  $n^2$  falls, then the cans directly above  $n^2$  fall, then the cans directly above those cans, and so on). For example, the picture above shows the cans that would fall if can  $9^2$  is hit.

What is the sum of the numbers on all cans that fall? Recall that  $n^2 = n \times n$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 i64 sum2(i64 n) {
5     return n * (n + 1) * (2 * n + 1) / 6;
6 }
7 i64 sum2(i64 l, i64 r) {
8     return sum2(r) - sum2(l - 1);
9 }
10 void solve() {
11     int n;
12     cin >> n;
13     int x = 1;
14     while (x * (x + 1) / 2 < n) {
15         x++;
16     }
17     int y = n - x * (x - 1) / 2;
18     i64 ans = 0;
19     for (int i = 1; i <= x; i++) {
20         int l = max(1, y - (x - i));
21         int r = min(i, y);
22         ans += sum2(i * (i - 1) / 2 + l, i * (i - 1) / 2 + r);
23     }
24     cout << ans << "\n";
```

```

25 }
26 int main() {
27     ios::sync_with_stdio(false);
28     cin.tie(nullptr);
29     int t;
30     cin >> t;
31     while (t--) {
32         solve();
33     }
34     return 0;
35 }
```

## 246: Easy Number Challenge

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Let's denote  $d(n)$  as the number of divisors of a positive integer  $n$ . You are given three integers  $a$ ,  $b$  and  $c$ . Your task is to calculate the following sum:

Find the sum modulo 1073741824 (230).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int N = 1E6;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     vector<int> d(N + 1);
9     for (int i = 1; i <= N; i++) {
10         for (int j = i; j <= N; j += i) {
11             d[j]++;
12         }
13     }
14     int a, b, c;
15     cin >> a >> b >> c;
16     int ans = 0;
17     for (int i = 1; i <= a; i++) {
18         for (int j = 1; j <= b; j++) {
19             for (int k = 1; k <= c; k++) {
20                 ans += d[i * j * k];
21             }
22         }
23     }
24     cout << ans << "\n";
25     return 0;
26 }
```

## 247: XOR and OR

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The Bitlandians are quite weird people. They do everything differently. They have a different alphabet so they have a different definition for a string.

A Bitlandish string is a string made only of characters “0” and “1”.

BitHaval (the mayor of Bitland) loves to play with Bitlandish strings. He takes some Bitlandish string  $a$ , and applies several (possibly zero) operations to it. In one operation the mayor may take any two adjacent characters of a string, define one of them as  $x$  and the other one as  $y$ . Then he calculates two values  $p$  and  $q$ :  $p = x \text{ xor } y$ ,  $q = x \text{ or } y$ . Then he replaces one of the two taken characters by  $p$  and the other one by  $q$ .

The  $\text{xor}$  operation means the bitwise excluding OR operation. The  $\text{or}$  operation is the bitwise OR operation.

So for example one operation can transform string 11 to string 10 or to string 01. String 1 cannot be transformed into any other string.

You've got two Bitlandish strings  $a$  and  $b$ . Your task is to check if it is possible for BitHaval to transform string  $a$  to string  $b$  in several (possibly zero) described operations.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     string a, b;
8     cin >> a >> b;
9     if (a.size() == b.size() && (a.find('1') != -1) == (b.find('1') != -1)) {
10         cout << "YES\n";
11     } else {
12         cout << "NO\n";
13     }
14     return 0;
15 }
```

## 248: Cd and pwd commands

- Time limit: 3 seconds

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vasya is writing an operating system shell, and it should have commands for working with directories. To begin with, he decided to go with just two commands: cd (change the current directory) and pwd (display the current directory).

Directories in Vasya's operating system form a traditional hierarchical tree structure. There is a single root directory, denoted by the slash character “/”. Every other directory has a name - a non-empty string consisting of lowercase Latin letters. Each directory (except for the root) has a parent directory - the one that contains the given directory. It is denoted as “..”.

The command cd takes a single parameter, which is a path in the file system. The command changes the current directory to the directory specified by the path. The path consists of the names of directories separated by slashes. The name of the directory can be “..”, which means a step up to the parent directory. .. can be used in any place of the path, maybe several times. If the path begins with a slash, it is considered to be an absolute path, that is, the directory changes to the specified one, starting from the root. If the parameter begins with a directory name (or “..”), it is considered to be a relative path, that is, the directory changes to the specified directory, starting from the current one.

The command pwd should display the absolute path to the current directory. This path must not contain “..”.

Initially, the current directory is the root. All directories mentioned explicitly or passed indirectly within any command cd are considered to exist. It is guaranteed that there is no attempt of transition to the parent directory of the root directory.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<string> a;
10    while (n--) {
11        string o;
12        cin >> o;
13        if (o == "pwd") {
14            for (auto s : a) {
15                cout << "/" << s;
16            }
17            cout << "/\n";
18        } else {
```

```

19         string s;
20         cin >> s;
21         int i = 0;
22         while (true) {
23             int j = s.find('/', i);
24             if (!j) {
25                 a.clear();
26             } else if (j == -1) {
27                 string t = s.substr(i);
28                 if (t == "..") {
29                     a.pop_back();
30                 } else {
31                     a.push_back(t);
32                 }
33                 break;
34             } else {
35                 string t = s.substr(i, j - i);
36                 if (t == "..") {
37                     a.pop_back();
38                 } else {
39                     a.push_back(t);
40                 }
41             }
42             i = j+1;
43         }
44     }
45 }
46 return 0;
47 }
```

## 249: Progress Bar

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A progress bar is an element of graphical interface that displays the progress of a process for this very moment before it is completed. Let's take a look at the following form of such a bar.

A bar is represented as  $n$  squares, located in line. To add clarity, let's number them with positive integers from 1 to  $n$  from the left to the right. Each square has saturation ( $a_i$  for the  $i$ -th square), which is measured by an integer from 0 to  $k$ . When the bar for some  $i$  ( $1 \leq i \leq n$ ) is displayed, squares 1, 2, ...,  $i - 1$  has the saturation  $k$ , squares  $i + 1, i + 2, \dots, n$  has the saturation 0, and the saturation of the square  $i$  can have any value from 0 to  $k$ .

So some first squares of the progress bar always have the saturation  $k$ . Some last squares always have the saturation 0. And there is no more than one square that has the saturation different from 0 and  $k$ .

The degree of the process's completion is measured in percents. Let the process be  $t\%$  completed. Then the following inequation is fulfilled:

An example of such a bar can be seen on the picture.

For the given  $n, k, t$  determine the measures of saturation for all the squares  $a_i$  of the progress bar.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, k, t;
8     cin >> n >> k >> t;
9     int c = n * k * t / 100;
10    for (int i = 0; i < n; i++) {
11        int v = min(c, k);
12        cout << v << "\n"[i == n-1];
13        c -= v;
14    }
15    return 0;
16 }
```

## 250: Guilty - to the kitchen!

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

It's a very unfortunate day for Volodya today. He got bad mark in algebra and was therefore forced to do some work in the kitchen, namely to cook borsch (traditional Russian soup). This should also improve his algebra skills.

According to the borsch recipe it consists of  $n$  ingredients that have to be mixed in proportion litres (thus, there should be  $a_1 \cdot x, \dots, a_n \cdot x$  litres of corresponding ingredients mixed for some non-negative  $x$ ). In the kitchen Volodya found out that he has  $b_1, \dots, b_n$  litres of these ingredients at his disposal correspondingly. In order to correct his algebra mistakes he ought to cook as much soup as possible in a  $V$  litres volume pan (which means the amount of soup cooked can be between 0 and  $V$  litres). What is the volume of borsch Volodya will cook ultimately?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, V;
8     cin >> n >> V;
9     vector<int> a(n), b(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    for (int i = 0; i < n; i++) {
14        cin >> b[i];
15    }
16    int suma = accumulate(a.begin(), a.end(), 0);
17    double ans = V;
18    for (int i = 0; i < n; i++) {
19        ans = min(ans, 1. * b[i] / a[i] * suma);
20    }
21    cout << fixed << setprecision(10) << ans << "\n";
22 }
23 }
```

## math

### 251: Yarik and Musical Notes

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Yarik is a big fan of many kinds of music. But Yarik loves not only listening to music but also writing it. He likes electronic music most of all, so he has created his own system of music notes, which, in his opinion, is best for it.

Since Yarik also likes informatics, in his system notes are denoted by integers of  $2^k$ , where  $k \geq 1$  - a positive integer. But, as you know, you can't use just notes to write music, so Yarik uses combinations of two notes. The combination of two notes  $(a, b)$ , where  $a = 2^k$  and  $b = 2^l$ , he denotes by the integer  $a^b$ .

For example, if  $a = 8 = 2^3$ ,  $b = 4 = 2^2$ , then the combination  $(a, b)$  is denoted by the integer  $a^b = 8^4 = 4096$ . Note that different combinations can have the same notation, e.g., the combination  $(64, 2)$  is also denoted by the integer  $4096 = 64^2$ .

Yarik has already chosen  $n$  notes that he wants to use in his new melody. However, since their integers can be very large, he has written them down as an array  $a$  of length  $n$ , then the note  $i$  is  $b_i = 2^{a_i}$ . The integers in array  $a$  can be repeated.

The melody will consist of several combinations of two notes. Yarik was wondering how many pairs of notes  $b_i, b_j$  ( $i < j$ ) exist such that the combination  $(b_i, b_j)$  is equal to the combination  $(b_j, b_i)$ . In other words, he wants to count the number of pairs  $(i, j)$  ( $i < j$ ) such that  $b_i^{b_j} = b_j^{b_i}$ . Help him find the number of such pairs.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     map<int, int> cnt;
8     i64 ans = 0;
9     for (int i = 0; i < n; i++) {
10         int a;
11         cin >> a;
12         ans += cnt[a]++;
13     }
14     ans += 1LL * cnt[1] * cnt[2];
15     cout << ans << "\n";
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

## 252: Torn Lucky Ticket

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

A ticket is a non-empty string of digits from 1 to 9.

A lucky ticket is such a ticket that:

it has an even length;

the sum of digits in the first half is equal to the sum of digits in the second half.

You are given  $n$  ticket pieces  $s_1, s_2, \dots, s_n$ . How many pairs  $(i, j)$  (for  $1 \leq i, j \leq n$ ) are there such that  $s_i + s_j$  is a lucky ticket? Note that it's possible that  $i = j$ .

Here, the `+` operator denotes the concatenation of the two strings. For example, if  $s_i$  is 13, and  $s_j$  is 37, then  $s_i + s_j$  is 1337.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     array<vector<string>, 6> f;
10    for (int i = 0; i < n; i++) {
11        string s;
12        cin >> s;
13        f[s.size()].push_back(s);
14    }
15    i64 ans = 0;
16    for (int x = 1; x <= 5; x++) {
17        for (int y = 1; y <= 5; y++) {
18            if ((x + y) % 2 == 0) {
19                array<int, 100> cnt{};
20                int h = (x + y) / 2;
21                for (auto a : f[x]) {
22                    int s = 50;
23                    for (int i = 0; i < x; i++) {
24                        if (i < h) {
25                            s += a[i] - '0';
26                        } else {
27                            s -= a[i] - '0';
28                        }
29                    }
30                    cnt[s] += 1;
31                }
32                for (auto a : f[y]) {
33                    int s = 50;
34                    for (int i = 0; i < y; i++) {
35                        if (x + i >= h) {
36                            s += a[i] - '0';
37                        } else {
38                            s -= a[i] - '0';
39                        }
40                    }
41                    ans += cnt[s];
42                }
43            }
44        }
45    }
46    cout << ans << "\n";
47    return 0;
48 }
```

## 253: Rubik's Cube Coloring (easy version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

It is the easy version of the problem. The difference is that in this version, there are no nodes with already chosen colors.

Theofanis is starving, and he wants to eat his favorite food, sheftalia. However, he should first finish his homework. Can you help him with this problem?

You have a perfect binary tree of  $2^k - 1$  nodes - a binary tree where all vertices  $i$  from 1 to  $2^k - 1$  have exactly two children: vertices  $2i$  and  $2i + 1$ . Vertices from  $2^{k-1}$  to  $2^k - 1$  don't have any children. You want to color its vertices with the 6 Rubik's cube colors (White, Green, Red, Blue, Orange and Yellow).

Let's call a coloring good when all edges connect nodes with colors that are neighboring sides in the Rubik's cube.

More formally:

a white node can not be neighboring with white and yellow nodes;

a yellow node can not be neighboring with white and yellow nodes;

a green node can not be neighboring with green and blue nodes;

a blue node can not be neighboring with green and blue nodes;

a red node can not be neighboring with red and orange nodes;

an orange node can not be neighboring with red and orange nodes;

You want to calculate the number of the good colorings of the binary tree. Two colorings are considered different if at least one node is colored with a different color.

The answer may be too large, so output the answer modulo  $10^9 + 7$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {

```

```

9         res *= a;
10    }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int k;
38     cin >> k;
39     Z ans = 6 * power(Z(4), (1LL << k) - 2);
40     cout << ans << "\n";
41     return 0;
42 }

```

## 254: Divide and Equalize

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  consisting of  $n$  positive integers. You can perform the following operation on it:

Choose a pair of elements  $a_i$  and  $a_j$  ( $1 \leq i, j \leq n$  and  $i \neq j$ );

Choose one of the divisors of the integer  $a_i$ , i.e., an integer  $x$  such that  $a_i \bmod x = 0$ ;

Replace  $a_i$  with  $\frac{a_i}{x}$  and  $a_j$  with  $a_j \cdot x$ .

For example, let's consider the array  $a = [100, 2, 50, 10, 1]$  with 5 elements. Perform two operations on it:

Choose  $a_3 = 50$  and  $a_2 = 2, x = 5$ . Replace  $a_3$  with  $\frac{a_3}{x} = \frac{50}{5} = 10$ , and  $a_2$  with  $a_2 \cdot x = 2 \cdot 5 = 10$ . The resulting array is  $a = [100, 10, 10, 10, 1]$ ;

Choose  $a_1 = 100$  and  $a_5 = 1, x = 10$ . Replace  $a_1$  with  $\frac{a_1}{x} = \frac{100}{10} = 10$ , and  $a_5$  with  $a_5 \cdot x = 1 \cdot 10 = 10$ . The resulting array is  $a = [10, 10, 10, 10, 10]$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 vector<int> minp, primes;
5 void sieve(int n) {
6     minp.assign(n + 1, 0);
7     primes.clear();
8     for (int i = 2; i <= n; i++) {
9         if (minp[i] == 0) {
10             minp[i] = i;
11             primes.push_back(i);
12         }
13         for (auto p : primes) {
14             if (i * p > n) {
15                 break;
16             }
17             minp[i * p] = p;
18             if (p == minp[i]) {
19                 break;
20             }
21         }
22     }
23 }
24 constexpr int V = 1E6;
25 int cnt[V + 1];
26 void solve() {
27     int n;
28     cin >> n;
29     vector<int> stk;
30     for (int i = 0; i < n; i++) {
31         int x;
32         cin >> x;
33         while (x > 1) {
34             int p = minp[x];
35             x /= p;
36             stk.push_back(p);
37             cnt[p]++;
38         }
39     }
40     bool ok = true;
41     for (auto x : stk) {
42         if (cnt[x] % n) {
43             ok = false;
44         }
45         cnt[x] = 0;
46     }
47     cout << (ok ? "YES" : "NO") << "\n";
48 }
49 int main() {
50     ios::sync_with_stdio(false);

```

```

51     cin.tie(nullptr);
52     sieve(V);
53     int t;
54     cin >> t;
55     while (t--) {
56         solve();
57     }
58     return 0;
59 }
```

## 255: Money Trees

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Luca is in front of a row of  $n$  trees. The  $i$ -th tree has  $a_i$  fruit and height  $h_i$ .

He wants to choose a contiguous subarray of the array  $[h_l, h_{l+1}, \dots, h_r]$  such that for each  $i$  ( $l \leq i < r$ ),  $h_i$  is divisible<sup>†</sup> by  $h_{i+1}$ . He will collect all the fruit from each of the trees in the subarray (that is, he will collect  $a_l + a_{l+1} + \dots + a_r$  fruits). However, if he collects more than  $k$  fruits in total, he will get caught.

What is the maximum length of a subarray Luca can choose so he doesn't get caught?

<sup>†</sup>  $x$  is divisible by  $y$  if the ratio  $\frac{x}{y}$  is an integer.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> s(n + 1), h(n);
8     for (int i = 0; i < n; i++) {
9         int a;
10        cin >> a;
11        s[i + 1] = s[i] + a;
12    }
13    for (int i = 0; i < n; i++) {
14        cin >> h[i];
15    }
16    int ans = 0;
17    for (int i = 0, j = 0; i < n; i++) {
18        if (i && h[i - 1] % h[i] != 0) {
19            j = i;
20        }
21        while (s[i + 1] - s[j] > k) {
22            j++;
23        }
24        if (s[i + 1] - s[j] == k) {
25            ans = max(ans, i - j + 1);
26        }
27    }
28    cout << ans << endl;
29 }
```

```

23         }
24         ans = max(ans, i - j + 1);
25     }
26     cout << ans << "\n";
27 }
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(nullptr);
31     int t;
32     cin >> t;
33     while (t--) {
34         solve();
35     }
36     return 0;
37 }
```

## 256: Colorful Table

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given two integers  $n$  and  $k$ . You are also given an array of integers  $a_1, a_2, \dots, a_n$  of size  $n$ . It is known that for all  $1 \leq i \leq n, 1 \leq a_i \leq k$ .

Define a two-dimensional array  $b$  of size  $n \times n$  as follows:  $b_{i,j} = \min(a_i, a_j)$ . Represent array  $b$  as a square, where the upper left cell is  $b_{1,1}$ , rows are numbered from top to bottom from 1 to  $n$ , and columns are numbered from left to right from 1 to  $n$ . Let the color of a cell be the number written in it (for a cell with coordinates  $(i, j)$ , this is  $b_{i,j}$ ).

For each color from 1 to  $k$ , find the smallest rectangle in the array  $b$  containing all cells of this color. Output the sum of width and height of this rectangle.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> a(n);
8     vector<int> l(k, n), r(k, -1);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11         a[i]--;
12         l[a[i]] = min(l[a[i]], i);
13         r[a[i]] = i;
14     }
15 }
```

```

14     }
15     auto sl = l, sr = r;
16     for (int i = k - 2; i >= 0; i--) {
17         sl[i] = min(sl[i], sl[i + 1]);
18         sr[i] = max(sr[i], sr[i + 1]);
19     }
20     for (int i = 0; i < k; i++) {
21         if (l[i] > r[i]) {
22             cout << 0;
23         } else {
24             cout << 2 * (sr[i] - sl[i] + 1);
25         }
26         cout << " \n"[i == k - 1];
27     }
28 }
29 int main() {
30     ios::sync_with_stdio(false);
31     cin.tie(nullptr);
32     int t;
33     cin >> t;
34     while (t--) {
35         solve();
36     }
37     return 0;
38 }
```

## 257: Vupsen, Pupsen and 0

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vupsen and Pupsen were gifted an integer array. Since Vupsen doesn't like the number 0, he threw away all numbers equal to 0 from the array. As a result, he got an array  $a$  of length  $n$ .

Pupsen, on the contrary, likes the number 0 and he got upset when he saw the array without zeroes. To cheer Pupsen up, Vupsen decided to come up with another array  $b$  of length  $n$  such that  $\sum_{i=1}^n a_i \cdot b_i = 0$ . Since Vupsen doesn't like number 0, the array  $b$  must not contain numbers equal to 0. Also, the numbers in that array must not be huge, so the sum of their absolute values cannot exceed  $10^9$ . Please help Vupsen to find any such array  $b$ !

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
```

```

6      cin >> n;
7      vector<int> a(n), b(n);
8      map<int, int> p;
9      for (int i = 0; i < n; i++) {
10         cin >> a[i];
11         if (p.count(a[i])) {
12             b[p[a[i]]] = 1;
13             b[i] = -1;
14             p.erase(a[i]);
15         } else {
16             p[a[i]] = i;
17         }
18     }
19     while (p.size() >= 2) {
20         auto [i, j] = *p.begin();
21         p.erase(p.begin());
22         auto [x, y] = *p.begin();
23         p.erase(p.begin());
24         b[j] = x;
25         b[y] = -i;
26     }
27     if (!p.empty()) {
28         auto [i, j] = *p.begin();
29         int k = !j;
30         do {
31             b[j] += a[k];
32             b[k] -= i;
33         } while (b[k] == 0);
34     }
35     for (int i = 0; i < n; i++) {
36         cout << b[i] << " \n"[i == n - 1];
37     }
38 }
39 int main() {
40     ios::sync_with_stdio(false);
41     cin.tie(nullptr);
42     int t;
43     cin >> t;
44     while (t--) {
45         solve();
46     }
47     return 0;
48 }
```

## 258: Array Elimination

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given array  $a_1, a_2, \dots, a_n$ , consisting of non-negative integers.

Let's define operation of "elimination" with integer parameter  $k$  ( $1 \leq k \leq n$ ) as follows:

Choose  $k$  distinct array indices  $1 \leq i_1 < i_2 < \dots < i_k \leq n$ .

Calculate  $x = a_{i_1} \& a_{i_2} \& \dots \& a_{i_k}$ , where  $\&$  denotes the bitwise AND operation (notes section contains formal definition).

Subtract  $x$  from each of  $a_{i_1}, a_{i_2}, \dots, a_{i_k}$ ; all other elements remain untouched.

Find all possible values of  $k$ , such that it's possible to make all elements of array  $a$  equal to 0 using a finite number of elimination operations with parameter  $k$ . It can be proven that exists at least one possible  $k$  for any array  $a$ .

Note that you firstly choose  $k$  and only after that perform elimination operations with value  $k$  you've chosen initially.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     int cnt[30] {};
8     for (int i = 0; i < n; i++) {
9         int a;
10        cin >> a;
11        for (int j = 0; j < 30; j++) {
12            cnt[j] += a >> j & 1;
13        }
14    }
15    int ans = 0;
16    for (int i = 0; i < 30; i++) {
17        ans = gcd(ans, cnt[i]);
18    }
19    for (int i = 1; i <= n; i++) {
20        if (ans % i == 0) {
21            cout << i << " ";
22        }
23    }
24    cout << "\n";
25 }
26 int main() {
27     ios::sync_with_stdio(false);
28     cin.tie(nullptr);
29     int t;
30     cin >> t;
31     while (t--) {
32         solve();
33     }
34     return 0;
35 }
```

## 259: Minimize Distance

- Time limit: 1 second
- Memory limit: 256 megabytes

- Input file: standard input
- Output file: standard output

A total of  $n$  depots are located on a number line. Depot  $i$  lies at the point  $x_i$  for  $1 \leq i \leq n$ .

You are a salesman with  $n$  bags of goods, attempting to deliver one bag to each of the  $n$  depots. You and the  $n$  bags are initially at the origin 0. You can carry up to  $k$  bags at a time. You must collect the required number of goods from the origin, deliver them to the respective depots, and then return to the origin to collect your next batch of goods.

Calculate the minimum distance you need to cover to deliver all the bags of goods to the depots. You do not have to return to the origin after you have delivered all the bags.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> x(n);
8     for (int i = 0; i < n; i++) {
9         cin >> x[i];
10    }
11    sort(x.begin(), x.end());
12    i64 ans = 0;
13    for (int i = 0; i < n && x[i] < 0; i += k) {
14        ans += -x[i];
15    }
16    for (int i = n - 1; i >= 0 && x[i] > 0; i -= k) {
17        ans += x[i];
18    }
19    ans *= 2;
20    ans -= max(abs(x[0]), abs(x[n - 1]));
21    cout << ans << "\n";
22 }
23 int main() {
24     ios::sync_with_stdio(false);
25     cin.tie(nullptr);
26     int t;
27     cin >> t;
28     while (t--) {
29         solve();
30     }
31     return 0;
32 }
```

## 260: Battling with Numbers

- Time limit: 1 second

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

On the trip to campus during the mid semester exam period, Chaneka thinks of two positive integers  $X$  and  $Y$ . Since the two integers can be very big, both are represented using their prime factorisations, such that:

$$X = A_1^{B_1} \times A_2^{B_2} \times \dots \times A_N^{B_N} \text{ (each } A_i \text{ is prime, each } B_i \text{ is positive, and } A_1 < A_2 < \dots < A_N\text{)}$$

$$Y = C_1^{D_1} \times C_2^{D_2} \times \dots \times C_M^{D_M} \text{ (each } C_j \text{ is prime, each } D_j \text{ is positive, and } C_1 < C_2 < \dots < C_M\text{)}$$

Chaneka ponders about these two integers for too long throughout the trip, so Chaneka's friend commands her "Gece, deh!" (move fast) in order to not be late for the exam.

Because of that command, Chaneka comes up with a problem, how many pairs of positive integers  $p$  and  $q$  such that  $\text{LCM}(p, q) = X$  and  $\text{GCD}(p, q) = Y$ . Since the answer can be very big, output the answer modulo 998 244 353.

Notes:

$\text{LCM}(p, q)$  is the smallest positive integer that is simultaneously divisible by  $p$  and  $q$ .

$\text{GCD}(p, q)$  is the biggest positive integer that simultaneously divides  $p$  and  $q$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;

```

```

28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 constexpr int V = 2E6;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int N;
39     cin >> N;
40     vector<int> A(N);
41     for (int i = 0; i < N; i++) {
42         cin >> A[i];
43     }
44     vector<int> f(V), g(V);
45     for (int i = 0; i < N; i++) {
46         cin >> f[A[i]];
47     }
48     int M;
49     cin >> M;
50     vector<int> B(M);
51     for (int i = 0; i < M; i++) {
52         cin >> B[i];
53     }
54     for (int i = 0; i < M; i++) {
55         cin >> g[B[i]];
56     }
57     Z ans = 1;
58     for (int i = 0; i < V; i++) {
59         if (f[i] < g[i]) {
60             ans = 0;
61         }
62         if (f[i] > g[i]) {
63             ans *= 2;
64         }
65     }
66     cout << ans << "\n";
67     return 0;
68 }
```

## 261: Divisor Chain

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an integer  $x$ . Your task is to reduce  $x$  to 1.

To do that, you can do the following operation:

select a divisor  $d$  of  $x$ , then change  $x$  to  $x - d$ , i.e. reduce  $x$  by  $d$ . (We say that  $d$  is a divisor of  $x$  if  $d$  is a positive integer and there exists an integer  $q$  such that  $x = d \cdot q$ .)

There is an additional constraint: you cannot select the same value of  $d$  more than twice.

For example, for  $x = 5$ , the following scheme is invalid because 1 is selected more than twice:  $5 \xrightarrow{-1} 4 \xrightarrow{-1} 3 \xrightarrow{-1} 2 \xrightarrow{-1} 1$ . The following scheme is however a valid one:  $5 \xrightarrow{-1} 4 \xrightarrow{-2} 2 \xrightarrow{-1} 1$ .

Output any scheme which reduces  $x$  to 1 with at most 1000 operations. It can be proved that such a scheme always exists.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 vector<int> get(int x) {
5     if (x == 1) {
6         return {1};
7     }
8     vector<int> p;
9     for (int i = 2; i * i <= x; i++) {
10        while (x % i == 0) {
11            p.push_back(i);
12            x /= i;
13        }
14    }
15    if (x > 1) {
16        p.push_back(x);
17    }
18    if (p.size() == 1) {
19        auto res = get(p[0] - 1);
20        res.insert(res.begin(), p[0]);
21        return res;
22    }
23    reverse(p.begin(), p.end());
24    vector<int> ans{1};
25    for (auto x : p) {
26        auto res = get(x);
27        ans.pop_back();
28        for (auto &v : ans) {
29            v *= x;
30        }
31        ans.insert(ans.end(), res.begin(), res.end());
32    }
33    return ans;
34 }
35 void solve() {
36     int x;
37     cin >> x;
38     vector<int> ans = get(x);
39     cout << ans.size() << "\n";
40     for (auto x : ans) {
41         cout << x << " \n"[x == ans.back()];
42     }
43 }
44 int main() {
45     ios::sync_with_stdio(false);
46     cin.tie(nullptr);
47     int t;
48     cin >> t;
49     while (t--) {
50         solve();
51     }
52     return 0;

```

53 }

## 262: Ice Cream Balls

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Tema decided to improve his ice cream making skills. He has already learned how to make ice cream in a cone using exactly two balls.

Before his ice cream obsession, Tema was interested in mathematics. Therefore, he is curious about the minimum number of balls he needs to have in order to make exactly  $n$  different types of ice cream.

There are plenty possible ice cream flavours: 1, 2, 3, .... Tema can make two-balls ice cream with any flavours (probably the same).

Two ice creams are considered different if their sets of ball flavours are different. For example,  $\{1, 2\} = \{2, 1\}$ , but  $\{1, 1\} \neq \{1, 2\}$ .

For example, having the following ice cream balls:  $\{1, 1, 2\}$ , Tema can make only two types of ice cream:  $\{1, 1\}$  and  $\{1, 2\}$ .

Note, that Tema do not need to make all the ice cream cones at the same time. This means that he making ice cream cones independently. Also in order to make a following cone  $\{x, x\}$  for some  $x$ , Tema needs at least 2 balls of type  $x$ .

Help Tema answer this question. It can be shown that answer always exist.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     i64 n;
6     cin >> n;
7     i64 k = sqrt(2.0L * n);
8     while (k * (k + 1) <= 2 * n) {
9         k++;
10    }
11    i64 ans = k + n - k * (k - 1) / 2;
12    cout << ans << "\n";
13 }
14 int main() {

```

```

15     ios::sync_with_stdio(false);
16     cin.tie(nullptr);
17     int t;
18     cin >> t;
19     while (t--) {
20         solve();
21     }
22     return 0;
23 }
```

**263: Position in Fraction**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have a fraction  $\frac{a}{b}$ . You need to find the first occurrence of digit  $c$  into decimal notation of the fraction after decimal point.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int a, b, c;
8     cin >> a >> b >> c;
9     int v = a;
10    for (int i = 0; i < b; i++) {
11        v = v * 10;
12        if (v / b == c) {
13            cout << i + 1 << "\n";
14            return 0;
15        }
16        v %= b;
17    }
18    cout << -1 << "\n";
19    return 0;
20 }
```

**264: Pride**

- Time limit: 2 seconds
- Memory limit: 256 megabytes

- Input file: standard input
- Output file: standard output

You have an array  $a$  with length  $n$ , you can perform operations. Each operation is like this: choose two adjacent elements from  $a$ , say  $x$  and  $y$ , and replace one of them with  $\text{gcd}(x, y)$ , where  $\text{gcd}$  denotes the greatest common divisor.

What is the minimum number of operations you need to make all of the elements equal to 1?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     int ans = n + 1;
10    vector<int> a(n);
11    for (int i = 0; i < n; i++) {
12        cin >> a[i];
13    }
14    for (int i = 0; i < n; i++) {
15        int g = 0;
16        for (int j = i; j < n; j++) {
17            g = gcd(g, a[j]);
18            if (g == 1) {
19                ans = min(ans, j - i);
20                break;
21            }
22        }
23    }
24    if (ans > n) {
25        cout << -1 << "\n";
26        return 0;
27    }
28    int cnt = count(a.begin(), a.end(), 1);
29    if (cnt > 0) {
30        ans = n - cnt;
31    } else {
32        ans += n - 1;
33    }
34    cout << ans << "\n";
35    return 0;
36 }
```

## 265: The Walkway

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

There are  $n$  benches near the Main Walkway in Summer Infomatics School. These benches are numbered by integers from 1 to  $n$  in order they follow. Also there are  $m$  cookie sellers near the Walkway. The  $i$ -th ( $1 \leq i \leq m$ ) cookie sellers is located near the  $s_i$ -th bench.

Petya is standing in the beginning of the Walkway. He will pass near all benches starting from the 1-st bench and ending with the  $n$ -th bench. Petya passes the distance between two consecutive benches in 1 minute. He has a knapsack with an infinite amount of cookies. Petya is going to eat cookies from his knapsack and buy them from cookie sellers during the walk.

Petya eats cookies only near the benches according to the following rule: he will eat the cookie near the  $i$ -th ( $1 \leq i \leq n$ ) bench if and only if at least one of the following conditions holds:

There is a cookie seller near the  $i$ -th bench. Then Petya will buy a cookie from cookie seller and eat it immediately.

Petya has not yet eaten a cookie. Then Petya will take a cookie from his knapsack and eat it immediately.

At least  $d$  minutes passed since Petya ate the previous cookie. In other words, Petya has not eaten a cookie near the benches  $i - 1, i - 2, \dots, \max(i - d + 1, 1)$ . Then Petya will take a cookie from his knapsack and eat it immediately.

You may assume that Petya eats cookies instantly. Petya will not eat two or more cookies near the same bench.

You want to minimize the number of cookies Petya will eat during his walk. In order to do this, you will ask the administration of the Summer Informatics School to remove exactly one cookie seller from the Walkway before Petya starts his walk.

Please determine the minimum possible number of cookies Petya can eat after removing exactly one cookie seller. Also determine the number of cookie sellers, such that if you remove one of them, Petya will eat the minimum possible number of cookies.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m, d;
6     cin >> n >> m >> d;
7     vector<int> s(m + 2);
8     for (int i = 1; i <= m; i++) {
9         cin >> s[i];
10    }
11    s[0] = -d + 1;

```

```

12     s[m + 1] = n + 1;
13     int sum = 0;
14     for (int i = 1; i <= m + 1; i++) {
15         sum += (s[i] - s[i - 1] - 1) / d;
16     }
17     int ans = n + 1;
18     int cnt = 0;
19     for (int i = 1; i <= m; i++) {
20         int res = sum;
21         res -= (s[i] - s[i - 1] - 1) / d;
22         res -= (s[i + 1] - s[i] - 1) / d;
23         res += (s[i + 1] - s[i - 1] - 1) / d;
24         res += m - 1;
25         if (res < ans) {
26             ans = res;
27             cnt = 1;
28         } else if (res == ans) {
29             cnt += 1;
30         }
31     }
32     cout << ans << " " << cnt << "\n";
33 }
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int t;
38     cin >> t;
39     while (t--) {
40         solve();
41     }
42     return 0;
43 }
```

## 266: Almost Identity Permutations

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A permutation  $p$  of size  $n$  is an array such that every integer from 1 to  $n$  occurs exactly once in this array.

Let's call a permutation an almost identity permutation iff there exist at least  $n - k$  indices  $i$  ( $1 \leq i \leq n$ ) such that  $p_i = i$ .

Your task is to count the number of almost identity permutations for given numbers  $n$  and  $k$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
```

```

2  using namespace std;
3  using i64 = long long;
4  int main() {
5      ios::sync_with_stdio(false);
6      cin.tie(nullptr);
7      int n, k;
8      cin >> n >> k;
9      i64 ans = 1;
10     if (k >= 2) {
11         ans += n * (n - 1) / 2;
12     }
13     if (k >= 3) {
14         ans += n * (n - 1) * (n - 2) / 3;
15     }
16     if (k >= 4) {
17         ans += 1LL * n * (n - 1) * (n - 2) * (n - 3) / 24 * 9;
18     }
19     cout << ans << "\n";
20     return 0;
21 }
```

## 267: Sum and Product

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have an array  $a$  of length  $n$ .

Your task is to answer  $q$  queries: given  $x, y$ , find the number of pairs  $i$  and  $j$  ( $1 \leq i < j \leq n$ ) that both  $a_i + a_j = x$  and  $a_i \cdot a_j = y$ .

That is, for the array  $[1, 3, 2]$  and asking for  $x = 3, y = 2$  the answer is 1:

$i = 1$  and  $j = 2$  fail because  $1 + 3 = 4$  and not 3, also  $1 \cdot 3 = 3$  and not 2;

$i = 1$  and  $j = 3$  satisfies both conditions;

$i = 2$  and  $j = 3$  fail because  $3 + 2 = 5$  and not 3, also  $3 \cdot 2 = 6$  and not 2;

Problem: [link](#)

Solution: [link](#)

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  using i64 = long long;
4  i64 sqrt(i64 n) {
5      i64 lo = 0, hi = 2 * sqrt(n);
6      while (lo < hi) {
7          i64 x = (lo + hi) / 2;
8          if (x * x >= n) {
9              hi = x;
10         } else {
```

```

11         lo = x + 1;
12     }
13 }
14 return lo;
15 }
16 void solve() {
17     int n;
18     cin >> n;
19     map<i64, int> cnt;
20     for (int i = 0; i < n; i++) {
21         int a;
22         cin >> a;
23         cnt[a]++;
24     }
25     int q;
26     cin >> q;
27     for (int i = 0; i < q; i++) {
28         i64 x, y;
29         cin >> x >> y;
30         if (x * x - 4 * y < 0) {
31             cout << 0 << " ";
32             continue;
33         }
34         i64 s = sqrt(x * x - 4 * y);
35         i64 a = (x + s) / 2;
36         i64 b = (x - s) / 2;
37         if (a + b != x || a * b != y) {
38             cout << 0 << " ";
39             continue;
40         }
41         if (a == b) {
42             int c = cnt[a];
43             cout << 1LL * c * (c - 1) / 2 << " ";
44         } else {
45             cout << 1LL * cnt[a] * cnt[b] << " ";
46         }
47     }
48     cout << "\n";
49 }
50 int main() {
51     ios::sync_with_stdio(false);
52     cin.tie(nullptr);
53     int t;
54     cin >> t;
55     while (t--) {
56         solve();
57     }
58     return 0;
59 }

```

## 268: Power of Points

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given  $n$  points with integer coordinates  $x_1, \dots, x_n$ , which lie on a number line.

For some integer  $s$ , we construct segments  $[s, x_1], [s, x_2], \dots, [s, x_n]$ . Note that if  $x_i < s$ , then the segment will look like  $[x_i, s]$ . The segment  $[a, b]$  covers all integer points  $a, a + 1, a + 2, \dots, b$ .

We define the power of a point  $p$  as the number of segments that intersect the point with coordinate  $p$ , denoted as  $f_p$ .

Your task is to compute  $\sum_{p=1}^{10^9} f_p$  for each  $s \in \{x_1, \dots, x_n\}$ , i.e., the sum of  $f_p$  for all integer points from 1 to  $10^9$ .

For example, if the initial coordinates are  $[1, 2, 5, 7, 1]$  and we choose  $s = 5$ , then the segments will be:  $[1, 5], [2, 5], [5, 5], [5, 7], [1, 5]$ . And the powers of the points will be:  $f_1 = 2, f_2 = 3, f_3 = 3, f_4 = 3, f_5 = 5, f_6 = 1, f_7 = 1, f_8 = 0, \dots, f_{10^9} = 0$ . Their sum is  $2 + 3 + 3 + 3 + 5 + 1 + 1 = 18$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> x(n);
8     for (int i = 0; i < n; i++) {
9         cin >> x[i];
10    }
11    vector<int> p(n);
12    iota(p.begin(), p.end(), 0);
13    sort(p.begin(), p.end(),
14        [&](int i, int j) {
15            return x[i] < x[j];
16        });
17    i64 res = accumulate(x.begin(), x.end(), 0LL) - 1LL * x[p[0]] * n + n;
18    vector<i64> ans(n);
19    for (int i = 0; i < n; i++) {
20        if (i > 0) {
21            res += 1LL * (x[p[i]] - x[p[i - 1]]) * (i - (n - i));
22        }
23        ans[p[i]] = res;
24    }
25    for (int i = 0; i < n; i++) {
26        cout << ans[i] << " \n"[i == n - 1];
27    }
28 }
29 int main() {
30     ios::sync_with_stdio(false);
31     cin.tie(nullptr);
32     int t;
33     cin >> t;
34     while (t--) {
35         solve();
36     }
37     return 0;
38 }
```

## 269: Strong Vertices

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Given two arrays  $a$  and  $b$ , both of length  $n$ . Elements of both arrays indexed from 1 to  $n$ . You are constructing a directed graph, where edge from  $u$  to  $v$  ( $u \neq v$ ) exists if  $a_u - a_v \geq b_u - b_v$ .

A vertex  $V$  is called strong if there exists a path from  $V$  to all other vertices.

A path in a directed graph is a chain of several vertices, connected by edges, such that moving from the vertex  $u$ , along the directions of the edges, the vertex  $v$  can be reached.

Your task is to find all strong vertices.

For example, if  $a = [3, 1, 2, 4]$  and  $b = [4, 3, 2, 1]$ , the graph will look like this:

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    for (int i = 0; i < n; i++) {
12        int b;
13        cin >> b;
14        a[i] -= b;
15    }
16    int x = *max_element(a.begin(), a.end());
17    vector<int> p;
18    for (int i = 0; i < n; i++) {
19        if (a[i] == x) {
20            p.push_back(i);
21        }
22    }
23    cout << p.size() << "\n";
24    for (int i = 0; i < p.size(); i++) {
25        cout << p[i] + 1 << " \n"[i == p.size() - 1];
26    }
27 }
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(nullptr);
31     int t;
32     cin >> t;
33     while (t--) {
34         solve();

```

```

35     }
36     return 0;
37 }
```

## 270: Dual (Easy Version)

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The only difference between the two versions of this problem is the constraint on the maximum number of operations. You can make hacks only if all versions of the problem are solved.

You are given an array  $a_1, a_2, \dots, a_n$  of integers (positive, negative or 0). You can perform multiple operations on the array (possibly 0 operations).

In one operation, you choose  $i, j$  ( $1 \leq i, j \leq n$ , they can be equal) and set  $a_i := a_i + a_j$  (i.e., add  $a_j$  to  $a_i$ ).

Make the array non-decreasing (i.e.,  $a_i \leq a_{i+1}$  for  $1 \leq i \leq n - 1$ ) in at most 50 operations. You do not need to minimize the number of operations.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 vector<pair<int, int>> work(vector<int> a) {
5     int n = a.size();
6     auto [mn, mx] = minmax_element(a.begin(), a.end());
7     if (*mx + *mn < 0) {
8         for (auto &x : a) {
9             x = -x;
10        }
11        reverse(a.begin(), a.end());
12        auto ans = work(a);
13        for (auto &[x, y] : ans) {
14            x = n - 1 - x;
15            y = n - 1 - y;
16        }
17        return ans;
18    }
19    vector<pair<int, int>> ans;
20    int k = mx - a.begin();
21    int cntneg = 0;
22    for (int i = 0; i < n; i++) {
23        if (a[i] < 0) {
24            cntneg++;
25        }
26    }
```

```

27     if (cntneg <= 12) {
28         for (int i = 0; i < n; i++) {
29             if (a[i] < 0) {
30                 a[i] += a[k];
31                 ans.emplace_back(i, k);
32             }
33         }
34         for (int i = 1; i < n; i++) {
35             a[i] += a[i - 1];
36             ans.emplace_back(i, i - 1);
37         }
38     }
39     return ans;
40 }
41 int x = mn - a.begin();
42 for (int i = 0; i < 5; i++) {
43     a[x] += a[x];
44     ans.emplace_back(x, x);
45 }
46 for (int i = 0; i < n; i++) {
47     if (a[i] > 0) {
48         a[i] += a[x];
49         ans.emplace_back(i, x);
50     }
51     for (int i = n - 2; i >= 0; i--) {
52         a[i] += a[i + 1];
53         ans.emplace_back(i, i + 1);
54     }
55 }
56 }
57 void solve() {
58     int n;
59     cin >> n;
60     vector<int> a(n);
61     for (int i = 0; i < n; i++) {
62         cin >> a[i];
63     }
64     auto ans = work(a);
65     cout << ans.size() << "\n";
66     for (auto [x, y] : ans) {
67         cout << x + 1 << " " << y + 1 << "\n";
68     }
69 }
70 int main() {
71     ios::sync_with_stdio(false);
72     cin.tie(nullptr);
73     int t;
74     cin >> t;
75     while (t--) {
76         solve();
77     }
78     return 0;
79 }
```

**misc****271: Greetings**

- Time limit: 5 seconds

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  people on the number line; the  $i$ -th person is at point  $a_i$  and wants to go to point  $b_i$ . For each person,  $a_i < b_i$ , and the starting and ending points of all people are distinct. (That is, all of the  $2n$  numbers  $a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n$  are distinct.)

All the people will start moving simultaneously at a speed of 1 unit per second until they reach their final point  $b_i$ . When two people meet at the same point, they will greet each other once. How many greetings will there be?

Note that a person can still greet other people even if they have reached their final point.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template <typename T>
5 # struct Fenwick;
6 void solve() {
7     int n;
8     cin >> n;
9     vector<int> a(n), b(n), v;
10    v.reserve(2 * n);
11    for (int i = 0; i < n; i++) {
12        cin >> a[i] >> b[i];
13        v.push_back(a[i]);
14        v.push_back(b[i]);
15    }
16    sort(v.begin(), v.end());
17    for (int i = 0; i < n; i++) {
18        a[i] = lower_bound(v.begin(), v.end(), a[i]) - v.begin();
19        b[i] = lower_bound(v.begin(), v.end(), b[i]) - v.begin();
20    }
21    vector<int> f(2 * n, -1);
22    for (int i = 0; i < n; i++) {
23        f[a[i]] = b[i];
24    }
25    Fenwick<int> fen(2 * n);
26    i64 ans = 0;
27    for (int i = 2 * n - 1; i >= 0; i--) {
28        if (f[i] != -1) {
29            ans += fen.sum(f[i]);
30            fen.add(f[i], 1);
31        }
32    }
33    cout << ans << "\n";
34 }
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int t;
39     cin >> t;
40     while (t--) {

```

```

41         solve();
42     }
43     return 0;
44 }
```

## 272: Effects of Anti Pimples

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Chaneka has an array  $[a_1, a_2, \dots, a_n]$ . Initially, all elements are white. Chaneka will choose one or more different indices and colour the elements at those chosen indices black. Then, she will choose all white elements whose indices are multiples of the index of at least one black element and colour those elements green. After that, her score is the maximum value of  $a_i$  out of all black and green elements.

There are  $2^n - 1$  ways for Chaneka to choose the black indices. Find the sum of scores for all possible ways Chaneka can choose the black indices. Since the answer can be very big, print the answer modulo 998 244 353.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
```

```

29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int n;
38     cin >> n;
39     vector<int> a(n + 1);
40     for (int i = 1; i <= n; i++) {
41         cin >> a[i];
42     }
43     for (int i = 1; i <= n; i++) {
44         for (int j = 2 * i; j <= n; j += i) {
45             a[i] = max(a[i], a[j]);
46         }
47     }
48     sort(a.begin() + 1, a.end());
49     Z ans = 0;
50     Z p = 1;
51     for (int i = 1; i <= n; i++) {
52         ans += p * a[i];
53         p *= 2;
54     }
55     cout << ans << "\n";
56     return 0;
57 }
```

### 273: Rumor

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vova promised himself that he would never play computer games... But recently Firestorm - a well-known game developing company - published their newest game, World of Farcraft, and it became really popular. Of course, Vova started playing it.

Now he tries to solve a quest. The task is to come to a settlement named Overcity and spread a rumor in it.

Vova knows that there are  $n$  characters in Overcity. Some characters are friends to each other, and they share information they got. Also Vova knows that he can bribe each character so he or she starts spreading the rumor;  $i$ -th character wants  $c_i$  gold in exchange for spreading the rumor. When a character hears the rumor, he tells it to all his friends, and they start spreading the rumor to their friends (for free), and so on.

The quest is finished when all  $n$  characters know the rumor. What is the minimum amount of gold Vova needs to spend in order to finish the quest?

Take a look at the notes if you think you haven't understood the problem completely.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<int> c(n);
10    for (int i = 0; i < n; i++) {
11        cin >> c[i];
12    }
13    vector<vector<int>> adj(n);
14    for (int i = 0; i < m; i++) {
15        int u, v;
16        cin >> u >> v;
17        u--;
18        v--;
19        adj[u].push_back(v);
20        adj[v].push_back(u);
21    }
22    vector<bool> vis(n);
23    i64 ans = 0;
24    for (int i = 0; i < n; i++) {
25        if (vis[i]) {
26            continue;
27        }
28        queue<int> q;
29        q.push(i);
30        vis[i] = true;
31        int minc = c[i];
32        while (!q.empty()) {
33            int x = q.front();
34            q.pop();
35            minc = min(minc, c[x]);
36            for (auto y : adj[x]) {
37                if (!vis[y]) {
38                    vis[y] = true;
39                    q.push(y);
40                }
41            }
42            ans += minc;
43        }
44        cout << ans << "\n";
45    }
46 }
```

## 274: Hometask

- Time limit: 2 seconds
- Memory limit: 256 megabytes

- Input file: standard input
- Output file: standard output

Sergey attends lessons of the N-ish language. Each lesson he receives a hometask. This time the task is to translate some sentence to the N-ish language. Sentences of the N-ish language can be represented as strings consisting of lowercase Latin letters without spaces or punctuation marks.

Sergey totally forgot about the task until half an hour before the next lesson and hastily scribbled something down. But then he recollectored that in the last lesson he learned the grammar of N-ish. The spelling rules state that N-ish contains some “forbidden” pairs of letters: such letters can never occur in a sentence next to each other. Also, the order of the letters doesn’t matter (for example, if the pair of letters “ab” is forbidden, then any occurrences of substrings “ab” and “ba” are also forbidden). Also, each pair has different letters and each letter occurs in no more than one forbidden pair.

Now Sergey wants to correct his sentence so that it doesn’t contain any “forbidden” pairs of letters that stand next to each other. However, he is running out of time, so he decided to simply cross out some letters from the sentence. What smallest number of letters will he have to cross out? When a letter is crossed out, it is “removed” so that the letters to its left and right (if they existed), become neighboring. For example, if we cross out the first letter from the string “aba”, we get the string “ba”, and if we cross out the second letter, we get “aa”.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     string s;
8     cin >> s;
9     int k;
10    cin >> k;
11    int n = s.size();
12    vector<int> mx(26);
13    bool forb[26][26] {};
14    for (int i = 0; i < k; i++) {
15        char x, y;
16        cin >> x >> y;
17        forb[x - 'a'][y - 'a'] = true;
18        forb[y - 'a'][x - 'a'] = true;
19    }
20    for (int i = 0; i < n; i++) {
21        int f = 0;
22        for (int j = 0; j < 26; j++) {
23            if (!forb[s[i] - 'a'][j]) {
24                f = max(f, mx[j] + 1);
25            }
26        }
}

```

```

27         mx[s[i] - 'a'] = f;
28     }
29     auto ans = n - *max_element(mx.begin(), mx.end());
30     cout << ans << "\n";
31     return 0;
32 }
```

## 275: Wooden Toy Festival

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In a small town, there is a workshop specializing in woodwork. Since the town is small, only three carvers work there.

Soon, a wooden toy festival is planned in the town. The workshop employees want to prepare for it.

They know that  $n$  people will come to the workshop with a request to make a wooden toy. People are different and may want different toys. For simplicity, let's denote the pattern of the toy that the  $i$ -th person wants as  $a_i$  ( $1 \leq a_i \leq 10^9$ ).

Each of the carvers can choose an integer pattern  $x$  ( $1 \leq x \leq 10^9$ ) in advance, different carvers can choose different patterns.  $x$  is the integer. During the preparation for the festival, the carvers will perfectly work out the technique of making the toy of the chosen pattern, which will allow them to cut it out of wood instantly. To make a toy of pattern  $y$  for a carver who has chosen pattern  $x$ , it will take  $|x - y|$  time, because the more the toy resembles the one he can make instantly, the faster the carver will cope with the work.

On the day of the festival, when the next person comes to the workshop with a request to make a wooden toy, the carvers can choose who will take on the job. At the same time, the carvers are very skilled people and can work on orders for different people simultaneously.

Since people don't like to wait, the carvers want to choose patterns for preparation in such a way that the maximum waiting time over all people is as small as possible.

Output the best maximum waiting time that the carvers can achieve.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
```

```

3  using i64 = long long;
4  void solve() {
5      int n;
6      cin >> n;
7      vector<int> a(n);
8      for (int i = 0; i < n; i++) {
9          cin >> a[i];
10     }
11     sort(a.begin(), a.end());
12     int lo = 0, hi = 1E9;
13     while (lo < hi) {
14         int x = (lo + hi) / 2;
15         int i = 0;
16         for (int t = 0; t < 3 && i < n; t++) {
17             i = upper_bound(a.begin(), a.end(), 2LL * x + a[i]) - a.begin();
18         }
19         if (i == n) {
20             hi = x;
21         } else {
22             lo = x + 1;
23         }
24     }
25     cout << lo << "\n";
26 }
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(nullptr);
30     int t;
31     cin >> t;
32     while (t--) {
33         solve();
34     }
35     return 0;
36 }
```

## 276: Round Dance

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

$n$  people came to the festival and decided to dance a few round dances. There are at least 2 people in the round dance and each person has exactly two neighbors. If there are 2 people in the round dance then they have the same neighbor on each side.

You decided to find out exactly how many dances there were. But each participant of the holiday remembered exactly one neighbor. Your task is to determine what the minimum and maximum number of round dances could be.

For example, if there were 6 people at the holiday, and the numbers of the neighbors they remembered are equal [2, 1, 4, 3, 6, 5], then the minimum number of round dances is 1:

1 – 2 – 3 – 4 – 5 – 6 – 1

1 – 2 – 1

3 – 4 – 3

5 – 6 – 5

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 void solve() {
6     int n;
7     cin >> n;
8     vector<int> a(n);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11         a[i]--;
12     }
13     int min = 0, max = 0;
14     bool chain = false;
15     DSU dsu(n);
16     for (int i = 0; i < n; i++) {
17         dsu.merge(i, a[i]);
18     }
19     vector<int> e(n);
20     for (int i = 0; i < n; i++) {
21         if (a[a[i]] == i) {
22             e[dsu.find(i)] = 1;
23             chain = true;
24         }
25     }
26     for (int i = 0; i < n; i++) {
27         if (dsu.find(i) == i) {
28             if (!e[i]) {
29                 min++;
30             }
31             max++;
32         }
33     }
34     min += chain;
35     cout << min << " " << max << "\n";
36 }
37 int main() {
38     ios::sync_with_stdio(false);
39     cin.tie(nullptr);
40     int t;
41     cin >> t;
42     while (t--) {
43         solve();
44     }
45     return 0;
46 }
```

## 277: Dreaming of Freedom

- Time limit: 2.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  programmers choosing their favorite algorithm amongst  $m$  different choice options. Before the first round, all  $m$  options are available. In each round, every programmer makes a vote for one of the remaining algorithms. After the round, only the algorithms with the maximum number of votes remain. The voting process ends when there is only one option left. Determine whether the voting process can continue indefinitely or no matter how people vote, they will eventually choose a single option after some finite amount of rounds?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 vector<int> minp, primes;
5 void sieve(int n) {
6     minp.assign(n + 1, 0);
7     primes.clear();
8     for (int i = 2; i <= n; i++) {
9         if (minp[i] == 0) {
10             minp[i] = i;
11             primes.push_back(i);
12         }
13         for (auto p : primes) {
14             if (i * p > n) {
15                 break;
16             }
17             minp[i * p] = p;
18             if (p == minp[i]) {
19                 break;
20             }
21         }
22     }
23 }
24 void solve() {
25     int n, m;
26     cin >> n >> m;
27     if (n > 1 && minp[n] <= m) {
28         cout << "NO\n";
29     } else {
30         cout << "YES\n";
31     }
32 }
33 int main() {
34     ios::sync_with_stdio(false);
35     cin.tie(nullptr);
36     sieve(1E6);

```

```

37     int t;
38     cin >> t;
39     while (t--) {
40         solve();
41     }
42     return 0;
43 }
```

**278: Forever Winter**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A snowflake graph is generated from two integers  $x$  and  $y$ , both greater than 1, as follows:

Start with one central vertex.

Connect  $x$  new vertices to this central vertex.

Connect  $y$  new vertices to each of these  $x$  vertices.

The snowflake graph above has a central vertex 15, then  $x = 5$  vertices attached to it (3, 6, 7, 8, and 20), and then  $y = 3$  vertices attached to each of those.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     int xy = 0;
8     vector<int> deg(n);
9     for (int i = 0; i < m; i++) {
10         int u, v;
11         cin >> u >> v;
12         u--, v--;
13         deg[u]++;
14         deg[v]++;
15     }
16     xy = count(deg.begin(), deg.end(), 1);
17     int x = n - 1 - xy;
18     int y = xy / x;
19     cout << x << " " << y << "\n";
20 }
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     int t;
25     cin >> t;
```

```

26     while (t--) {
27         solve();
28     }
29     return 0;
30 }
```

**279: LCM Challenge**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Some days ago, I learned the concept of LCM (least common multiple). I've played with it for several times and I want to make a big number with it.

But I also don't want to use many numbers, so I'll choose three positive integers (they don't have to be distinct) which are not greater than n. Can you help me to find the maximum possible least common multiple of these three integers?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     if (n <= 2) {
10         cout << n << "\n";
11     } else if (n % 2) {
12         cout << 1LL * n * (n - 1) * (n - 2) << "\n";
13     } else if (n % 3) {
14         cout << 1LL * n * (n - 1) * (n - 3) << "\n";
15     } else {
16         cout << 1LL * (n - 1) * (n - 2) * (n - 3) << "\n";
17     }
18     return 0;
19 }
```

**280: Almost Increasing Subsequence**

- Time limit: 2 seconds
- Memory limit: 256 megabytes

- Input file: standard input
- Output file: standard output

A sequence is almost-increasing if it does not contain three consecutive elements  $x, y, z$  such that  $x \geq y \geq z$ .

You are given an array  $a_1, a_2, \dots, a_n$  and  $q$  queries.

Each query consists of two integers  $1 \leq l \leq r \leq n$ . For each query, find the length of the longest almost-increasing subsequence of the subarray  $a_l, a_{l+1}, \dots, a_r$ .

A subsequence is a sequence that can be derived from the given sequence by deleting zero or more elements without changing the order of the remaining elements.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, q;
8     cin >> n >> q;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    vector<int> s(n), f(n);
14    for (int i = 1; i < n; i++) {
15        s[i] = s[i - 1] + (a[i] > a[i - 1]);
16        f[i] = f[i - 1] + (i >= 2 && a[i - 1] > a[i - 2] && a[i - 1] >= a[i]);
17    }
18    while (q--) {
19        int l, r;
20        cin >> l >> r;
21        l--, r--;
22        int ans = 1 + f[r] - f[l] + s[r] - s[l];
23        if (l < r && s[l] >= s[l + 1] && (l == 0 || s[l] <= s[l - 1])) {
24            ans++;
25        }
26        cout << ans << "\n";
27    }
28    return 0;
29 }
```

## 281: Strongly Composite

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A prime number is an integer greater than 1, which has exactly two divisors. For example, 7 is a prime, since it has two divisors {1, 7}. A composite number is an integer greater than 1, which has more than two different divisors.

Note that the integer 1 is neither prime nor composite.

Let's look at some composite number  $v$ . It has several divisors: some divisors are prime, others are composite themselves. If the number of prime divisors of  $v$  is less or equal to the number of composite divisors, let's name  $v$  as strongly composite.

For example, number 12 has 6 divisors: {1, 2, 3, 4, 6, 12}, two divisors 2 and 3 are prime, while three divisors 4, 6 and 12 are composite. So, 12 is strongly composite. Other examples of strongly composite numbers are 4, 8, 9, 16 and so on.

On the other side, divisors of 15 are {1, 3, 5, 15}: 3 and 5 are prime, 15 is composite. So, 15 is not a strongly composite. Other examples are: 2, 3, 5, 6, 7, 10 and so on.

You are given  $n$  integers  $a_1, a_2, \dots, a_n$  ( $a_i > 1$ ). You have to build an array  $b_1, b_2, \dots, b_k$  such that following conditions are satisfied:

Product of all elements of array  $a$  is equal to product of all elements of array  $b$ :  $a_1 \cdot a_2 \cdot \dots \cdot a_n = b_1 \cdot b_2 \cdot \dots \cdot b_k$ ;

All elements of array  $b$  are integers greater than 1 and strongly composite;

The size  $k$  of array  $b$  is the maximum possible.

Find the size  $k$  of array  $b$ , or report, that there is no array  $b$  satisfying the conditions.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 vector<int> minp, primes;
5 void sieve(int n) {
6     minp.assign(n + 1, 0);
7     primes.clear();
8     for (int i = 2; i <= n; i++) {
9         if (minp[i] == 0) {
10             minp[i] = i;
11             primes.push_back(i);
12         }
13         for (auto p : primes) {
14             if (i * p > n) {
15                 break;
16             }
17             minp[i * p] = p;
18             if (p == minp[i]) {
19                 break;
20             }
21         }
22     }

```

```

22     }
23 }
24 void solve() {
25     int n;
26     cin >> n;
27     vector<int> a(n);
28     for (int i = 0; i < n; i++) {
29         cin >> a[i];
30     }
31     int ans = 0;
32     set<int> s;
33     for (int i = 0; i < n; i++) {
34         int x = a[i];
35         while (x > 1) {
36             int p = minp[x];
37             x /= p;
38             if (s.count(p)) {
39                 ans++;
40                 s.erase(p);
41             } else {
42                 s.insert(p);
43             }
44         }
45     }
46     ans += s.size() / 3;
47     cout << ans << "\n";
48 }
49 int main() {
50     ios::sync_with_stdio(false);
51     cin.tie(nullptr);
52     sieve(1E7);
53     int t;
54     cin >> t;
55     while (t--) {
56         solve();
57     }
58     return 0;
59 }
```

## 282: Petya and Inequations

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Little Petya loves inequations. Help him find  $n$  positive integers  $a_1, a_2, \dots, a_n$ , such that the following two conditions are satisfied:

$$a_{12} + a_{22} + \dots + a_{n2} \geq x$$

$$a_1 + a_2 + \dots + a_n \leq y$$

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     i64 x, y;
9     cin >> n >> x >> y;
10    if (n > y || n - 1 + (y - n + 1) * (y - n + 1) < x) {
11        cout << -1 << "\n";
12        return 0;
13    }
14    for (int i = 0; i < n; i++) {
15        cout << (i ? 1 : y - n + 1) << "\n";
16    }
17    return 0;
18 }
```

## 283: Painting Eggs

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The Bitlandians are quite weird people. They have very peculiar customs.

As is customary, Uncle J. wants to have  $n$  eggs painted for Bitruz (an ancient Bitland festival). He has asked G. and A. to do the work.

The kids are excited because just as is customary, they're going to be paid for the job!

Overall uncle J. has got  $n$  eggs. G. named his price for painting each egg. Similarly, A. named his price for painting each egg. It turns out that for each egg the sum of the money both A. and G. want for the painting equals 1000.

Uncle J. wants to distribute the eggs between the children so as to give each egg to exactly one child. Also, Uncle J. wants the total money paid to A. to be different from the total money paid to G. by no more than 500.

Help Uncle J. Find the required distribution of eggs or otherwise say that distributing the eggs in the required manner is impossible.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
```

```

4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n), b(n);
10    int s = 0;
11    for (int i = 0; i < n; i++) {
12        cin >> a[i] >> b[i];
13        if (s + a[i] <= 500) {
14            s += a[i];
15            cout << 'A';
16        } else {
17            s -= b[i];
18            cout << 'G';
19        }
20    }
21    cout << '\n';
22    return 0;
23 }
```

## 284: Hamsters and Tigers

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Today there is going to be an unusual performance at the circus - hamsters and tigers will perform together! All of them stand in circle along the arena edge and now the trainer faces a difficult task: he wants to swap the animals' positions so that all the hamsters stood together and all the tigers also stood together. The trainer swaps the animals in pairs not to create a mess. He orders two animals to step out of the circle and swap places. As hamsters feel highly uncomfortable when tigers are nearby as well as tigers get nervous when there's so much potential prey around (consisting not only of hamsters but also of yummier spectators), the trainer wants to spend as little time as possible moving the animals, i.e. he wants to achieve it with the minimal number of swaps. Your task is to help him.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     string s;
```

```

10     cin >> s;
11     int c = count(s.begin(), s.end(), 'H');
12     int ans = n;
13     for (int i = 0; i < n; i++) {
14         int cnt = count(s.begin(), s.begin() + c, 'T');
15         ans = min(ans, cnt);
16         rotate(s.begin(), s.begin() + 1, s.end());
17     }
18     cout << ans << "\n";
19     return 0;
20 }
```

## 285: Lucky Tickets

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vasya thinks that lucky tickets are the tickets whose numbers are divisible by 3. He gathered quite a large collection of such tickets but one day his younger brother Leonid was having a sulk and decided to destroy the collection. First he tore every ticket exactly in two, but he didn't think it was enough and Leonid also threw part of the pieces away. Having seen this, Vasya got terrified but still tried to restore the collection. He chose several piece pairs and glued each pair together so that each pair formed a lucky ticket. The rest of the pieces Vasya threw away reluctantly. Thus, after the gluing of the  $2t$  pieces he ended up with  $t$  tickets, each of which was lucky.

When Leonid tore the tickets in two pieces, one piece contained the first several letters of his number and the second piece contained the rest.

Vasya can glue every pair of pieces in any way he likes, but it is important that he gets a lucky ticket in the end. For example, pieces 123 and 99 can be glued in two ways: 12399 and 99123.

What maximum number of tickets could Vasya get after that?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     int cnt[3] {};
```

```

10     for (int i = 0; i < n; i++) {
11         int x;
12         cin >> x;
13         cnt[x % 3]++;
14     }
15     int ans = min(cnt[1], cnt[2]) + cnt[0] / 2;
16     cout << ans << "\n";
17     return 0;
18 }
```

**286: Email address**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Sometimes one has to spell email addresses over the phone. Then one usually pronounces a dot as dot, an at sign as at. As a result, we get something like vasyaatgmaildotcom. Your task is to transform it into a proper email address ([email protected]).

It is known that a proper email address contains only such symbols as . @ and lower-case Latin letters, doesn't start with and doesn't end with a dot. Also, a proper email address doesn't start with and doesn't end with an at sign. Moreover, an email address contains exactly one such symbol as @, yet may contain any number (possible, zero) of dots.

You have to carry out a series of replacements so that the length of the result was as short as possible and it was a proper email address. If the lengths are equal, you should print the lexicographically minimal result.

Overall, two variants of replacement are possible: dot can be replaced by a dot, at can be replaced by an at.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     string s;
8     cin >> s;
9     string ans;
10    int n = s.size();
11    for (int i = 1; i+2 < n; i++) {
```

```

12         if (s.substr(i, 2) == "at") {
13             string t;
14             for (int j = 0; j < i; ) {
15                 if (j && j+3 <= i && s.substr(j, 3) == "dot") {
16                     t += '.';
17                     j += 3;
18                 } else {
19                     t += s[j];
20                     j++;
21                 }
22             }
23             t += '@';
24             for (int j = i+2; j < n; ) {
25                 if (j+3 < n && s.substr(j, 3) == "dot") {
26                     t += '.';
27                     j += 3;
28                 } else {
29                     t += s[j];
30                     j++;
31                 }
32             }
33             if (ans.empty() || t.size() < ans.size() || (t.size() == ans.size() && t < ans))
34                 ans = t;
35             }
36         }
37     }
38     cout << ans << "\n";
39     return 0;
40 }
```

## 287: Making Anti-Palindromes

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a string  $s$ , consisting of lowercase English letters. In one operation, you are allowed to swap any two characters of the string  $s$ .

A string  $s$  of length  $n$  is called an anti-palindrome, if  $s[i] \neq s[n - i + 1]$  for every  $i$  ( $1 \leq i \leq n$ ). For example, the strings “codeforces”, “string” are anti-palindromes, but the strings “abacaba”, “abc”, “test” are not.

Determine the minimum number of operations required to make the string  $s$  an anti-palindrome, or output  $-1$ , if this is not possible.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     string s;
8     cin >> s;
9     array<int, 26> cnt{};
10    for (int i = 0; i < n; i++) {
11        cnt[s[i] - 'a']++;
12    }
13    if (n % 2 || *max_element(cnt.begin(), cnt.end()) > n/2) {
14        cout << -1 << "\n";
15        return;
16    }
17    cnt = {};
18    int tot = 0;
19    for (int i = 0; i < n/2; i++) {
20        if (s[i] == s[n-1-i]) {
21            cnt[s[i] - 'a']++;
22            tot++;
23        }
24    }
25    int ans = max((tot+1) / 2, *max_element(cnt.begin(), cnt.end()));
26    cout << ans << "\n";
27 }
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(nullptr);
31     int t;
32     cin >> t;
33     while (t--) {
34         solve();
35     }
36     return 0;
37 }
```

## 288: Replace To Make Regular Bracket Sequence

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given string  $s$  consists of opening and closing brackets of four kinds  $<>$ ,  $\{$ ,  $\}$ ,  $[$ ,  $]$ . There are two types of brackets: opening and closing. You can replace any bracket by another of the same type. For example, you can replace  $<$  by the bracket  $\{$ , but you can't replace it by  $)$  or  $>$ .

The following definition of a regular bracket sequence is well-known, so you can be familiar with it.

Let's define a regular bracket sequence (RBS). Empty string is RBS. Let  $s_1$  and  $s_2$  be a RBS then the strings  $s_2$ ,  $\{s_1\}s_2$ ,  $[s_1]s_2$ ,  $(s_1)s_2$  are also RBS.

For example the string “[([{}])<>” is RBS, but the strings “[()” and ”][()” are not.

Determine the least number of replaces to make the string s RBS.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     map<char, int> id;
8     id['('] = 1;
9     id[')'] = -1;
10    id['['] = 2;
11    id[']'] = -2;
12    id['{'] = 3;
13    id['}'] = -3;
14    id['<'] = 4;
15    id['>'] = -4;
16    string s;
17    cin >> s;
18    int n = s.size();
19    int ans = 0;
20    vector<int> stk;
21    for (auto c : s) {
22        if (id[c] > 0) {
23            stk.push_back(id[c]);
24        } else if (stk.empty()) {
25            cout << "Impossible" << "\n";
26            return 0;
27        } else {
28            if (stk.back() != -id[c]) {
29                ans += 1;
30            }
31            stk.pop_back();
32        }
33    }
34    if (!stk.empty()) {
35        cout << "Impossible" << "\n";
36        return 0;
37    }
38    cout << ans << "\n";
39    return 0;
40 }
```

## 289: Load Balancing

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In the school computer room there are  $n$  servers which are responsible for processing several computing tasks. You know the number of scheduled tasks for each server: there are  $m_i$  tasks assigned to the  $i$ -th

server.

In order to balance the load for each server, you want to reassign some tasks to make the difference between the most loaded server and the least loaded server as small as possible. In other words you want to minimize expression  $ma - mb$ , where  $a$  is the most loaded server and  $b$  is the least loaded one.

In one second you can reassign a single task. Thus in one second you can choose any pair of servers and move a single task from one server to another.

Write a program to find the minimum number of seconds needed to balance the load of servers.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> m(n);
10    for (int i = 0; i < n; i++) {
11        cin >> m[i];
12    }
13    int sum = accumulate(m.begin(), m.end(), 0);
14    int lo = sum / n;
15    int hi = (sum+n-1) / n;
16    int l = 0, r = 0;
17    for (int i = 0; i < n; i++) {
18        if (m[i] < lo) {
19            l += lo - m[i];
20        }
21        if (m[i] > hi) {
22            r += m[i] - hi;
23        }
24    }
25    int ans = max(l, r);
26    cout << ans << "\n";
27    return 0;
28 }
```

## 290: Queries about less or equal elements

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given two arrays of integers  $a$  and  $b$ . For each element of the second array  $b_j$  you should find the number of elements in array  $a$  that are less than or equal to the value  $b_j$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<int> a(n), b(m);
10    for (auto &x : a) {
11        cin >> x;
12    }
13    for (auto &x : b) {
14        cin >> x;
15    }
16    sort(a.begin(), a.end());
17    for (int i = 0; i < m; i++) {
18        cout << upper_bound(a.begin(), a.end(), b[i]) - a.begin() << " \n"[i == m-1];
19    }
20    return 0;
21 }
```

## 291: Extract Numbers

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given string  $s$ . Let's call word any largest sequence of consecutive symbols without symbols ‘,’ (comma) and ‘;’ (semicolon). For example, there are four words in string “aba,123;1a;0”: “aba”, “123”, “1a”, “0”. A word can be empty: for example, the string  $s=“;;”$  contains three empty words separated by ‘;’.

You should find all words in the given string that are nonnegative INTEGER numbers without leading zeroes and build by them new string  $a$ . String  $a$  should contain all words that are numbers separating them by ‘,’ (the order of numbers should remain the same as in the string  $s$ ). By all other words you should build string  $b$  in the same way (the order of numbers should remain the same as in the string  $s$ ).

Here strings “101”, “0” are INTEGER numbers, but “01” and “1.0” are not.

For example, for the string aba,123;1a;0 the string  $a$  would be equal to “123,0” and string  $b$  would be equal to “aba,1a”.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     string s;
8     cin >> s;
9     int n = s.size();
10    vector<string> words;
11    for (int i = 0, j = -1; i <= n; i++) {
12        if (i == n || s[i] == ',' || s[i] == ';') {
13            words.push_back(s.substr(j+1, i-j-1));
14            j = i;
15        }
16    }
17    vector<string> ints, oths;
18    for (auto s : words) {
19        if (all_of(s.begin(), s.end(), isdigit) && !s.empty() && (s[0] != '0' || s.size()
20            == 1)) {
21            ints.push_back(s);
22        } else {
23            oths.push_back(s);
24        }
25    }
26    for (auto v : {ints, oths}) {
27        if (v.empty()) {
28            cout << "-\n";
29        } else {
30            cout << "\n";
31            for (int i = 0; i < v.size(); i++) {
32                cout << v[i] << ",\n"[i == v.size()-1];
33            }
34        }
35    }
36    return 0;
37 }
```

## 292: Queries on a String

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a string  $s$  and should process  $m$  queries. Each query is described by two 1-based indices  $l_i, r_i$  and integer  $k_i$ . It means that you should cyclically shift the substring  $s[l_i \dots r_i]$   $k_i$  times. The queries should be processed one after another in the order they are given.

One operation of a cyclic shift (rotation) is equivalent to moving the last character to the position of the first character and shifting all other characters one position to the right.

For example, if the string  $s$  is abacaba and the query is  $l_1 = 3, r_1 = 6, k_1 = 1$  then the answer is abbacaa. If after that we would process the query  $l_2 = 1, r_2 = 4, k_2 = 2$  then we would get the string baabcaa.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     string s;
8     cin >> s;
9     int m;
10    cin >> m;
11    for (int i = 0; i < m; i++) {
12        int l, r, k;
13        cin >> l >> r >> k;
14        l--;
15        int len = r - l;
16        k %= len;
17        rotate(s.begin() + l, s.begin() + r - k, s.begin() + r);
18    }
19    cout << s << "\n";
20    return 0;
21 }
```

### 293: Cola

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

To celebrate the opening of the Winter Computer School the organizers decided to buy  $n$  liters of cola. However, an unexpected difficulty occurred in the shop: it turned out that cola is sold in bottles 0.5, 1 and 2 liters in volume. At that, there are exactly  $a$  bottles 0.5 in volume,  $b$  one-liter bottles and  $c$  of two-liter ones. The organizers have enough money to buy any amount of cola. What caused the heated arguments was how many bottles of every kind to buy, as this question is pivotal for the distribution of cola among the participants (and organizers as well).

Thus, while the organizers are having the argument, discussing different variants of buying cola, the Winter School can't start. Your task is to count the number of all the possible ways to buy exactly  $n$  liters of cola and persuade the organizers that this number is too large, and if they keep on arguing, then the Winter Computer School will have to be organized in summer.

All the bottles of cola are considered indistinguishable, i.e. two variants of buying are different from each other only if they differ in the number of bottles of at least one kind.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, a, b, c;
8     cin >> n >> a >> b >> c;
9     int ans = 0;
10    for (int i = 0; i <= c; i++) {
11        for (int j = 0; j <= b; j++) {
12            int k = 2 * (n - 2 * i - j);
13            if (0 <= k && k <= a) {
14                ans++;
15            }
16        }
17    }
18    cout << ans << "\n";
19    return 0;
20 }
```

## 294: Repaintings

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A chessboard  $n \times m$  in size is given. During the zero minute we repaint all the black squares to the 0 color. During the  $i$ -th minute we repaint to the  $i$  color the initially black squares that have exactly four corner-adjacent squares painted  $i - 1$  (all such squares are repainted simultaneously). This process continues ad infinitum. You have to figure out how many squares we repainted exactly  $x$  times.

The upper left square of the board has to be assumed to be always black. Two squares are called corner-adjacent, if they have exactly one common point.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m, x;
```

```

8     cin >> n >> m >> x;
9     int ans = (max(0, n - 2 * (x-1)) * max(0, m - 2 * (x-1)) + 1) / 2;
10    ans -= (max(0, n - 2 * x) * max(0, m - 2 * x) + 1) / 2;
11    cout << ans << "\n";
12    return 0;
13 }

```

**295: Spelling Check**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Petya has noticed that when he types using a keyboard, he often presses extra buttons and adds extra letters to the words. Of course, the spell-checking system underlines the words for him and he has to click every word and choose the right variant. Petya got fed up with correcting his mistakes himself, that's why he decided to invent the function that will correct the words itself. Petya started from analyzing the case that happens to him most of the time, when all one needs is to delete one letter for the word to match a word from the dictionary. Thus, Petya faces one mini-task: he has a printed word and a word from the dictionary, and he should delete one letter from the first word to get the second one. And now the very non-trivial question that Petya faces is: which letter should he delete?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 a = input()
2 b = input()
3 i = 0
4 while i < len(b) and a[i] == b[i] :
5     i += 1
6 c = a[:i]+a[i+1:]
7 if b != c :
8     print(0)
9     exit()
10 j = i
11 while j and a[j - 1] == a[j] :
12     j -= 1
13 ans = list(range(j + 1, i + 2))
14 print(len(ans))
15 print(' '.join(map(str, ans)))

```

**296: Multiplication Table**

- Time limit: 2 seconds

- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

Petya studies positional notations. He has already learned to add and subtract numbers in the systems of notations with different radices and has moved on to a more complicated action - multiplication. To multiply large numbers one has to learn the multiplication table. Unfortunately, in the second grade students learn only the multiplication table of decimals (and some students even learn it in the first grade). Help Petya make a multiplication table for numbers in the system of notations with the radix k.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1  k = int(input())
2  for i in range(1,k) :
3      ans = []
4      for j in range(1,k) :
5          p = i * j
6          s = ''
7          while p :
8              s += str(p % k)
9              p //= k
10         ans.append(s[::-1])
11     print(' '.join(ans))
```

## 297: Company Income Growth

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

Petya works as a PR manager for a successful Berland company BerSoft. He needs to prepare a presentation on the company income growth since 2001 (the year of its founding) till now. Petya knows that in 2001 the company income amounted to a1 billion bourles, in 2002 - to a2 billion, ..., and in the current (2000 + n)-th year - an billion bourles. On the base of the information Petya decided to show in his presentation the linear progress history which is in his opinion perfect. According to a graph Petya has already made, in the first year BerSoft company income must amount to 1 billion bourles, in the second year - 2 billion bourles etc., each following year the income increases by 1 billion bourles. Unfortunately, the real numbers are different from the perfect ones. Among the numbers ai can even occur negative ones that are a sign of the companys losses in some years. That is why Petya wants

to ignore some data, in other words, cross some numbers  $a_i$  from the sequence and leave only some subsequence that has perfect growth.

Thus Petya has to choose a sequence of years  $y_1, y_2, \dots, y_k$ , so that in the year  $y_1$  the company income amounted to 1 billion bourles, in the year  $y_2$  - 2 billion bourles etc., in accordance with the perfect growth dynamics. Help him to choose the longest such sequence.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 n = int(input())
2 a = list(map(int, input().split()))
3 b = []
4 for i in range(n) :
5     if a[i] == len(b) + 1 :
6         b.append(2001 + i)
7 print(len(b))
8 print(''.join(map(str, b)))
```

## 298: Fractal

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: input.txt
- Output file: output.txt

Ever since Kalevitch, a famous Berland abstractionist, heard of fractals, he made them the main topic of his canvases. Every morning the artist takes a piece of graph paper and starts with making a model of his future canvas. He takes a square as big as  $n$  squares and paints some of them black. Then he takes a clean square piece of paper and paints the fractal using the following algorithm:

Step 1. The paper is divided into  $n^2$  identical squares and some of them are painted black according to the model.

Step 2. Every square that remains white is divided into  $n^2$  smaller squares and some of them are painted black according to the model.

Every following step repeats step 2.

Unfortunately, this tiresome work demands too much time from the painting genius. Kalevitch has been dreaming of making the process automatic to move to making 3D or even 4D fractals.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 #ifdef ONLINE_JUDGE
5     ifstream fin("input.txt");
6     ofstream fout("output.txt");
7 #else
8     #define fin cin
9     #define fout cout
10 #endif
11 int main() {
12     int n, k;
13     fin >> n >> k;
14     int N = 1;
15     for (int i = 0; i < k; i++) {
16         N *= n;
17     }
18     vector<string> s(n);
19     for (int i = 0; i < n; i++) {
20         fin >> s[i];
21     }
22     for (int i = 0; i < N; i++) {
23         for (int j = 0; j < N; j++) {
24             int x = i, y = j;
25             bool black = false;
26             for (int l = 0; l < k; l++) {
27                 black |= (s[x % n][y % n] == '*');
28                 x /= n, y /= n;
29             }
30             fout << ".**"[black];
31         }
32         fout << "\n";
33     }
34     return 0;
35 }
```

## 299: Extra-terrestrial Intelligence

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: input.txt
- Output file: output.txt

Recently Vasya got interested in finding extra-terrestrial intelligence. He made a simple extra-terrestrial signals receiver and was keeping a record of the signals for  $n$  days in a row. Each of those  $n$  days Vasya wrote a 1 in his notebook if he had received a signal that day and a 0 if he hadn't. Vasya thinks that he has found extra-terrestrial intelligence if there is a system in the way the signals have been received, i.e. if all the intervals between successive signals are equal. Otherwise, Vasya thinks that the signals were sent by some stupid aliens no one cares about. Help Vasya to deduce from the information given by the receiver if he has found extra-terrestrial intelligence or not.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 #ifdef ONLINE_JUDGE
5     ifstream fin("input.txt");
6     ofstream fout("output.txt");
7 #else
8     #define fin cin
9     #define fout cout
10 #endif
11 int main() {
12     int n;
13     fin >> n;
14     string s;
15     fin >> s;
16     vector<int> p;
17     for (int i = 0; i < n; i++) {
18         if (s[i] == '1') {
19             p.push_back(i);
20         }
21     }
22     for (int i = 1; i + 1 < p.size(); i++) {
23         if (p[i] * 2 != p[i - 1] + p[i + 1]) {
24             fout << "NO\n";
25             return 0;
26         }
27     }
28     fout << "YES\n";
29     return 0;
30 }
```

### 300: Road Map

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  cities in Berland. Each city has its index - an integer number from 1 to  $n$ . The capital has index  $r_1$ . All the roads in Berland are two-way. The road system is such that there is exactly one path from the capital to each city, i.e. the road map looks like a tree. In Berland's chronicles the road map is kept in the following way: for each city  $i$ , different from the capital, there is kept number  $\pi_i$  - index of the last city on the way from the capital to  $i$ .

Once the king of Berland Berl XXXIV decided to move the capital from city  $r_1$  to city  $r_2$ . Naturally, after this the old representation of the road map in Berland's chronicles became incorrect. Please, help the king find out a new representation of the road map in the way described above.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, r1, r2;
8     cin >> n >> r1 >> r2;
9     r1--, r2--;
10    vector<vector<int>> adj(n);
11    for (int i = 0; i < n; i++) {
12        if (i != r1) {
13            int p;
14            cin >> p;
15            p--;
16            adj[p].push_back(i);
17            adj[i].push_back(p);
18        }
19    }
20    vector<int> parent(n, -1);
21    function<void(int)> dfs = [&](int x) {
22        for (auto y : adj[x]) {
23            if (y == parent[x]) {
24                continue;
25            }
26            parent[y] = x;
27            dfs(y);
28        }
29    };
30    dfs(r2);
31    for (int i = 0; i < n; i++) {
32        if (i != r2) {
33            cout << parent[i] + 1 << " ";
34        }
35    }
36    return 0;
37 }
```

## blue

### constructive algorithms

#### 301: Colorful Grid

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Elena has a grid formed by  $n$  horizontal lines and  $m$  vertical lines. The horizontal lines are numbered by integers from 1 to  $n$  from top to bottom. The vertical lines are numbered by integers from 1 to  $m$

from left to right. For each  $x$  and  $y$  ( $1 \leq x \leq n, 1 \leq y \leq m$ ), the notation  $(x, y)$  denotes the point at the intersection of the  $x$ -th horizontal line and  $y$ -th vertical line.

Two points  $(x_1, y_1)$  and  $(x_2, y_2)$  are adjacent if and only if  $|x_1 - x_2| + |y_1 - y_2| = 1$ .

Elena calls a sequence of points  $p_1, p_2, \dots, p_g$  of length  $g$  a walk if and only if all the following conditions hold:

The first point  $p_1$  in this sequence is  $(1, 1)$ .

The last point  $p_g$  in this sequence is  $(n, m)$ .

For each  $1 \leq i < g$ , the points  $p_i$  and  $p_{i+1}$  are adjacent.

Note that the walk may contain the same point more than once. In particular, it may contain point  $(1, 1)$  or  $(n, m)$  multiple times.

There are  $n(m - 1) + (n - 1)m$  segments connecting the adjacent points in Elena's grid. Elena wants to color each of these segments in blue or red color so that there exists a walk  $p_1, p_2, \dots, p_{k+1}$  of length  $k + 1$  such that

out of  $k$  segments connecting two consecutive points in this walk, no two consecutive segments have the same color (in other words, for each  $1 \leq i < k$ , the color of the segment between points  $p_i$  and  $p_{i+1}$  differs from the color of the segment between points  $p_{i+1}$  and  $p_{i+2}$ ).

Please find any such coloring or report that there is no such coloring.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m, k;
6     cin >> n >> m >> k;
7     if (k < n + m - 2 || (k + n + m) % 2 != 0) {
8         cout << "NO\n";
9         return;
10    }
11    cout << "YES\n";
12    for (int i = 0; i < n; i++) {
13        for (int j = 0; j < m - 1; j++) {
14            int x = (i + j) % 2;
15            if (i == 0 && j == 0) {
16                x ^= 1;
17            }
18            cout << "RB"[x] << " \n"[j == m - 2];
19        }
20    }
21    for (int i = 0; i < n - 1; i++) {
22        for (int j = 0; j < m; j++) {
23            int x = (i + j) % 2;
24            if (i == 0 && j == 1) {
25                x ^= 1;
26            }
27            cout << "RB"[x] << " \n"[j == m - 1];
28        }
29    }
30}

```

```

26         }
27         cout << "RB"[x] << "\n"[j == m - 1];
28     }
29 }
30 int main() {
31     ios::sync_with_stdio(false);
32     cin.tie(nullptr);
33     int t;
34     cin >> t;
35     while (t--) {
36         solve();
37     }
38     return 0;
39 }
40 }
```

### 302: Array Painting

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array of  $n$  integers, where each integer is either 0, 1, or 2. Initially, each element of the array is blue.

Your goal is to paint each element of the array red. In order to do so, you can perform operations of two types:

pay one coin to choose a blue element and paint it red;

choose a red element which is not equal to 0 and a blue element adjacent to it, decrease the chosen red element by 1, and paint the chosen blue element red.

What is the minimum number of coins you have to spend to achieve your goal?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
```

```

12     }
13     vector<int> r(n);
14     iota(r.begin(), r.end(), 0);
15     vector<int> lst(n, 0), nxt(n, n - 1);
16     for (int i = 0; i < n - 1; i++) {
17         lst[i + 1] = a[i] == 0 ? i : lst[i];
18     }
19     for (int i = n - 1; i > 0; i--) {
20         nxt[i - 1] = a[i] == 0 ? i : nxt[i];
21     }
22     for (int i = 0; i < n; i++) {
23         if (a[i] == 1) {
24             r[lst[i]] = max(r[lst[i]], i);
25             r[i] = max(r[i], nxt[i]);
26         } else if (a[i] == 2) {
27             r[lst[i]] = max(r[lst[i]], nxt[i]);
28         }
29     }
30     for (int i = 1; i < n; i++) {
31         r[i] = max(r[i], r[i - 1]);
32     }
33     int ans = 0;
34     for (int i = 0; i < n; i = r[i] + 1) {
35         ans++;
36     }
37     cout << ans << "\n";
38     return 0;
39 }
```

### 303: Rudolph and Mimic

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an interactive task.

Rudolph is a scientist who studies alien life forms. There is a room in front of Rudolph with  $n$  different objects scattered around. Among the objects there is exactly one amazing creature - a mimic that can turn into any object. He has already disguised himself in this room and Rudolph needs to find him by experiment.

The experiment takes place in several stages. At each stage, the following happens:

Rudolf looks at all the objects in the room and writes down their types. The type of each object is indicated by a number; there can be several objects of the same type.

After inspecting, Rudolph can point to an object that he thinks is a mimic. After that, the experiment ends. Rudolph only has one try, so if he is unsure of the mimic's position, he does the next step instead.

Rudolf can remove any number of objects from the room (possibly zero). Then Rudolf leaves the room and at this time all objects, including the mimic, are mixed with each other, their order is changed, and the mimic can transform into any other object (even one that is not in the room).

After this, Rudolf returns to the room and repeats the stage. The mimic may not change appearance, but it can not remain a same object for more than two stages in a row.

Rudolf's task is to detect mimic in no more than five stages.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     i64 n;
6     cin >> n;
7     vector<int> a(n);
8     vector<int> cnta(10);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11         cnta[a[i]]++;
12     }
13     vector<int> b(n);
14     auto cntb = cnta;
15     while (cnta == cntb) {
16         cout << "- " << 0 << endl;
17         cntb.assign(10, 0);
18         for (int i = 0; i < n; i++) {
19             cin >> b[i];
20             cntb[b[i]]++;
21         }
22     }
23     int v = 0;
24     while (cntb[v] <= cnta[v]) {
25         v++;
26     }
27     vector<int> p;
28     for (int i = 0; i < n; i++) {
29         if (b[i] != v) {
30             p.push_back(i + 1);
31         }
32     }
33     cout << "- " << p.size();
34     for (auto x : p) {
35         cout << " " << x;
36     }
37     cout << endl;
38     n = cntb[v];
39     vector<int> c(n);
40     for (int i = 0; i < n; i++) {
41         cin >> c[i];
42     }
43     while (count(c.begin(), c.end(), v) == n) {
44         cout << "- " << 0 << endl;
45         for (int i = 0; i < n; i++) {
46             cin >> c[i];
47         }
48     }
}
```

```

49     for (int i = 0; i < n; i++) {
50         if (c[i] != v) {
51             cout << "! " << i + 1 << endl;
52             return;
53         }
54     }
55 }
56 int main() {
57     ios::sync_with_stdio(false);
58     cin.tie(nullptr);
59     cout << fixed << setprecision(10);
60     int t;
61     cin >> t;
62     while (t--) {
63         solve();
64     }
65     return 0;
66 }
```

### 304: Tenzing and His Animal Friends

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Tenzing has  $n$  animal friends. He numbers them from 1 to  $n$ .

One day Tenzing wants to play with his animal friends. To do so, Tenzing will host several games.

In one game, he will choose a set  $S$  which is a subset of  $\{1, 2, 3, \dots, n\}$  and choose an integer  $t$ . Then, he will play the game with the animals in  $S$  for  $t$  minutes.

But there are some restrictions:

Tenzing loves friend 1 very much, so 1 must be an element of  $S$ .

Tenzing doesn't like friend  $n$ , so  $n$  must not be an element of  $S$ .

There are  $m$  additional restrictions. The  $i$ -th special restriction is described by integers  $u_i$ ,  $v_i$  and  $y_i$ , suppose  $x$  is the total time that exactly one of  $u_i$  and  $v_i$  is playing with Tenzing. Tenzing must ensure that  $x$  is less or equal to  $y_i$ . Otherwise, there will be unhappiness.

Tenzing wants to know the maximum total time that he can play with his animal friends. Please find out the maximum total time that Tenzing can play with his animal friends and a way to organize the games that achieves this maximum total time, or report that he can play with his animal friends for an infinite amount of time. Also, Tenzing does not want to host so many games, so he will host at most  $n^2$  games.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<vector<pair<int, int>>> adj(n);
10    for (int i = 0; i < m; i++) {
11        int u, v, w;
12        cin >> u >> v >> w;
13        u--;
14        adj[u].emplace_back(v, w);
15        adj[v].emplace_back(u, w);
16    }
17    priority_queue<pair<i64, int>, vector<pair<i64, int>>, greater<> q;
18    q.emplace(0, 0);
19    vector<i64> dis(n, -1);
20    vector<int> a;
21    while (!q.empty()) {
22        auto [d, x] = q.top();
23        q.pop();
24        if (dis[x] != -1) {
25            continue;
26        }
27        dis[x] = d;
28        a.push_back(x);
29        if (x == n - 1) {
30            break;
31        }
32        for (auto [y, w] : adj[x]) {
33            q.emplace(d + w, y);
34        }
35    }
36    if (dis[n - 1] == -1) {
37        cout << "inf\n";
38        return 0;
39    }
40    cout << dis[n - 1] << " " << a.size() - 1 << "\n";
41    for (int i = 1; i < a.size(); i++) {
42        string s(n, '0');
43        for (int j = 0; j < i; j++) {
44            s[a[j]] = '1';
45        }
46        cout << s << " " << dis[a[i]] - dis[a[i - 1]] << "\n";
47    }
48    return 0;
49 }
```

### 305: Ira and Flamenco

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Ira loves Spanish flamenco dance very much. She decided to start her own dance studio and found  $n$  students,  $i$ th of whom has level  $a_i$ .

Ira can choose several of her students and set a dance with them. So she can set a huge number of dances, but she is only interested in magnificent dances. The dance is called magnificent if the following is true:

exactly  $m$  students participate in the dance;

levels of all dancers are pairwise distinct;

levels of every two dancers have an absolute difference strictly less than  $m$ .

For example, if  $m = 3$  and  $a = [4, 2, 2, 3, 6]$ , the following dances are magnificent (students participating in the dance are highlighted in red):  $[4, \textcolor{red}{2}, 2, 3, 6]$ ,  $[\textcolor{red}{4}, 2, 2, 3, 6]$ . At the same time dances  $[4, \textcolor{red}{2}, 2, 3, 6]$ ,  $[4, 2, \textcolor{red}{2}, 3, 6]$ ,  $[\textcolor{red}{4}, 2, 2, 3, 6]$  are not magnificent.

In the dance  $[4, \textcolor{red}{2}, 2, 3, 6]$  only 2 students participate, although  $m = 3$ .

The dance  $[4, \textcolor{red}{2}, 2, 3, 6]$  involves students with levels 2 and 2, although levels of all dancers must be pairwise distinct.

In the dance  $[4, \textcolor{red}{2}, 2, 3, 6]$  students with levels 3 and 6 participate, but  $|3 - 6| = 3$ , although  $m = 3$ .

Help Ira count the number of magnificent dances that she can set. Since this number can be very large, count it modulo  $10^9 + 7$ . Two dances are considered different if the sets of students participating in them are different.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
```

```

22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 void solve() {
35     int n, m;
36     cin >> n >> m;
37     vector<int> a(n);
38     map<int, int> cnt;
39     for (int i = 0; i < n; i++) {
40         cin >> a[i];
41         cnt[a[i]]++;
42     }
43     Z ans = 0;
44     Z res = 1;
45     int last = -1;
46     int miss = 0;
47     for (auto [x, y] : cnt) {
48         if (last != x - 1) {
49             res = 1;
50             miss = x - 1;
51         }
52         res *= y;
53         if (x - m > miss) {
54             res /= cnt[x - m];
55         }
56         if (x - m >= miss) {
57             ans += res;
58         }
59         last = x;
60     }
61     cout << ans << "\n";
62 }
63 int main() {
64     ios::sync_with_stdio(false);
65     cin.tie(nullptr);
66     int t;
67     cin >> t;
68     while (t--) {
69         solve();
70     }
71     return 0;
72 }
```

### 306: Fish Graph

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a simple undirected graph with  $n$  nodes and  $m$  edges. Note that the graph is not necessarily connected. The nodes are labeled from 1 to  $n$ .

We define a graph to be a Fish Graph if it contains a simple cycle with a special node  $u$  belonging to the cycle. Apart from the edges in the cycle, the graph should have exactly 2 extra edges. Both edges should connect to node  $u$ , but they should not be connected to any other node of the cycle.

Determine if the graph contains a subgraph that is a Fish Graph, and if so, find any such subgraph.

In this problem, we define a subgraph as a graph obtained by taking any subset of the edges of the original graph.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<vector<int>> adj(n);
8     for (int i = 0; i < m; i++) {
9         int u, v;
10        cin >> u >> v;
11        u--, v--;
12        adj[u].push_back(v);
13        adj[v].push_back(u);
14    }
15    for (int u = 0; u < n; u++) {
16        if (adj[u].size() >= 4) {
17            queue<pair<int, int>> q;
18            vector<array<int, 2>> s(n, {-1, -1});
19            vector<array<pair<int, int>, 2>> pre(n);
20            for (auto v : adj[u]) {
21                s[v][0] = v;
22                pre[v][0] = {u, 0};
23                q.emplace(v, 0);
24            }
25            while (!q.empty()) {
26                auto [x, t] = q.front();
27                q.pop();
28                for (auto y : adj[x]) {
29                    if (y != u) {
30                        if (s[y][0] == -1) {
31                            s[y][0] = s[x][t];
32                            pre[y][0] = {x, t};
33                            q.emplace(y, 0);
34                        } else if (s[y][1] == -1 && s[y][0] != s[x][t]) {
35                            s[y][1] = s[x][t];
36                            pre[y][1] = {x, t};
37                            q.emplace(y, 1);
38                        }
39                    } else if (t == 1) {
40                        cout << "YES\n";
41                        vector<bool> vis(n);
42                        vector<pair<int, int>> ans;
43                        ans.emplace_back(u + 1, x + 1);
44                        while (x != u) {

```

```

45             vis[x] = true;
46             auto [y, q] = pre[x][t];
47             ans.emplace_back(x + 1, y + 1);
48             tie(x, t) = pair(y, q);
49         }
50         int k = 0;
51         for (auto v : adj[u]) {
52             if (!vis[v] && k < 2) {
53                 ans.emplace_back(u + 1, v + 1);
54                 k++;
55             }
56         }
57         cout << ans.size() << "\n";
58         for (auto [x, y] : ans) {
59             cout << x << " " << y << "\n";
60         }
61         return;
62     }
63 }
64 }
65 }
66 }
67 cout << "NO\n";
68 }
69 int main() {
70     ios::sync_with_stdio(false);
71     cin.tie(nullptr);
72     int t;
73     cin >> t;
74     while (t--) {
75         solve();
76     }
77     return 0;
78 }
```

### 307: Unique Palindromes

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A palindrome is a string that reads the same backwards as forwards. For example, the string abcba is a palindrome, while the string abca is not.

Let  $p(t)$  be the number of unique palindromic substrings of string  $t$ , i. e. the number of substrings  $t[l \dots r]$  that are palindromes themselves. Even if some substring occurs in  $t$  several times, it's counted exactly once. (The whole string  $t$  is also counted as a substring of  $t$ ).

For example, string  $t = \text{abcbbcabcb}$  has  $p(t) = 6$  unique palindromic substrings: a, b, c, bb, bcb and cbcb.

Now, let's define  $p(s, m) = p(t)$  where  $t = s[1 \dots m]$ . In other words,  $p(s, m)$  is the number of palindromic substrings in the prefix of  $s$  of length  $m$ . For example,  $p(\text{abcbbcabcb}, 5) = p(\text{abcbb}) = 5$ .

You are given an integer  $n$  and  $k$  “conditions” ( $k \leq 20$ ). Let’s say that string  $s$ , consisting of  $n$  lowercase Latin letters, is good if all  $k$  conditions are satisfied at the same time. A condition is a pair  $(x_i, c_i)$  and have the following meaning:

$p(s, x_i) = c_i$ , i. e. a prefix of  $s$  of length  $x_i$  contains exactly  $c_i$  unique palindromic substrings.

Look in Notes if you need further clarifications.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> x(k), c(k);
8     for (int i = 0; i < k; i++) {
9         cin >> x[i];
10    }
11    for (int i = 0; i < k; i++) {
12        cin >> c[i];
13    }
14    if (c[0] > x[0]) {
15        cout << "NO\n";
16        return;
17    }
18    for (int i = 1; i < k; i++) {
19        if (c[i] - c[i - 1] > x[i] - x[i - 1]) {
20            cout << "NO\n";
21            return;
22        }
23    }
24    cout << "YES\n";
25    string s = "abc";
26    char lst = 'c';
27    int res = 3;
28    for (int i = 0; i < k; i++) {
29        while (x[i] - s.size() > c[i] - res) {
30            lst = lst == 'c' ? 'a' : lst + 1;
31            s += lst;
32        }
33        while (s.size() < x[i]) {
34            s += 'a' + i + 3;
35            res++;
36        }
37    }
38    cout << s << "\n";
39 }
40 int main() {
41     ios::sync_with_stdio(false);
42     cin.tie(nullptr);
43     int t;
44     cin >> t;
45     while (t--) {
46         solve();
47     }
48     return 0;
49 }
```

**308: 3-cycles**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

During a recent research Berland scientists found out that there were  $n$  cities in Ancient Berland, joined by two-way paths. Any two cities are joined by no more than one path. No path joins a city with itself. According to a well-known tradition, the road network was built so that it would be impossible to choose three cities from each of which one can get to any other one directly. That is, there was no cycle exactly as long as 3. Unfortunately, the road map has not been preserved till nowadays. Now the scientists are interested how much developed a country Ancient Berland was. Help them - find, what maximal number of roads could be in the country. You also have to restore any of the possible road maps.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     int m = n/2;
10    cout << m * (n-m) << "\n";
11    for (int i = 1; i <= m; i++) {
12        for (int j = m+1; j <= n; j++) {
13            cout << i << " " << j << "\n";
14        }
15    }
16    return 0;
17 }
```

**309: Make Palindrome**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A string is called palindrome if it reads the same from left to right and from right to left. For example “kazak”, “oo”, “r” and “mikhailrubinchikkihcniburliahkim” are palindroms, but strings “abb” and “ij” are not.

You are given string  $s$  consisting of lowercase Latin letters. At once you can choose any position in the string and change letter in that position to any other lowercase letter. So after each changing the length of the string doesn't change. At first you can change some letters in  $s$ . Then you can permute the order of letters as you want. Permutation doesn't count as changes.

You should obtain palindrome with the minimal number of changes. If there are several ways to do that you should get the lexicographically (alphabetically) smallest palindrome. So firstly you should minimize the number of changes and then minimize the palindrome lexicographically.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     string s;
8     cin >> s;
9     int cnt[26] {};
10    for (auto c : s) {
11        cnt[c - 'a']++;
12    }
13    vector<int> odd;
14    for (int i = 0; i < 26; i++) {
15        if (cnt[i] & 1) {
16            odd.push_back(i);
17        }
18    }
19    for (int i = 0; i < odd.size()/2; i++) {
20        cnt[odd[i]]++;
21        cnt[odd.size()-1 - i]--;
22    }
23    s = "";
24    if (odd.size() & 1) {
25        int x = odd[odd.size() / 2];
26        s += 'a' + x;
27        cnt[x]--;
28    }
29    for (int i = 25; i >= 0; i--) {
30        s = string(cnt[i]/2, 'a' + i) + s + string(cnt[i]/2, 'a' + i);
31    }
32    cout << s << "\n";
33    return 0;
34 }
```

### 310: Four Segments

- Time limit: 2 seconds

- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

Several months later Alex finally got his brother Bob's creation by post. And now, in his turn, Alex wants to boast about something to his brother. He thought for a while, and came to the conclusion that he has no ready creations, and decided to write a program for rectangles detection. According to his plan, the program detects if the four given segments form a rectangle of a positive area and with sides parallel to coordinate axes. As Alex does badly at school and can't write this program by himself, he asks you to help him.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int x1 = 1E9, x2 = -1E9, y1 = 1E9, y2 = -1E9;
8     set<pair<pair<int, int>, pair<int, int>> s;
9     for (int i = 0; i < 4; i++) {
10         pair<int, int> a, b;
11         cin >> a.first >> a.second >> b.first >> b.second;
12         if (a > b) {
13             swap(a, b);
14         }
15         s.emplace(a, b);
16         x1 = min(x1, a.first);
17         x1 = min(x1, b.first);
18         x2 = max(x2, a.first);
19         x2 = max(x2, b.first);
20         y1 = min(y1, a.second);
21         y1 = min(y1, b.second);
22         y2 = max(y2, a.second);
23         y2 = max(y2, b.second);
24     }
25     if (x1 == x2 || y1 == y2) {
26         cout << "NO\n";
27         return 0;
28     }
29     if (!s.count({{x1, y1}, {x1, y2}})) {
30         cout << "NO\n";
31         return 0;
32     }
33     if (!s.count({{x1, y1}, {x2, y1}})) {
34         cout << "NO\n";
35         return 0;
36     }
37     if (!s.count({{x2, y1}, {x2, y2}})) {
38         cout << "NO\n";
39         return 0;
40     }
41     if (!s.count({{x1, y2}, {x2, y2}})) {
```

```

42         cout << "NO\n";
43         return 0;
44     }
45     cout << "YES\n";
46     return 0;
47 }
```

### 311: Binary String Sorting

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a binary string  $s$  consisting of only characters 0 and/or 1.

You can perform several operations on this string (possibly zero). There are two types of operations:  
choose two consecutive elements and swap them. In order to perform this operation, you pay  $10^{12}$  coins;

choose any element from the string and remove it. In order to perform this operation, you pay  $10^{12} + 1$  coins.

Your task is to calculate the minimum number of coins required to sort the string  $s$  in non-decreasing order (i. e. transform  $s$  so that  $s_1 \leq s_2 \leq \dots \leq s_m$ , where  $m$  is the length of the string after applying all operations). An empty string is also considered sorted in non-decreasing order.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 X = 1E12;
5 constexpr i64 inf = 1E18;
6 void update(i64 &a, i64 b) {
7     if (a > b) {
8         a = b;
9     }
10 }
11 void solve() {
12     string s;
13     cin >> s;
14     int n = s.size();
15     vector<i64> dp(n + 1, array<i64, 2>{inf, inf});
16     dp[0][0] = 0;
17     for (int i = 0; i < n; i++) {
18         for (int x = 0; x < 2; x++) {
19             if (s[i] - '0' >= x) {
```

```

20             update(dp[i + 1][s[i] - '0'], dp[i][x]);
21         }
22         update(dp[i + 1][x], dp[i][x] + x + 1);
23         if (i + 1 < n && x <= s[i + 1] - '0' && s[i + 1] <= s[i]) {
24             update(dp[i + 2][s[i] - '0'], dp[i][x] + x);
25         }
26     }
27 }
28 cout << min(dp[n][0], dp[n][1]) << "\n";
29 }
30 int main() {
31     ios::sync_with_stdio(false);
32     cin.tie(nullptr);
33     int t;
34     cin >> t;
35     while (t--) {
36         solve();
37     }
38     return 0;
39 }
```

## 312: Train Splitting

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  big cities in Italy, and there are  $m$  train routes between pairs of cities. Each route connects two different cities bidirectionally. Moreover, using the trains one can reach every city starting from any other city.

Right now, all the routes are operated by the government-owned Italian Carriage Passenger Company, but the government wants to privatize the routes. The government does not want to give too much power to a single company, but it also does not want to make people buy a lot of different subscriptions. Also, it would like to give a fair chance to all companies. In order to formalize all these wishes, the following model was proposed.

There will be  $k \geq 2$  private companies indexed by  $1, 2, \dots, k$ . Each train route will be operated by exactly one of the  $k$  companies. Then:

For any company, there should exist two cities such that it is impossible to reach one from the other using only routes operated by that company.

On the other hand, for any two companies, it should be possible to reach every city from any other city using only routes operated by these two companies.

Find a plan satisfying all these criteria. It can be shown that a viable plan always exists. Please note that you can choose the number  $k$  and you do not have to minimize or maximize it.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<int> u(m), v(m), deg(n);
8     for (int i = 0; i < m; i++) {
9         cin >> u[i] >> v[i];
10        u[i]--, v[i]--;
11        deg[u[i]]++, deg[v[i]]++;
12    }
13    for (int x = 0; x < n; x++) {
14        if (deg[x] < n - 1) {
15            cout << 2 << "\n";
16            for (int i = 0; i < m; i++) {
17                cout << ((u[i] == x || v[i] == x) ? 1 : 2) << " \n"[i == m - 1];
18            }
19            return;
20        }
21    }
22    cout << 3 << "\n";
23    for (int i = 0; i < m; i++) {
24        int x;
25        if (u[i] + v[i] == 1) {
26            x = 1;
27        } else if (u[i] == 0 || v[i] == 0) {
28            x = 2;
29        } else {
30            x = 3;
31        }
32        cout << x << " \n"[i == m - 1];
33    }
34 }
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int t;
39     cin >> t;
40     while (t--) {
41         solve();
42     }
43     return 0;
44 }
```

### 313: Watch the Videos

- Time limit: 1 second
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Monocarp wants to watch  $n$  videos. Each video is only one minute long, but its size may be arbitrary.

The  $i$ -th video has the size  $a_i$  megabytes. All videos are published on the Internet. A video should be downloaded before it can be watched. Monocarp has poor Internet connection - it takes exactly 1 minute to download 1 megabyte of data, so it will require  $a_i$  minutes to download the  $i$ -th video.

Monocarp's computer has a hard disk of  $m$  megabytes. The disk is used to store the downloaded videos. Once Monocarp starts the download of a video of size  $s$ , the  $s$  megabytes are immediately reserved on a hard disk. If there are less than  $s$  megabytes left, the download cannot be started until the required space is freed. Each single video can be stored on the hard disk, since  $a_i \leq m$  for all  $i$ . Once the download is started, it cannot be interrupted. It is not allowed to run two or more downloads in parallel.

Once a video is fully downloaded to the hard disk, Monocarp can watch it. Watching each video takes exactly 1 minute and does not occupy the Internet connection, so Monocarp can start downloading another video while watching the current one.

When Monocarp finishes watching a video, he doesn't need it on the hard disk anymore, so he can delete the video, instantly freeing the space it occupied on a hard disk. Deleting a video takes negligible time.

Monocarp wants to watch all  $n$  videos as quickly as possible. The order of watching does not matter, since Monocarp needs to watch all of them anyway. Please calculate the minimum possible time required for that.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<int> a(n);
10    multiset<int> s;
11    i64 ans = 0;
12    for (int i = 0; i < n; i++) {
13        cin >> a[i];
14        ans += a[i];
15        s.insert(a[i]);
16    }
17    int last = 0;
18    for (int i = 0; i < n; i++) {
19        auto it = s.upper_bound(m - last);
20        if (it == s.begin()) {
21            ans++;
22            last = *s.rbegin();
23            s.erase(prev(s.end()));
24        } else {
25            it--;
26            last = *it;
27        }
28    }
29}
```

```

27         s.erase(it);
28     }
29 }
30 ans++;
31 cout << ans << "\n";
32 return 0;
33 }
```

### 314: Letter Exchange

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

A cooperative game is played by  $m$  people. In the game, there are  $3m$  sheets of paper:  $m$  sheets with letter ‘w’,  $m$  sheets with letter ‘i’, and  $m$  sheets with letter ‘n’.

Initially, each person is given three sheets (possibly with equal letters).

The goal of the game is to allow each of the  $m$  people to spell the word “win” using their sheets of paper. In other words, everyone should have one sheet with letter ‘w’, one sheet with letter ‘i’, and one sheet with letter ‘n’.

To achieve the goal, people can make exchanges. Two people participate in each exchange. Both of them choose exactly one sheet of paper from the three sheets they own and exchange it with each other.

Find the shortest sequence of exchanges after which everyone has one ‘w’, one ‘i’, and one ‘n’.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<string> s(n);
8     for (int i = 0; i < n; i++) {
9         cin >> s[i];
10    }
11    map<pair<char, char>, vector<int>> a;
12    vector<tuple<int, char, int, char>> ans;
13    for (int i = 0; i < n; i++) {
14        map<char, int> cnt;
15        for (auto x : s[i]) {
16            cnt[x]++;
17        }
18        for (auto [c1, c2] : a) {
19            if (cnt[c1] > 0 && cnt[c2] > 0) {
20                ans.push_back({i, c1, i, c2});
21                cnt[c1]--;
22                cnt[c2]--;
23            }
24        }
25    }
26    cout << ans.size() << endl;
27    for (auto [i1, c1, i2, c2] : ans) {
28        cout << i1 + 1 << " " << c1 << " " << i2 + 1 << " " << c2 << endl;
29    }
30}
```

```

17         }
18         vector<char> need;
19         vector<char> give;
20         for (auto x : {'w', 'i', '\n'}) {
21             if (cnt[x] == 0) {
22                 need.push_back(x);
23             } else {
24                 for (int i = 1; i < cnt[x]; i++) {
25                     give.push_back(x);
26                 }
27             }
28         }
29         for (int j = 0; j < need.size(); j++) {
30             char x = need[j], y = give[j];
31             if (!a[{y, x}].empty()) {
32                 ans.emplace_back(i, y, a[{y, x}].back(), x);
33                 a[{y, x}].pop_back();
34             } else {
35                 a[{x, y}].push_back(i);
36             }
37         }
38     }
39     char ch[3] = {'w', 'i', '\n'};
40     if (a[{ch[0], ch[1]}].empty()) {
41         swap(ch[0], ch[1]);
42     }
43     for (int i = 0; i < a[{ch[0], ch[1]}].size(); i++) {
44         int x = a[{ch[0], ch[1]}][i];
45         int y = a[{ch[1], ch[2]}][i];
46         int z = a[{ch[2], ch[0]}][i];
47         ans.emplace_back(x, ch[1], y, ch[2]);
48         ans.emplace_back(x, ch[2], z, ch[0]);
49     }
50     cout << ans.size() << "\n";
51     for (auto [a, b, c, d] : ans) {
52         cout << a + 1 << " " << b << " " << c + 1 << " " << d << "\n";
53     }
54 }
55 int main() {
56     ios::sync_with_stdio(false);
57     cin.tie(nullptr);
58     int t;
59     cin >> t;
60     while (t--) {
61         solve();
62     }
63     return 0;
64 }
```

**315: Bit Guessing Game**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an interactive problem.

Kira has a hidden positive integer  $n$ , and Hayato needs to guess it.

Initially, Kira gives Hayato the value  $\text{cnt}$  - the number of unit bits in the binary notation of  $n$ . To guess  $n$ , Hayato can only do operations of one kind: choose an integer  $x$  and subtract it from  $n$ . Note that after each operation, the number  $n$  changes. Kira doesn't like bad requests, so if Hayato tries to subtract a number  $x$  greater than  $n$ , he will lose to Kira. After each operation, Kira gives Hayato the updated value  $\text{cnt}$  - the number of unit bits in the binary notation of the updated value of  $n$ .

Kira doesn't have much patience, so Hayato must guess the original value of  $n$  after no more than 30 operations.

Since Hayato is in elementary school, he asks for your help. Write a program that guesses the number  $n$ . Kira is an honest person, so he chooses the initial number  $n$  before all operations and does not change it afterward.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int query(int x) {
5     cout << "-" << x << endl;
6     int ans;
7     cin >> ans;
8     return ans;
9 }
10 void solve() {
11     int cur;
12     cin >> cur;
13     int k = 1, ans = 0;
14     bool last = false;
15     while (cur > 0) {
16         ans += last ? k / 2 : k;
17         int r = query(last ? k / 2 : k);
18         if (r < cur) {
19             last = false;
20             cur = r;
21         } else {
22             last = true;
23         }
24         k *= 2;
25     }
26     cout << "!" << ans << endl;
27 }
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(nullptr);
31     int t;
32     cin >> t;
33     while (t--) {
34         solve();
35     }
36     return 0;
37 }
```

### 316: Lucky Permutation

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a permutation<sup>†</sup>  $p$  of length  $n$ .

In one operation, you can choose two indices  $1 \leq i < j \leq n$  and swap  $p_i$  with  $p_j$ .

Find the minimum number of operations needed to have exactly one inversion<sup>‡</sup> in the permutation.

<sup>†</sup> A permutation is an array consisting of  $n$  distinct integers from 1 to  $n$  in arbitrary order. For example,  $[2, 3, 1, 5, 4]$  is a permutation, but  $[1, 2, 2]$  is not a permutation (2 appears twice in the array), and  $[1, 3, 4]$  is also not a permutation ( $n = 3$  but there is 4 in the array).

<sup>‡</sup> The number of inversions of a permutation  $p$  is the number of pairs of indices  $(i, j)$  such that  $1 \leq i < j \leq n$  and  $p_i > p_j$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> p(n);
8     for (int i = 0; i < n; i++) {
9         cin >> p[i];
10        p[i]--;
11    }
12    vector<int> vis(n, -1);
13    int ans = n;
14    int sameAdj = 0;
15    for (int i = 0; i < n; i++) {
16        if (vis[i] != -1) continue;
17        ans--;
18        int j = i;
19        while (vis[j] == -1) {
20            vis[j] = i;
21            j = p[j];
22        }
23    }
24    for (int i = 1; i < n; i++) {
25        if (vis[i] == vis[i - 1]) {
26            sameAdj = 1;
27        }
28    }
29    cout << ans + (sameAdj ? -1 : 1) << "\n";
30 }
31 int main() {
32     ios::sync_with_stdio(false);

```

```

33     cin.tie(nullptr);
34     int t;
35     cin >> t;
36     while (t--) {
37         solve();
38     }
39     return 0;
40 }
```

### 317: Boris and His Amazing Haircut

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Boris thinks that chess is a tedious game. So he left his tournament early and went to a barber shop as his hair was a bit messy.

His current hair can be described by an array  $a_1, a_2, \dots, a_n$ , where  $a_i$  is the height of the hair standing at position  $i$ . His desired haircut can be described by an array  $b_1, b_2, \dots, b_n$  in a similar fashion.

The barber has  $m$  razors. Each has its own size and can be used at most once. In one operation, he chooses a razor and cuts a segment of Boris's hair. More formally, an operation is:

Choose any razor which hasn't been used before, let its size be  $x$ ;

Choose a segment  $[l, r]$  ( $1 \leq l \leq r \leq n$ );

Set  $a_i := \min(a_i, x)$  for each  $l \leq i \leq r$ ;

Notice that some razors might have equal sizes - the barber can choose some size  $x$  only as many times as the number of razors with size  $x$ .

He may perform as many operations as he wants, as long as any razor is used at most once and  $a_i = b_i$  for each  $1 \leq i \leq n$  at the end. He does not have to use all razors.

Can you determine whether the barber can give Boris his desired haircut?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 void solve() {
6     int n;
7     cin >> n;
8     vector<int> a(n), b(n);
```

```

9      for (int i = 0; i < n; i++) {
10         cin >> a[i];
11     }
12     for (int i = 0; i < n; i++) {
13         cin >> b[i];
14     }
15     int m;
16     cin >> m;
17     map<int, int> cnt;
18     for (int i = 0; i < m; i++) {
19         int x;
20         cin >> x;
21         cnt[x]++;
22     }
23     for (int i = 0; i < n; i++) {
24         if (a[i] < b[i]) {
25             cout << "NO\n";
26             return;
27         }
28     }
29     vector<int> p(n);
30     iota(p.begin(), p.end(), 0);
31     sort(p.begin(), p.end(), [&](int i, int j) {
32         return b[i] < b[j];
33     });
34     DSU dsu(n);
35     vector<int> mx(n);
36     for (int i = 0; i < n; i++) {
37         mx[i] = a[i] == b[i] ? -1 : b[i];
38     }
39     map<int, int> need;
40     for (int i = 0; i < n; i++) {
41         need[mx[i]]++;
42     }
43     auto merge = [&](int x, int y, int v) {
44         if (x < 0 || x >= n) return;
45         x = dsu.leader(x);
46         y = dsu.leader(y);
47         if (b[x] > v || b[y] > v) return;
48         if (x == y) return;
49         need[mx[x]]--;
50         need[mx[y]]--;
51         dsu.merge(x, y);
52         mx[x] = max(mx[x], mx[y]);
53         need[mx[x]]++;
54     };
55     for (int i = 0, j = 0; i < n; i = j) {
56         while (j < n && b[p[i]] == b[p[j]]) j++;
57         for (int k = i; k < j; k++) {
58             merge(p[k] - 1, p[k], b[p[i]]);
59             merge(p[k] + 1, p[k], b[p[i]]);
60         }
61         if (cnt[b[p[i]]] < need[b[p[i]]]) {
62             cout << "NO\n";
63             return;
64         }
65     }
66     cout << "YES\n";
67 }
68 int main() {
69     ios::sync_with_stdio(false);
70     cin.tie(nullptr);
71     int t;
72     cin >> t;

```

```

73     while (t--) {
74         solve();
75     }
76     return 0;
77 }
```

**318: Range = Sum**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an integer  $n$ . Find a sequence of  $n$  distinct integers  $a_1, a_2, \dots, a_n$  such that  $1 \leq a_i \leq 10^9$  for all  $i$  and

$$\max(a_1, a_2, \dots, a_n) - \min(a_1, a_2, \dots, a_n) = \sqrt{a_1 + a_2 + \dots + a_n}.$$

It can be proven that there exists a sequence of distinct integers that satisfies all the conditions above.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     if (n % 2 == 0) {
8         for (int i = n - n / 2; i < n; i++) {
9             cout << i << " ";
10    }
11    for (int i = n + 1; i <= n + n / 2; i++) {
12        cout << i << " \n"[i == n + n / 2];
13    }
14 } else {
15     int d = (n + 1) / 2;
16     for (int i = n + 2 - d; i <= n; i++) {
17         cout << i << " ";
18     }
19     for (int i = n + 3; i <= n + 2 + d; i++) {
20         cout << i << " \n"[i == n + 2 + d];
21     }
22 }
23 }
24 int main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
27     int t;
28     cin >> t;
29     while (t--) {
30         solve();
31     }
32     return 0;
33 }
```

33 }

### 319: Restore the Permutation

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A sequence of  $n$  numbers is called permutation if it contains all numbers from 1 to  $n$  exactly once. For example, the sequences [3, 1, 4, 2], [1] and [2, 1] are permutations, but [1, 2, 1], [0, 1] and [1, 3, 4] - are not.

For a permutation  $p$  of even length  $n$  you can make an array  $b$  of length  $\frac{n}{2}$  such that:

$$b_i = \max(p_{2i-1}, p_{2i}) \text{ for } 1 \leq i \leq \frac{n}{2}$$

For example, if  $p = [2, 4, 3, 1, 5, 6]$ , then:

$$b_1 = \max(p_1, p_2) = \max(2, 4) = 4$$

$$b_2 = \max(p_3, p_4) = \max(3, 1) = 3$$

$$b_3 = \max(p_5, p_6) = \max(5, 6) = 6$$

For a given array  $b$ , find the lexicographically minimal permutation  $p$  such that you can make the given array  $b$  from it.

If  $b = [4, 3, 6]$ , then the lexicographically minimal permutation from which it can be made is  $p = [1, 4, 2, 3, 5, 6]$ , since:

$$b_1 = \max(p_1, p_2) = \max(1, 4) = 4$$

$$b_2 = \max(p_3, p_4) = \max(2, 3) = 3$$

$$b_3 = \max(p_5, p_6) = \max(5, 6) = 6$$

A permutation  $x_1, x_2, \dots, x_n$  is lexicographically smaller than a permutation  $y_1, y_2, \dots, y_n$  if and only if there exists such  $i$  ( $1 \leq i \leq n$ ) that  $x_1 = y_1, x_2 = y_2, \dots, x_{i-1} = y_{i-1}$  and  $x_i < y_i$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
```

```

4  template<class Info,
5      class Merge = plus<Info>>
6  # struct SegmentTree;
7  # struct Info;
8  Info operator+(const Info &a, const Info &b) {
9      Info c;
10     c.sum = a.sum + b.sum;
11     c.pre = min(a.pre, a.sum + b.pre);
12     c.pos = c.pre == a.sum + b.pre ? b.pos : a.pos;
13     return c;
14 }
15 void solve() {
16     int n;
17     cin >> n;
18     vector<int> b(n / 2);
19     for (int i = 0; i < n / 2; i++) {
20         cin >> b[i];
21     }
22     if (set(b.begin(), b.end()).size() != b.size()) {
23         cout << -1 << "\n";
24         return;
25     }
26     vector<int> a(n, 1);
27     for (auto x : b) {
28         a[x - 1] = -1;
29     }
30     SegmentTree<Info> seg(vector<Info>(a.begin(), a.end()));
31     if (seg.rangeQuery(0, n).pre < 0) {
32         cout << -1 << "\n";
33         return;
34     }
35     for (int i = 0; i < n / 2; i++) {
36         int x = seg.rangeQuery(0, b[i] - 1).pos;
37         seg.modify(x, 0);
38         seg.modify(b[i] - 1, 0);
39         cout << x + 1 << " " << b[i] << " \n"[i == n / 2 - 1];
40     }
41 }
42 int main() {
43     ios::sync_with_stdio(false);
44     cin.tie(nullptr);
45     int t;
46     cin >> t;
47     while (t--) {
48         solve();
49     }
50     return 0;
51 }

```

### 320: Yet Another Problem

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  of  $n$  integers  $a_1, a_2, a_3, \dots, a_n$ .

You have to answer  $q$  independent queries, each consisting of two integers  $l$  and  $r$ .

Consider the subarray  $a[l : r] = [a_l, a_{l+1}, \dots, a_r]$ . You can apply the following operation to the subarray any number of times (possibly zero)- Choose two integers  $L, R$  such that  $l \leq L \leq R \leq r$  and  $R - L + 1$  is odd. Replace each element in the subarray from  $L$  to  $R$  with the XOR of the elements in the subarray  $[L, R]$ .

Choose two integers  $L, R$  such that  $l \leq L \leq R \leq r$  and  $R - L + 1$  is odd.

Replace each element in the subarray from  $L$  to  $R$  with the XOR of the elements in the subarray  $[L, R]$ .

The answer to the query is the minimum number of operations required to make all elements of the subarray  $a[l : r]$  equal to 0 or  $-1$  if it is impossible to make all of them equal to 0.

You can find more details about XOR operation here.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, q;
8     cin >> n >> q;
9     vector<i64> sum(n + 1);
10    vector<int> xsum(n + 1), a(n);
11    map<int, vector<int>> f[2];
12    for (int i = 0; i < n; i++) {
13        cin >> a[i];
14        xsum[i + 1] = xsum[i] ^ a[i];
15        sum[i + 1] = sum[i] + a[i];
16    }
17    for (int i = 0; i <= n; i++) {
18        f[i % 2][xsum[i]].push_back(i);
19    }
20    auto query = [&](int l, int r) {
21        if (sum[l] == sum[r]) {
22            return 0;
23        }
24        if (xsum[l] != xsum[r]) {
25            return -1;
26        }
27        if ((r - l) % 2) {
28            return 1;
29        }
30        if (!a[l] || !a[r - 1]) {
31            return 1;
32        }
33        auto &b = f[!(l % 2)][xsum[l]];
34        auto it = lower_bound(b.begin(), b.end(), l);
35        if (it != b.end() && *it < r) {
36            return 2;
37        }
38    };
39}

```

```

40     for (int i = 0; i < q; i++) {
41         int l, r;
42         cin >> l >> r;
43         l--;
44         cout << query(l, r) << "\n";
45     }
46     return 0;
47 }
```

**dp****321: Kim's Quest**

- Time limit: 3 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

In the long-forgotten halls of Kombinatoria's ancient academy, a gifted mathematician named Kim is faced with an unusual challenge. They found an old sequence of integers, which is believed to be a cryptic message from the legendary Kombinatoria's Oracle, and Kim wants to decipher its hidden meaning.

Kim's mission is to find specific patterns within the sequence, known as Harmonious Subsequences. These are extraordinary subsequences where the sum of every three consecutive numbers is even, and each subsequence must be at least three numbers in length.

Given a sequence  $a_i$  ( $1 \leq i \leq n$ ) of length  $n$ , its subsequence of length  $m$  is equal to  $a_{b_1}, a_{b_2}, \dots, a_{b_m}$  and is uniquely defined by a set of  $m$  indices  $b_j$ , such that  $1 \leq b_1 < b_2 < \dots < b_m \leq n$ . Subsequences given by different sets of indices  $b_j$  are considered different.

There's a twist in Kim's quest: the number of these Harmonious Subsequences could be overwhelming. To report the findings effectively, Kim must calculate the total number of these subsequences, presenting the answer as a remainder after dividing by the number 998 244 353.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
```

```

8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int n;
38     cin >> n;
39     vector<int> a(n);
40     for (int i = 0; i < n; i++) {
41         cin >> a[i];
42         a[i] %= 2;
43     }
44     array<Z, 4> dp{};
45     array<int, 2> cnt{};
46     Z ans = 0;
47     for (int i = 0; i < n; i++) {
48         auto g = dp;
49         for (int x = 0; x < 2; x++) {
50             int y = x ^ a[i];
51             dp[y | a[i] << 1] += g[x | y << 1];
52             ans += g[x | y << 1];
53         }
54         dp[0 | a[i] << 1] += cnt[0];
55         dp[1 | a[i] << 1] += cnt[1];
56         cnt[a[i]] += 1;
57     }
58     cout << ans << "\n";
59     return 0;
60 }

```

**322: Merge Not Sort**

- Time limit: 1 second
- Memory limit: 1024 megabytes
- Input file: standard input

- Output file: standard output

You are currently researching the Merge Sort algorithm. Merge Sort is a sorting algorithm that is based on the principle of Divide and Conquer. It works by dividing an array into two subarrays of equal length, sorting each subarrays, then merging the sorted subarrays back together to form the final sorted array.

You are particularly interested in the merging routine. Common merge implementation will combine two subarrays by iteratively comparing their first elements, and move the smaller one to a new merged array. More precisely, the merge algorithm can be presented by the following pseudocode.

During your research, you are keen to understand the behaviour of the merge algorithm when arrays  $A$  and  $B$  are not necessarily sorted. For example, if  $A = [3, 1, 6]$  and  $B = [4, 5, 2]$ , then  $\text{Merge}(A, B) = [3, 1, 4, 5, 2, 6]$ .

To further increase the understanding of the merge algorithm, you decided to work on the following problem. You are given an array  $C$  of length  $2 \cdot N$  such that it is a permutation of 1 to  $2 \cdot N$ . Construct any two arrays  $A$  and  $B$  of the same length  $N$ , such that  $\text{Merge}(A, B) = C$ , or determine if it is impossible to do so.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int N;
8     cin >> N;
9     vector<int> C(2 * N);
10    vector<int> dp(N + 1, 2 * N + 1);
11    dp[0] = 0;
12    vector<int> len;
13    int lst = 0;
14    for (int i = 0; i < 2 * N; i++) {
15        cin >> C[i];
16    }
17    for (int i = 1; i <= 2 * N; i++) {
18        if (i == 2 * N || C[i] > C[lst]) {
19            len.push_back(i - lst);
20            lst = i;
21        }
22    }
23    for (int i = 0; i < len.size(); i++) {
24        for (int j = N - len[i]; j >= 0; j--) {
25            if (dp[j] <= i && dp[j + len[i]] > i + 1) {
26                dp[j + len[i]] = i + 1;
27            }
28        }
29    }
30    if (dp[N] > len.size()) {
31        cout << -1 << "\n";

```

```

32         return 0;
33     }
34     vector<int> f(len.size());
35     int j = N;
36     for (int i = len.size() - 1; i >= 0; i--) {
37         if (dp[j] <= i) {
38             f[i] = 1;
39         } else {
40             f[i] = 0;
41             j -= len[i];
42         }
43     }
44     vector<int> A, B;
45     lst = 0;
46     for (int i = 0; i < len.size(); i++) {
47         for (int j = 0; j < len[i]; j++) {
48             int x = C[lst++];
49             if (f[i] == 0) {
50                 A.push_back(x);
51             } else {
52                 B.push_back(x);
53             }
54         }
55     }
56     for (int i = 0; i < N; i++) {
57         cout << A[i] << " \n"[i == N - 1];
58     }
59     for (int i = 0; i < N; i++) {
60         cout << B[i] << " \n"[i == N - 1];
61     }
62     return 0;
63 }
```

### 323: Robot Queries

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There is an infinite 2-dimensional grid. Initially, a robot stands in the point  $(0, 0)$ . The robot can execute four commands:

U - move from point  $(x, y)$  to  $(x, y + 1)$ ;

D - move from point  $(x, y)$  to  $(x, y - 1)$ ;

L - move from point  $(x, y)$  to  $(x - 1, y)$ ;

R - move from point  $(x, y)$  to  $(x + 1, y)$ .

You are given a sequence of commands  $s$  of length  $n$ . Your task is to answer  $q$  independent queries: given four integers  $x, y, l$  and  $r$ ; determine whether the robot visits the point  $(x, y)$ , while executing

a sequence  $s$ , but the substring from  $l$  to  $r$  is reversed (i. e. the robot performs commands in order  $s_1s_2s_3 \dots s_{l-1}s_ls_{r-1}s_{r-2} \dots s_ls_{r+1}s_{r+2} \dots s_n$ ).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct Point;
5 Point operator+(Point a, Point b) {
6     return {a.x + b.x, a.y + b.y};
7 }
8 Point operator-(Point a, Point b) {
9     return {a.x - b.x, a.y - b.y};
10 }
11 bool operator<(Point a, Point b) {
12     return a.x < b.x || (a.x == b.x && a.y < b.y);
13 }
14 int main() {
15     ios::sync_with_stdio(false);
16     cin.tie(nullptr);
17     int n, q;
18     cin >> n >> q;
19     vector<Point> p(n + 1);
20     string s;
21     cin >> s;
22     for (int i = 0; i < n; i++) {
23         p[i + 1] = p[i];
24         if (s[i] == 'U') {
25             p[i + 1].y += 1;
26         } else if (s[i] == 'D') {
27             p[i + 1].y -= 1;
28         } else if (s[i] == 'L') {
29             p[i + 1].x -= 1;
30         } else {
31             p[i + 1].x += 1;
32         }
33     }
34     map<Point, vector<int>> pos;
35     for (int i = 0; i <= n; i++) {
36         pos[p[i]].push_back(i);
37     }
38     auto get = [&](Point p, int l, int r) {
39         if (!pos.contains(p)) {
40             return false;
41         }
42         auto &vec = pos[p];
43         auto it = lower_bound(vec.begin(), vec.end(), l);
44         return it != vec.end() && *it <= r;
45     };
46     while (q--) {
47         int x, y, l, r;
48         cin >> x >> y >> l >> r;
49         l--;
50         if (get(Point{x, y}, 0, l) || get(Point{x, y}, r, n) || get(p[l] + p[r] - Point{x,
51             y}, l, r)) {
52             cout << "YES\n";
53         } else {
54             cout << "NO\n";
55         }
56     }
57 }
```

```

54         }
55     }
56     return 0;
57 }
```

### 324: Bakry and Partitioning

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Bakry faced a problem, but since he's lazy to solve it, he asks for your help.

You are given a tree of  $n$  nodes, the  $i$ -th node has value  $a_i$  assigned to it for each  $i$  from 1 to  $n$ . As a reminder, a tree on  $n$  nodes is a connected graph with  $n - 1$  edges.

You want to delete at least 1, but at most  $k - 1$  edges from the tree, so that the following condition would hold:

For every connected component calculate the bitwise XOR of the values of the nodes in it. Then, these values have to be the same for all connected components.

For every connected component calculate the bitwise XOR of the values of the nodes in it. Then, these values have to be the same for all connected components.

Is it possible to achieve this condition?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<vector<int>> adj(n);
12    for (int i = 1; i < n; i++) {
13        int u, v;
14        cin >> u >> v;
15        u--;
16        v--;
17        adj[u].push_back(v);
18        adj[v].push_back(u);
19    }
20    auto s = a;
21    auto dfs = [&](auto self, int x, int p) -> void {
22        for (auto y : adj[x]) {
```

```

22         if (y == p) {
23             continue;
24         }
25         self(self, y, x);
26         s[x] ^= s[y];
27     }
28 }
29 dfs(dfs, 0, -1);
30 if (s[0] == 0) {
31     cout << "YES\n";
32     return;
33 }
34 if (k == 2) {
35     cout << "NO\n";
36     return;
37 }
38 int cnt = 1;
39 auto dfs1 = [&](auto self, int x, int p) -> int {
40     int sum = a[x];
41     for (auto y : adj[x]) {
42         if (y == p) {
43             continue;
44         }
45         sum ^= self(self, y, x);
46     }
47     if (sum == s[0]) {
48         cnt += 1;
49         return 0;
50     }
51     return sum;
52 };
53 dfs1(dfs1, 0, -1);
54 if (cnt >= 3) {
55     cout << "YES\n";
56 } else {
57     cout << "NO\n";
58 }
59 }
60 int main() {
61     ios::sync_with_stdio(false);
62     cin.tie(nullptr);
63     int t;
64     cin >> t;
65     while (t--) {
66         solve();
67     }
68     return 0;
69 }
```

### 325: The Number of Imposters

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Theofanis started playing the new online game called “Among them”. However, he always plays with Cypriot players, and they all have the same name: “Andreas” (the most common name in Cyprus).

In each game, Theofanis plays with  $n$  other players. Since they all have the same name, they are numbered from 1 to  $n$ .

The players write  $m$  comments in the chat. A comment has the structure of “ $i\ j\ c$ ” where  $i$  and  $j$  are two distinct integers and  $c$  is a string ( $1 \leq i, j \leq n; i \neq j; c$  is either imposter or crewmate). The comment means that player  $i$  said that player  $j$  has the role  $c$ .

An imposter always lies, and a crewmate always tells the truth.

Help Theofanis find the maximum possible number of imposters among all the other Cypriot players, or determine that the comments contradict each other (see the notes for further explanation).

Note that each player has exactly one role: either imposter or crewmate.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<vector<pair<int, int>>> adj(n);
8     for (int i = 0; i < m; i++) {
9         int u, v;
10        string c;
11        cin >> u >> v >> c;
12        u--, v--;
13        int x = (c == "impostor");
14        adj[u].emplace_back(v, x);
15        adj[v].emplace_back(u, x);
16    }
17    vector<int> f(n, -1);
18    int ans = 0;
19    for (int i = 0; i < n; i++) {
20        if (f[i] == -1) {
21            queue<int> q;
22            q.push(i);
23            f[i] = 0;
24            array<int, 2> cnt{0, 0};
25            while (!q.empty()) {
26                int x = q.front();
27                q.pop();
28                cnt[f[x]] += 1;
29                for (auto [y, w] : adj[x]) {
30                    if (f[y] == -1) {
31                        f[y] = f[x] ^ w;
32                        q.push(y);
33                    } else if (f[y] != (f[x] ^ w)) {
34                        ans = -1;
35                    }
36                }
37            }
38            if (ans != -1) {
39                ans += max(cnt[0], cnt[1]);
40            }
41        }
42    }
}

```

```

43     cout << ans << "\n";
44 }
45 int main() {
46     ios::sync_with_stdio(false);
47     cin.tie(nullptr);
48     int t;
49     cin >> t;
50     while (t--) {
51         solve();
52     }
53     return 0;
54 }
```

### 326: Minimum Maximum Distance

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have a tree with  $n$  vertices, some of which are marked. A tree is a connected undirected graph without cycles.

Let  $f_i$  denote the maximum distance from vertex  $i$  to any of the marked vertices.

Your task is to find the minimum value of  $f_i$  among all vertices.

For example, in the tree shown in the example, vertices 2, 6, and 7 are marked. Then the array  $f(i) = [2, 3, 2, 4, 4, 3, 3]$ . The minimum  $f_i$  is for vertices 1 and 3.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> mark(n);
8     for (int i = 0; i < k; i++) {
9         int a;
10        cin >> a;
11        a--;
12        mark[a] = 1;
13    }
14    vector<vector<int>> adj(n);
15    for (int i = 1; i < n; i++) {
16        int u, v;
17        cin >> u >> v;
18        u--, v--;
```

```

19         adj[u].push_back(v);
20         adj[v].push_back(u);
21     }
22     vector<int> dis(n);
23     auto bfs = [&](int x) {
24         queue<int> q;
25         dis.assign(n, -1);
26         q.push(x);
27         dis[x] = 0;
28         while (!q.empty()) {
29             int x = q.front();
30             q.pop();
31             for (auto y : adj[x]) {
32                 if (dis[y] == -1) {
33                     dis[y] = dis[x] + 1;
34                     q.push(y);
35                 }
36             }
37         }
38         int t = -1;
39         for (int i = 0; i < n; i++) {
40             if (mark[i] && (t == -1 || dis[i] > dis[t])) {
41                 t = i;
42             }
43         }
44         return t;
45     };
46     int a = bfs(0);
47     int b = bfs(a);
48     auto f = dis;
49     bfs(b);
50     for (int i = 0; i < n; i++) {
51         f[i] = max(f[i], dis[i]);
52     }
53     int ans = *min_element(f.begin(), f.end());
54     cout << ans << "\n";
55 }
56 int main() {
57     ios::sync_with_stdio(false);
58     cin.tie(nullptr);
59     int t;
60     cin >> t;
61     while (t--) {
62         solve();
63     }
64     return 0;
65 }
```

### 327: Completely Searching for Inversions

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Pak Chanek has a directed acyclic graph (a directed graph that does not have any cycles) containing  $N$  vertices. Vertex  $i$  has  $S_i$  edges directed away from that vertex. The  $j$ -th edge of vertex  $i$  that is

directed away from it, is directed towards vertex  $L_{i,j}$  and has an integer  $W_{i,j}$  ( $0 \leq W_{i,j} \leq 1$ ). Another information about the graph is that the graph is shaped in such a way such that each vertex can be reached from vertex 1 via zero or more directed edges.

Pak Chanek has an array  $Z$  that is initially empty.

Pak Chanek defines the function dfs as follows:

Note that the function does not keep track of which vertices have been visited, so each vertex can be processed more than once.

Let's say Pak Chanek does  $\text{dfs}(1)$  once. After that, Pak Chanek will get an array  $Z$  containing some elements 0 or 1. Define an inversion in array  $Z$  as a pair of indices  $(x, y)$  ( $x < y$ ) such that  $Z_x > Z_y$ . How many different inversions in  $Z$  are there if Pak Chanek does  $\text{dfs}(1)$  once? Since the answer can be very big, output the answer modulo 998 244 353.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int N;
38     cin >> N;

```

```

39     vector<vector<pair<int, int>>> adj(N);
40     for (int i = 0; i < N; i++) {
41         int S;
42         cin >> S;
43         for (int j = 0; j < S; j++) {
44             int L, W;
45             cin >> L >> W;
46             L--;
47             adj[i].push_back({L, W});
48         }
49     }
50     vector<array<Z, 3>> f(N);
51     vector<bool> vis(N);
52     auto dfs = [&](auto self, int x) {
53         if (vis[x]) {
54             return f[x];
55         }
56         vis[x] = true;
57         for (auto [y, w] : adj[x]) {
58             f[x][2] += (w == 0) * f[y][1];
59             f[x][w] += 1;
60             auto g = self(self, y);
61             f[x][2] += f[x][1] * g[0] + g[2];
62             f[x][0] += g[0];
63             f[x][1] += g[1];
64         }
65         return f[x];
66     };
67     cout << dfs(dfs, 0)[2] << "\n";
68     return 0;
69 }
```

### 328: Magic Will Save the World

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A portal of dark forces has opened at the border of worlds, and now the whole world is under a terrible threat. To close the portal and save the world, you need to defeat  $n$  monsters that emerge from the portal one after another.

Only the sorceress Vika can handle this. She possesses two magical powers - water magic and fire magic. In one second, Vika can generate  $w$  units of water mana and  $f$  units of fire mana. She will need mana to cast spells. Initially Vika have 0 units of water mana and 0 units of fire mana.

Each of the  $n$  monsters that emerge from the portal has its own strength, expressed as a positive integer. To defeat the  $i$ -th monster with strength  $s_i$ , Vika needs to cast a water spell or a fire spell of at least the same strength. In other words, Vika can spend at least  $s_i$  units of water mana on a water spell, or at least  $s_i$  units of fire mana on a fire spell.

Vika can create and cast spells instantly. Vika can cast an unlimited number of spells every second as long she has enough mana for that.

The sorceress wants to save the world as quickly as possible, so tell her how much time she will need.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int N = 100 * 1E4;
5 void solve() {
6     int w, f;
7     cin >> w >> f;
8     int n;
9     cin >> n;
10    vector<int> s(n);
11    for (int i = 0; i < n; i++) {
12        cin >> s[i];
13    }
14    bitset<N + 1> dp{};
15    dp[0] = 1;
16    int sum = 0;
17    for (int i = 0; i < n; i++) {
18        dp |= dp << s[i];
19        sum += s[i];
20    }
21    int ans = 1E9;
22    for (int i = 0; i <= sum; i++) {
23        if (dp[i]) {
24            ans = min(ans, max((i + w - 1) / w, (sum - i + f - 1) / f));
25        }
26    }
27    cout << ans << "\n";
28 }
29 int main() {
30     ios::sync_with_stdio(false);
31     cin.tie(nullptr);
32     int t;
33     cin >> t;
34     while (t--) {
35         solve();
36     }
37     return 0;
38 }
```

### 329: Credit Card

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Recently Luba got a credit card and started to use it. Let's consider  $n$  consecutive days Luba uses the card.

She starts with 0 money on her account.

In the evening of  $i$ -th day a transaction  $a_i$  occurs. If  $a_i > 0$ , then  $a_i$  bourles are deposited to Luba's account. If  $a_i < 0$ , then  $a_i$  bourles are withdrawn. And if  $a_i = 0$ , then the amount of money on Luba's account is checked.

In the morning of any of  $n$  days Luba can go to the bank and deposit any positive integer amount of burles to her account. But there is a limitation: the amount of money on the account can never exceed  $d$ .

It can happen that the amount of money goes greater than  $d$  by some transaction in the evening. In this case answer will be -1.

Luba must not exceed this limit, and also she wants that every day her account is checked (the days when  $a_i = 0$ ) the amount of money on her account is non-negative. It takes a lot of time to go to the bank, so Luba wants to know the minimum number of days she needs to deposit some money to her account (if it is possible to meet all the requirements). Help her!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, d;
8     cin >> n >> d;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    vector<int> sum(n + 1);
14    for (int i = 0; i < n; i++) {
15        sum[i + 1] = sum[i] + a[i];
16    }
17    auto sufmax = sum;
18    for (int i = n - 1; i >= 0; i--) {
19        sufmax[i] = max(sufmax[i], sufmax[i + 1]);
20    }
21    int add = 0;
22    int ans = 0;
23    if (sufmax[0] > d) {
24        cout << -1 << "\n";
25        return 0;
26    }
27    for (int i = 0; i < n; i++) {
28        if (a[i] == 0) {

```

```

29         if (sum[i] + add < 0) {
30             if (sufmax[i] - sum[i] > d) {
31                 cout << -1 << "\n";
32                 return 0;
33             }
34             add += d - (sufmax[i] + add);
35             ans++;
36         }
37     }
38 }
39 cout << ans << "\n";
40 return 0;
41 }
```

### 330: Andrey and Escape from Capygrad

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

An incident occurred in Capygrad, the capital of Tyagoland, where all the capybaras in the city went crazy and started throwing mandarins. Andrey was forced to escape from the city as far as possible, using portals.

Tyagoland is represented by a number line, and the city of Capygrad is located at point 0. There are  $n$  portals all over Tyagoland, each of which is characterised by four integers  $l_i, r_i, a_i$  and  $b_i$  ( $1 \leq l_i \leq a_i \leq b_i \leq r_i \leq 10^9$ ). Note that the segment  $[a_i, b_i]$  is contained in the segment  $[l_i, r_i]$ .

If Andrey is on the segment  $[l_i, r_i]$ , then the portal can teleport him to any point on the segment  $[a_i, b_i]$ . Andrey has a pass that allows him to use the portals an unlimited number of times.

Andrey thinks that the point  $x$  is on the segment  $[l, r]$  if the inequality  $l \leq x \leq r$  is satisfied.

Andrey has  $q$  options for where to start his escape, each option is characterized by a single integer  $x_i$  - the starting position of the escape. He wants to escape from Capygrad as far as possible (to the point with the maximum possible coordinate). Help Andrey determine how far he could escape from Capygrad, starting at each of the  $q$  positions.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<array<int, 2>> seg(n);
```

```

8     for (int i = 0; i < n; i++) {
9         int l, r, a, b;
10        cin >> l >> r >> a >> b;
11        seg[i] = {l, b};
12    }
13    sort(seg.begin(), seg.end());
14    vector<array<int, 2>> a;
15    for (auto [l, r] : seg) {
16        if (!a.empty() && l <= a.back()[1]) {
17            a.back()[1] = max(a.back()[1], r);
18        } else {
19            a.push_back({l, r});
20        }
21    }
22    int q;
23    cin >> q;
24    for (int i = 0; i < q; i++) {
25        int x;
26        cin >> x;
27        int j = lower_bound(a.begin(), a.end(), array{x + 1, 0}) - a.begin() - 1;
28        if (j >= 0) {
29            x = max(x, a[j][1]);
30        }
31        cout << x << " \n"[i == q - 1];
32    }
33 }
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int t;
38     cin >> t;
39     while (t--) {
40         solve();
41     }
42     return 0;
43 }
```

### 331: PermuTree (easy version)

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

This is the easy version of the problem. The differences between the two versions are the constraint on  $n$  and the time limit. You can make hacks only if both versions of the problem are solved.

You are given a tree with  $n$  vertices rooted at vertex 1.

For some permutation<sup>†</sup>  $a$  of length  $n$ , let  $f(a)$  be the number of pairs of vertices  $(u, v)$  such that  $a_u < a_{\text{lca}(u,v)} < a_v$ . Here,  $\text{lca}(u, v)$  denotes the lowest common ancestor of vertices  $u$  and  $v$ .

Find the maximum possible value of  $f(a)$  over all permutations  $a$  of length  $n$ .

<sup>†</sup> A permutation of length  $n$  is an array consisting of  $n$  distinct integers from 1 to  $n$  in arbitrary order.

For example, [2, 3, 1, 5, 4] is a permutation, but [1, 2, 2] is not a permutation (2 appears twice in the array), and [1, 3, 4] is also not a permutation ( $n = 3$  but there is 4 in the array).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> p(n, -1), siz(n, 1);
10    for (int i = 1; i < n; i++) {
11        cin >> p[i];
12        p[i]--;
13    }
14    for (int i = n - 1; i; i--) {
15        siz[p[i]] += siz[i];
16    }
17    i64 ans = 0;
18    vector<vector<int>> a(n);
19    for (int i = 1; i < n; i++) {
20        a[p[i]].push_back(siz[i]);
21    }
22    for (int i = 0; i < n; i++) {
23        if (a[i].empty()) {
24            continue;
25        }
26        auto &v = a[i];
27        sort(v.begin(), v.end(), greater());
28        int sum = accumulate(v.begin() + 1, v.end(), 0);
29        vector<int> dp(sum + 1);
30        dp[0] = 1;
31        for (int i = 1, j = i; i < v.size(); i = j) {
32            while (j < v.size() && v[i] == v[j]) {
33                j++;
34            }
35            int cnt = j - i;
36            int k = 1;
37            while (k < cnt) {
38                int x = v[i] * k;
39                for (int j = sum; j >= x; j--) {
40                    dp[j] |= dp[j - x];
41                }
42                cnt -= k;
43                k *= 2;
44            }
45            int x = v[i] * cnt;
46            for (int j = sum; j >= x; j--) {
47                dp[j] |= dp[j - x];
48            }
49        }
50        i64 res = 0;
51        for (int j = 0; j <= sum; j++) {
52            if (dp[j]) {
53                res = max(res, 1LL * j * (siz[i] - 1 - j));
54                res = max(res, 1LL * (j + v[0]) * (siz[i] - 1 - (j + v[0])));
55            }
56        }
}

```

```

57         ans += res;
58     }
59     cout << ans << "\n";
60     return 0;
61 }
```

### 332: Rudolf and CodeVid-23

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A new virus called “CodeVid-23” has spread among programmers. Rudolf, being a programmer, was not able to avoid it.

There are  $n$  symptoms numbered from 1 to  $n$  that can appear when infected. Initially, Rudolf has some of them. He went to the pharmacy and bought  $m$  medicines.

For each medicine, the number of days it needs to be taken is known, and the set of symptoms it removes. Unfortunately, medicines often have side effects. Therefore, for each medicine, the set of symptoms that appear when taking it is also known.

After reading the instructions, Rudolf realized that taking more than one medicine at a time is very unhealthy.

Rudolph wants to be healed as soon as possible. Therefore, he asks you to calculate the minimum number of days to remove all symptoms, or to say that it is impossible.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     i64 n, m;
6     cin >> n >> m;
7     string s;
8     cin >> s;
9     int S = 0;
10    for (int i = 0; i < n; i++) {
11        S |= (s[i] - '0') << i;
12    }
13    vector<int> c(m), a(m), b(m);
14    for (int i = 0; i < m; i++) {
15        cin >> c[i];
16        cin >> s;
17        for (int j = 0; j < n; j++) {
18            a[i] |= (s[j] - '0') << j;
19        }
}
```

```

20     cin >> s;
21     for (int j = 0; j < n; j++) {
22         b[i] |= (s[j] - '0') << j;
23     }
24 }
25 vector dis(1 << n, -1);
26 priority_queue<pair<int, int>, vector<pair<int, int>>, greater<> q;
27 q.emplace(0, S);
28 while (!q.empty()) {
29     auto [d, x] = q.top();
30     q.pop();
31     if (dis[x] != -1) {
32         continue;
33     }
34     dis[x] = d;
35     for (int i = 0; i < m; i++) {
36         q.emplace(d + c[i], (x & ~a[i]) | b[i]);
37     }
38 }
39 cout << dis[0] << "\n";
40 }
41 int main() {
42     ios::sync_with_stdio(false);
43     cin.tie(nullptr);
44     cout << fixed << setprecision(10);
45     int t;
46     cin >> t;
47     while (t--) {
48         solve();
49     }
50     return 0;
51 }

```

### 333: Omsk Metro (simple version)

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

This is the simple version of the problem. The only difference between the simple and hard versions is that in this version  $u = 1$ .

As is known, Omsk is the capital of Berland. Like any capital, Omsk has a well-developed metro system. The Omsk metro consists of a certain number of stations connected by tunnels, and between any two stations there is exactly one path that passes through each of the tunnels no more than once. In other words, the metro is a tree.

To develop the metro and attract residents, the following system is used in Omsk. Each station has its own weight  $x \in \{-1, 1\}$ . If the station has a weight of  $-1$ , then when the station is visited by an Omsk resident, a fee of 1 burle is charged. If the weight of the station is 1, then the Omsk resident is rewarded with 1 burle.

Omsk Metro currently has only one station with number 1 and weight  $x = 1$ . Every day, one of the following events occurs:

A new station with weight  $x$  is added to the station with number  $v_i$ , and it is assigned a number that is one greater than the number of existing stations.

Alex, who lives in Omsk, wonders: is there a subsegment† (possibly empty) of the path between vertices  $u$  and  $v$  such that, by traveling along it, exactly  $k$  burles can be earned (if  $k < 0$ , this means that  $k$  burles will have to be spent on travel). In other words, Alex is interested in whether there is such a subsegment of the path that the sum of the weights of the vertices in it is equal to  $k$ . Note that the subsegment can be empty, and then the sum is equal to 0.

You are a friend of Alex, so your task is to answer Alex's questions.

†Subsegment - continuous sequence of elements.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct Info;
5 Info operator+(Info a, Info b) {
6     Info c;
7     c.mxpre = max(a.mxpre, a.sum + b.mxpre);
8     c.mnpre = min(a.mnpre, a.sum + b.mnpre);
9     c.mxsuf = max(b.mxsuf, b.sum + a.mxsuf);
10    c.mnsuf = min(b.mnsuf, b.sum + a.mnsuf);
11    c.mx = max({a.mx, b.mx, a.mxsuf + b.mxpre});
12    c.mn = min({a.mn, b.mn, a.mnsuf + b.mnpre});
13    c.sum = a.sum + b.sum;
14    return c;
15 }
16 Info rev(Info a) {
17     swap(a.mxpre, a.mxsuf);
18     swap(a.mnpre, a.mnsuf);
19     return a;
20 }
21 void solve() {
22     int n;
23     cin >> n;
24     vector<int> p{-1}, x{1};
25     vector<array<int, 3>> qry;
26     for (int i = 0; i < n; i++) {
27         char o;
28         cin >> o;
29         if (o == '+') {
30             int v, xi;
31             cin >> v >> xi;
32             v--;
33             p.push_back(v);
34             x.push_back(xi);
35         } else {
36             int u, v, k;
37             cin >> u >> v >> k;
38             u--, v--;

```

```

39         qry.push_back({u, v, k});
40     }
41 }
42 n = p.size();
43 const int logn = __lg(n);
44 vector pp(logn + 1, vector<int>(n, -1));
45 vector f(logn + 1, vector<Info>(n));
46 vector<int> dep(n);
47 for (int i = 0; i < n; i++) {
48     if (i > 0) {
49         dep[i] = dep[p[i]] + 1;
50     }
51     pp[0][i] = p[i];
52     if (x[i] == 1) {
53         f[0][i].mx = f[0][i].mxpre = f[0][i].mxsuf = f[0][i].sum = 1;
54     } else {
55         f[0][i].mn = f[0][i].mnpree = f[0][i].mnsuf = f[0][i].sum = -1;
56     }
57     for (int j = 0; (2 << j) <= dep[i] + 1; j++) {
58         pp[j + 1][i] = pp[j][pp[j][i]];
59         f[j + 1][i] = f[j][i] + f[j][pp[j][i]];
60     }
61 }
62 auto query = [&](int x, int y) {
63     if (dep[x] < dep[y]) {
64         swap(x, y);
65     }
66     Info l, r;
67     for (int i = logn; i >= 0; i--) {
68         if (dep[x] - (1 << i) >= dep[y]) {
69             l = l + f[i][x];
70             x = pp[i][x];
71         }
72     }
73     if (x == y) {
74         return l + f[0][x];
75     }
76     for (int i = logn; i >= 0; i--) {
77         if (pp[i][x] != pp[i][y]) {
78             l = l + f[i][x];
79             r = r + f[i][y];
80             x = pp[i][x];
81             y = pp[i][y];
82         }
83     }
84     return l + f[1][x] + f[0][y] + rev(r);
85 };
86 for (auto [u, v, k] : qry) {
87     auto info = query(u, v);
88     if (info.mn <= k && k <= info.mx) {
89         cout << "YES\n";
90     } else {
91         cout << "NO\n";
92     }
93 }
94 int main() {
95     ios::sync_with_stdio(false);
96     cin.tie(nullptr);
97     int t;
98     cin >> t;
99     while (t--) {
100         solve();
101     }
102 }
```

```
103     return 0;
104 }
```

### 334: Ranom Numbers

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

No, not “random” numbers.

Ranom digits are denoted by uppercase Latin letters from A to E. Moreover, the value of the letter A is 1, B is 10, C is 100, D is 1000, E is 10000.

A Ranom number is a sequence of Ranom digits. The value of the Ranom number is calculated as follows: the values of all digits are summed up, but some digits are taken with negative signs: a digit is taken with negative sign if there is a digit with a strictly greater value to the right of it (not necessarily immediately after it); otherwise, that digit is taken with a positive sign.

For example, the value of the Ranom number DAAABDCA is  $1000 - 1 - 1 - 1 - 10 + 1000 + 100 + 1 = 2088$ .

You are given a Ranom number. You can change no more than one digit in it. Calculate the maximum possible value of the resulting number.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int val[] = {1, 10, 100, 1000, 10000};
5 void solve() {
6     string s;
7     cin >> s;
8     int n = s.size();
9     vector<array<int, 5>> dp(n + 1);
10    for (int i = 0; i < n; i++) {
11        int x = s[i] - 'A';
12        for (int j = 0; j < 5; j++) {
13            dp[i + 1][j] = dp[i][max(x, j)] + (x < j ? -1 : 1) * val[x];
14        }
15    }
16    int ans = -1E9;
17    int mx = 0;
18    int res = 0;
19    for (int i = n - 1; i >= 0; i--) {
```

```

20         for (int j = 0; j < 5; j++) {
21             ans = max(ans, res + (j < mx ? -1 : 1) * val[j] + dp[i][max(mx, j)]);
22         }
23         int x = s[i] - 'A';
24         if (x < mx) {
25             res -= val[x];
26         } else {
27             res += val[x];
28             mx = x;
29         }
30     }
31     cout << ans << "\n";
32 }
33 int main() {
34     ios::sync_with_stdio(false);
35     cin.tie(nullptr);
36     int t;
37     cin >> t;
38     while (t--) {
39         solve();
40     }
41     return 0;
42 }
```

### 335: Ksyusha and Chinchilla

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Ksyusha has a pet chinchilla, a tree on  $n$  vertices and huge scissors. A tree is a connected graph without cycles. During a boring physics lesson Ksyusha thought about how to entertain her pet.

Chinchillas like to play with branches. A branch is a tree of 3 vertices.

A cut is the removal of some (not yet cut) edge in the tree. Ksyusha has plenty of free time, so she can afford to make enough cuts so that the tree splits into branches. In other words, after several (possibly zero) cuts, each vertex must belong to exactly one branch.

Help Ksyusha choose the edges to be cut or tell that it is impossible.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
```

```
7     vector<vector<pair<int, int>>> adj(n);
8     for (int i = 1; i < n; i++) {
9         int u, v;
10        cin >> u >> v;
11        u--, v--;
12        adj[u].emplace_back(v, i);
13        adj[v].emplace_back(u, i);
14    }
15    vector<int> ans;
16    function<int(int, int)> dfs = [&](int x, int p) {
17        int cnt1 = 0, cnt2 = 0;
18        for (auto [y, i] : adj[x]) {
19            if (y == p) {
20                continue;
21            }
22            int v = dfs(y, x);
23            if (v == -1) {
24                return -1;
25            }
26            if (!v) {
27                ans.push_back(i);
28            } else if (v == 1) {
29                cnt1++;
30            } else {
31                cnt2++;
32            }
33        }
34        if (!cnt1 && !cnt2) {
35            return 1;
36        } else if (cnt2 == 1 && !cnt1) {
37            return 0;
38        } else if (cnt2) {
39            return -1;
40        } else if (cnt1 == 1) {
41            return 2;
42        } else if (cnt1 == 2) {
43            return 0;
44        } else {
45            return -1;
46        }
47    };
48    if (dfs(0, -1) != 0) {
49        cout << -1 << "\n";
50        return;
51    }
52    cout << ans.size() << "\n";
53    for (int i = 0; i < ans.size(); i++) {
54        cout << ans[i] << " \n"[i == ans.size() - 1];
55    }
56 }
57 int main() {
58     ios::sync_with_stdio(false);
59     cin.tie(nullptr);
60     int t;
61     cin >> t;
62     while (t--) {
63         solve();
64     }
65     return 0;
66 }
```

### 336: Caesar's Legions

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Gaius Julius Caesar, a famous general, loved to line up his soldiers. Overall the army had  $n_1$  footmen and  $n_2$  horsemen. Caesar thought that an arrangement is not beautiful if somewhere in the line there are strictly more than  $k_1$  footmen standing successively one after another, or there are strictly more than  $k_2$  horsemen standing successively one after another. Find the number of beautiful arrangements of the soldiers.

Note that all  $n_1 + n_2$  warriors should be present at each arrangement. All footmen are considered indistinguishable among themselves. Similarly, all horsemen are considered indistinguishable among themselves.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1E8;
```

```

33  using Z = MInt<P>;
34  int main() {
35      ios::sync_with_stdio(false);
36      cin.tie(nullptr);
37      int n1, n2, k1, k2;
38      cin >> n1 >> n2 >> k1 >> k2;
39      vector<vector<array<Z, 2>>> dp(n1 + 1, vector<vector<array<Z, 2>>>());
40      dp[0][0][0] = dp[0][0][1] = 1;
41      for (int i = 0; i <= n1; i++) {
42          for (int j = 0; j <= n2; j++) {
43              for (int x = 1; x <= min(k1, n1 - i); x++) {
44                  dp[i + x][j][0] += dp[i][j][1];
45              }
46              for (int x = 1; x <= min(k2, n2 - j); x++) {
47                  dp[i][j + x][1] += dp[i][j][0];
48              }
49          }
50      }
51      Z ans = dp[n1][n2][0] + dp[n1][n2][1];
52      cout << ans << "\n";
53      return 0;
54  }

```

### 337: Running Miles

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There is a street with  $n$  sights, with sight number  $i$  being  $i$  miles from the beginning of the street. Sight number  $i$  has beauty  $b_i$ . You want to start your morning jog  $l$  miles and end it  $r$  miles from the beginning of the street. By the time you run, you will see sights you run by (including sights at  $l$  and  $r$  miles from the start). You are interested in the 3 most beautiful sights along your jog, but every mile you run, you get more and more tired.

So choose  $l$  and  $r$ , such that there are at least 3 sights you run by, and the sum of beauties of the 3 most beautiful sights minus the distance in miles you have to run is maximized. More formally, choose  $l$  and  $r$ , such that  $b_{i_1} + b_{i_2} + b_{i_3} - (r - l)$  is maximum possible, where  $i_1, i_2, i_3$  are the indices of the three maximum elements in range  $[l, r]$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int inf = 1E9;
5 void solve() {

```

```

6     int n;
7     cin >> n;
8     vector<int> a(n);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11     }
12     array<int, 4> dp{-inf};
13     dp[0] = 0;
14     for (int i = 0; i < n; i++) {
15         dp[1] -= 1;
16         dp[2] -= 1;
17         for (int j = 3; j; j--) {
18             dp[j] = max(dp[j], dp[j - 1] + a[i]);
19         }
20     }
21     cout << dp[3] << "\n";
22 }
23 int main() {
24     ios::sync_with_stdio(false);
25     cin.tie(nullptr);
26     int t;
27     cin >> t;
28     while (t--) {
29         solve();
30     }
31     return 0;
32 }
```

### 338: Don't Blame Me

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Sadly, the problem setter couldn't think of an interesting story, thus he just asks you to solve the following problem.

Given an array  $a$  consisting of  $n$  positive integers, count the number of non-empty subsequences for which the bitwise AND of the elements in the subsequence has exactly  $k$  set bits in its binary representation. The answer may be large, so output it modulo  $10^9 + 7$ .

Recall that the subsequence of an array  $a$  is a sequence that can be obtained from  $a$  by removing some (possibly, zero) elements. For example,  $[1, 2, 3]$ ,  $[3]$ ,  $[1, 3]$  are subsequences of  $[1, 2, 3]$ , but  $[3, 2]$  and  $[4, 5, 6]$  are not.

Note that AND represents the bitwise AND operation.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
```

```

2  using namespace std;
3  using i64 = long long;
4  template<class T>
5  constexpr T power(T a, i64 b) {
6      T res = 1;
7      for (; b; b /= 2, a *= a) {
8          if (b % 2) {
9              res *= a;
10         }
11     }
12     return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 void solve() {
35     int n, k;
36     cin >> n >> k;
37     Z f[64] {};
38     for (int i = 0; i < n; i++) {
39         int x;
40         cin >> x;
41         f[x] += 1;
42     }
43     for (int i = 1; i < 64; i *= 2) {
44         for (int j = 0; j < 64; j += 2 * i) {
45             for (int k = 0; k < i; k++) {
46                 f[j + k] += f[i + j + k];
47             }
48         }
49     }
50     for (int i = 0; i < 64; i++) {
51         f[i] = power(Z(2), int(f[i])) - 1;
52     }
53     for (int i = 1; i < 64; i *= 2) {
54         for (int j = 0; j < 64; j += 2 * i) {
55             for (int k = 0; k < i; k++) {
56                 f[j + k] -= f[i + j + k];
57             }
58         }
59     }
60     Z ans = 0;
61     for (int i = 0; i < 64; i++) {
62         if (__builtin_popcount(i) == k) {
63             ans += f[i];
64         }
65     }

```

```

66     cout << ans << "\n";
67 }
68 int main() {
69     ios::sync_with_stdio(false);
70     cin.tie(nullptr);
71     int t;
72     cin >> t;
73     while (t--) {
74         solve();
75     }
76     return 0;
77 }
```

### 339: Xenia and Weights

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Xenia has a set of weights and pan scales. Each weight has an integer weight from 1 to 10 kilos. Xenia is going to play with scales and weights a little. For this, she puts weights on the scalepans, one by one. The first weight goes on the left scalepan, the second weight goes on the right scalepan, the third one goes on the left scalepan, the fourth one goes on the right scalepan and so on. Xenia wants to put the total of  $m$  weights on the scalepans.

Simply putting weights on the scales is not interesting, so Xenia has set some rules. First, she does not put on the scales two consecutive weights of the same weight. That is, the weight that goes  $i$ -th should be different from the  $(i + 1)$ -th weight for any  $i$  ( $1 \leq i < m$ ). Second, every time Xenia puts a weight on some scalepan, she wants this scalepan to outweigh the other one. That is, the sum of the weights on the corresponding scalepan must be strictly greater than the sum on the other pan.

You are given all types of weights available for Xenia. You can assume that the girl has an infinite number of weights of each specified type. Your task is to help Xenia lay  $m$  weights on the scales or to say that it can't be done.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     string s;
```

```

8     int m;
9     cin >> s >> m;
10    vector dp(m + 1, vector(11, vector(10, -1)));
11    dp[0][0][0] = dp[0][0][1] = 0;
12    for (int i = 0; i < m; i++) {
13        for (int j = 0; j <= 10; j++) {
14            for (int x = 0; x < 10; x++) {
15                for (int y = 0; y < 10; y++) {
16                    if (x != y && s[y] == '1' && y + 1 - j > 0 && dp[i][j][x] != -1) {
17                        dp[i + 1][y + 1 - j][y] = x;
18                    }
19                }
20            }
21        }
22    }
23    for (int j = 0; j <= 10; j++) {
24        for (int x = 0; x < 10; x++) {
25            if (dp[m][j][x] != -1) {
26                cout << "YES\n";
27                vector<int> ans;
28                for (int i = m; i; i--) {
29                    ans.push_back(x + 1);
30                    int y = dp[i][j][x];
31                    j = x + 1 - j;
32                    x = y;
33                }
34                reverse(ans.begin(), ans.end());
35                for (int i = 0; i < m; i++) {
36                    cout << ans[i] << " \n"[i == m - 1];
37                }
38            }
39        }
40    }
41    cout << "NO\n";
42    return 0;
43 }
44 }
```

### 340: Phone Talks

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Cool J has recently become a businessman Mr. Jackson, and he has to make a lot of phone calls now. Today he has  $n$  calls planned. For each call we know the moment  $t_i$  (in seconds since the start of the day) when it is scheduled to start and its duration  $d_i$  (in seconds). All  $t_i$  are different. Mr. Jackson is a very important person, so he never dials anybody himself, all calls will be incoming.

Mr. Jackson isn't Caesar and he can't do several things at once. If somebody calls him while he hasn't finished the previous conversation, Mr. Jackson puts the new call on hold in the queue. In this case immediately after the end of the current call Mr. Jackson takes the earliest incoming call from the queue and starts the conversation. If Mr. Jackson started the call at the second  $t$ , and the call continues

for  $d$  seconds, then Mr. Jackson is busy at seconds  $t, t + 1, \dots, t + d - 1$ , and he can start a new call at second  $t + d$ . Note that if Mr. Jackson is not busy talking when somebody calls, he can't put this call on hold.

Mr. Jackson isn't Napoleon either, he likes to sleep. So sometimes he allows himself the luxury of ignoring a call, as if it never was scheduled. He can ignore at most  $k$  calls. Note that a call which comes while he is busy talking can be ignored as well.

What is the maximum number of seconds Mr. Jackson can sleep today, assuming that he can choose an arbitrary continuous time segment from the current day (that is, with seconds from the 1-st to the 86400-th, inclusive) when he is not busy talking?

Note that some calls can be continued or postponed to the next day or even later. However, the interval for sleep should be completely within the current day.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int N = 86400;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, k;
9     cin >> n >> k;
10    vector<int> t(n), d(n);
11    for (int i = 0; i < n; i++) {
12        cin >> t[i] >> d[i];
13    }
14    int ans = n ? t[0] - 1 : N;
15    vector<int> dp(k+1);
16    for (int i = 0; i < n; i++) {
17        for (int j = k; j >= 0; j--) {
18            if (j < k) {
19                dp[j+1] = min(dp[j+1], dp[j]);
20            }
21            dp[j] = max(dp[j]+1, t[i]) + d[i] - 1;
22        }
23        ans = max(ans, (i < n-1 ? t[i+1]-1 : N) - dp[k]);
24    }
25    cout << ans << "\n";
26    return 0;
27 }
```

## graphs

### 341: Time Travel

- Time limit: 2 seconds

- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Berland is a country with ancient history, where roads were built and destroyed for centuries. It is known that there always were  $n$  cities in Berland. You also have records of  $t$  key moments in the history of the country, numbered from 1 to  $t$ . Each record contains a list of bidirectional roads between some pairs of cities, which could be used for travel in Berland at a specific moment in time.

You have discovered a time machine that transports you between key moments. Unfortunately, you cannot choose what point in time to end up at, but you know the order consisting of  $k$  moments in time  $a_i$ , in which the machine will transport you. Since there is little time between the travels, when you find yourself in the next key moment in time (including after the last time travel), you can travel on at most one existing road at that moment, coming out from the city you were in before time travel.

Currently, you are in city 1, and the time machine has already transported you to moment  $a_1$ . You want to reach city  $n$  as quickly as possible. Determine the minimum number of time travels, including the first one, that you need to make in order to reach city  $n$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, t;
8     cin >> n >> t;
9     vector<vector<pair<int, int>>> adj(n);
10    for (int i = 0; i < t; i++) {
11        int m;
12        cin >> m;
13        for (int j = 0; j < m; j++) {
14            int x, y;
15            cin >> x >> y;
16            x--, y--;
17            adj[x].emplace_back(y, i);
18            adj[y].emplace_back(x, i);
19        }
20    }
21    int k;
22    cin >> k;
23    vector<int> a(k);
24    vector<vector<int>> pos(t);
25    for (int i = 0; i < k; i++) {
26        cin >> a[i];
27        a[i]--;
28        pos[a[i]].push_back(i);
29    }

```

```

30     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> q;
31     q.emplace(0, 0);
32     vector dis(n, -1);
33     while (!q.empty()) {
34         auto [d, x] = q.top();
35         q.pop();
36         if (dis[x] != -1) {
37             continue;
38         }
39         dis[x] = d;
40         for (auto [y, i] : adj[x]) {
41             auto it = lower_bound(pos[i].begin(), pos[i].end(), d);
42             if (it != pos[i].end()) {
43                 q.emplace(*it + 1, y);
44             }
45         }
46     }
47     cout << dis[n - 1] << "\n";
48     return 0;
49 }
```

### 342: Mad City

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Marcel and Valeriu are in the mad city, which is represented by  $n$  buildings with  $n$  two-way roads between them.

Marcel and Valeriu start at buildings  $a$  and  $b$  respectively. Marcel wants to catch Valeriu, in other words, be in the same building as him or meet on the same road.

During each move, they choose to go to an adjacent building of their current one or stay in the same building. Because Valeriu knows Marcel so well, Valeriu can predict where Marcel will go in the next move. Valeriu can use this information to make his move. They start and end the move at the same time.

It is guaranteed that any pair of buildings is connected by some path and there is at most one road between any pair of buildings.

Assuming both players play optimally, answer if Valeriu has a strategy to indefinitely escape Marcel.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
```

```

5     int n, a, b;
6     cin >> n >> a >> b;
7     a--, b--;
8     vector<vector<int>> adj(n);
9     for (int i = 0; i < n; i++) {
10         int u, v;
11         cin >> u >> v;
12         u--, v--;
13         adj[u].push_back(v);
14         adj[v].push_back(u);
15     }
16     auto bfs = [&](int s) {
17         vector<int> dis(n, -1);
18         dis[s] = 0;
19         queue<int> q;
20         q.push(s);
21         while (!q.empty()) {
22             int x = q.front();
23             q.pop();
24             for (auto y : adj[x]) {
25                 if (dis[y] == -1) {
26                     dis[y] = dis[x] + 1;
27                     q.push(y);
28                 }
29             }
30         }
31         return dis;
32     };
33     auto dis = bfs(b);
34     int u = -1;
35     vector<bool> vis(n);
36     vector<int> par(n), dep(n);
37     auto dfs = [&](auto self, int x) -> void {
38         vis[x] = true;
39         for (auto y : adj[x]) {
40             if (!vis[y]) {
41                 par[y] = x;
42                 dep[y] = dep[x] + 1;
43                 self(self, y);
44             } else if (dep[y] < dep[x] - 1) {
45                 for (int i = x; ; i = par[i]) {
46                     if (u == -1 || dis[i] < dis[u]) {
47                         u = i;
48                     }
49                     if (i == y) {
50                         break;
51                     }
52                 }
53             }
54         }
55     };
56     dfs(dfs, 0);
57     if (bfs(u)[a] > dis[u]) {
58         cout << "YES\n";
59     } else {
60         cout << "NO\n";
61     }
62 }
63 int main() {
64     ios::sync_with_stdio(false);
65     cin.tie(nullptr);
66     int t;
67     cin >> t;
68     while (t--) {

```

```

69     solve();
70 }
71 return 0;
72 }
```

### 343: Gardening Friends

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Two friends, Alisa and Yuki, planted a tree with  $n$  vertices in their garden. A tree is an undirected graph without cycles, loops, or multiple edges. Each edge in this tree has a length of  $k$ . Initially, vertex 1 is the root of the tree.

Alisa and Yuki are growing the tree not just for fun, they want to sell it. The cost of the tree is defined as the maximum distance from the root to a vertex among all vertices of the tree. The distance between two vertices  $u$  and  $v$  is the sum of the lengths of the edges on the path from  $u$  to  $v$ .

The girls took a course in gardening, so they know how to modify the tree. Alisa and Yuki can spend  $c$  coins to shift the root of the tree to one of the neighbors of the current root. This operation can be performed any number of times (possibly zero). Note that the structure of the tree is left unchanged; the only change is which vertex is the root.

The friends want to sell the tree with the maximum profit. The profit is defined as the difference between the cost of the tree and the total cost of operations. The profit is cost of the tree minus the total cost of operations.

Help the girls and find the maximum profit they can get by applying operations to the tree any number of times (possibly zero).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k, c;
6     cin >> n >> k >> c;
7     vector<vector<int>> adj(n);
8     for (int i = 1; i < n; i++) {
9         int u, v;
10        cin >> u >> v;
```

```

11         u--, v--;
12         adj[u].push_back(v);
13         adj[v].push_back(u);
14     }
15     auto bfs = [&](int s) {
16         queue<int> q;
17         vector d(n, -1);
18         d[s] = 0;
19         q.push(s);
20         while (!q.empty()) {
21             int x = q.front();
22             q.pop();
23             for (auto y : adj[x]) {
24                 if (d[y] == -1) {
25                     d[y] = d[x] + 1;
26                     q.push(y);
27                 }
28             }
29         }
30         return d;
31     };
32     auto d0 = bfs(0);
33     int s = max_element(d0.begin(), d0.end()) - d0.begin();
34     auto ds = bfs(s);
35     int t = max_element(ds.begin(), ds.end()) - ds.begin();
36     auto dt = bfs(t);
37     i64 ans = 0;
38     for (int i = 0; i < n; i++) {
39         ans = max(ans, 1LL * k * max(ds[i], dt[i]) - 1LL * c * d0[i]);
40     }
41     cout << ans << "\n";
42 }
43 int main() {
44     ios::sync_with_stdio(false);
45     cin.tie(nullptr);
46     int t;
47     cin >> t;
48     while (t--) {
49         solve();
50     }
51     return 0;
52 }
```

### 344: Igor In the Museum

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Igor is in the museum and he wants to see as many pictures as possible.

Museum can be represented as a rectangular field of  $n$   $m$  cells. Each cell is either empty or impassable. Empty cells are marked with '.', impassable cells are marked with '\*'. Every two adjacent cells of different types (one empty and one impassable) are divided by a wall containing one picture.

At the beginning Igor is in some empty cell. At every moment he can move to any empty cell that share a side with the current one.

For several starting positions you should calculate the maximum number of pictures that Igor can see. Igor is able to see the picture only if he is in the cell adjacent to the wall with this picture. Igor have a lot of time, so he will examine every picture he can see.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int dx[] = {-1, 1, 0, 0};
5 constexpr int dy[] = {0, 0, -1, 1};
6 int main() {
7     ios::sync_with_stdio(false);
8     cin.tie(nullptr);
9     int n, m, k;
10    cin >> n >> m >> k;
11    vector<string> s(n);
12    for (int i = 0; i < n; i++) {
13        cin >> s[i];
14    }
15    vector<vector<int>> vis(n, vector(m, -1));
16    for (int i = 0; i < n; i++) {
17        for (int j = 0; j < m; j++) {
18            if (vis[i][j] == -1 && s[i][j] == '.') {
19                vector<pair<int, int>> q;
20                q.emplace_back(i, j);
21                vis[i][j] = 0;
22                int res = 0;
23                for (int t = 0; t < q.size(); t++) {
24                    auto [x, y] = q[t];
25                    for (int k = 0; k < 4; k++) {
26                        int nx = x + dx[k];
27                        int ny = y + dy[k];
28                        if (vis[nx][ny] == -1 && s[nx][ny] == '.') {
29                            q.emplace_back(nx, ny);
30                            vis[nx][ny] = 0;
31                        }
32                        if (s[nx][ny] == '*') {
33                            res += 1;
34                        }
35                    }
36                }
37                for (auto [x, y] : q) {
38                    vis[x][y] = res;
39                }
40            }
41        }
42    }
43    while (k--) {
44        int x, y;
45        cin >> x >> y;
46        x--, y--;
47        cout << vis[x][y] << "\n";
48    }
49    return 0;
50 }
```

### 345: Roads not only in Berland

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Berland Government decided to improve relations with neighboring countries. First of all, it was decided to build new roads so that from each city of Berland and neighboring countries it became possible to reach all the others. There are  $n$  cities in Berland and neighboring countries in total and exactly  $n - 1$  two-way roads. Because of the recent financial crisis, the Berland Government is strongly pressed for money, so to build a new road it has to close some of the existing ones. Every day it is possible to close one existing road and immediately build a new one. Your task is to determine how many days would be needed to rebuild roads so that from each city it became possible to reach all the others, and to draw a plan of closure of old roads and building of new ones.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n;
9     cin >> n;
10    vector<pair<int, int>> a, b;
11    DSU dsu(n);
12    for (int i = 1; i < n; i++) {
13        int u, v;
14        cin >> u >> v;
15        u--, v--;
16        if (!dsu.merge(u, v)) {
17            a.push_back({u + 1, v + 1});
18        }
19    }
20    for (int i = 0; i < n; i++) {
21        if (dsu.merge(0, i)) {
22            b.push_back({1, i + 1});
23        }
24    }
25    cout << a.size() << "\n";
26    for (int i = 0; i < a.size(); i++) {
27        cout << a[i].first << " " << a[i].second << " " << b[i].first << " " << b[i].second << "\n";
28    }
29    return 0;
30 }
```

### 346: Roads in Berland

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  cities numbered from 1 to  $n$  in Berland. Some of them are connected by two-way roads. Each road has its own length - an integer number from 1 to 1000. It is known that from each city it is possible to get to any other city by existing roads. Also for each pair of cities it is known the shortest distance between them. Berland Government plans to build  $k$  new roads. For each of the planned road it is known its length, and what cities it will connect. To control the correctness of the construction of new roads, after the opening of another road Berland government wants to check the sum of the shortest distances between all pairs of cities. Help them - for a given matrix of shortest distances on the old roads and plans of all new roads, find out how the sum of the shortest distances between all pairs of cities changes after construction of each road.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector d(n, vector<int>(n));
10    for (int i = 0; i < n; i++) {
11        for (int j = 0; j < n; j++) {
12            cin >> d[i][j];
13        }
14    }
15    int k;
16    cin >> k;
17    while (k--) {
18        int a, b, c;
19        cin >> a >> b >> c;
20        a--, b--;
21        i64 ans = 0;
22        for (int i = 0; i < n; i++) {
23            for (int j = 0; j < n; j++) {
24                d[i][j] = min({d[i][j], d[i][a] + c + d[b][j], d[i][b] + c + d[a][j]});
25                if (i < j) {
26                    ans += d[i][j];
27                }
28            }
29        }
30        cout << ans << "\n";
31    }
}

```

```

32     return 0;
33 }
```

### 347: System Administrator

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Bob got a job as a system administrator in X corporation. His first task was to connect  $n$  servers with the help of  $m$  two-way direct connection so that it becomes possible to transmit data from one server to any other server via these connections. Each direct connection has to link two different servers, each pair of servers should have at most one direct connection. Y corporation, a business rival of X corporation, made Bob an offer that he couldn't refuse: Bob was asked to connect the servers in such a way, that when server with index  $v$  fails, the transmission of data between some other two servers becomes impossible, i.e. the system stops being connected. Help Bob connect the servers.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m, v;
8     cin >> n >> m >> v;
9     v--;
10    if (m < n - 1 || m > 1 + (n - 1) * (n - 2) / 2) {
11        cout << -1 << "\n";
12        return 0;
13    }
14    vector<pair<int, int>> ans;
15    for (int i = 0; i < n; i++) {
16        if (v != i) {
17            ans.emplace_back(v, i);
18        }
19    }
20    int x = v ? 0 : 1;
21    for (int i = 0; i < n && ans.size() < m; i++) {
22        for (int j = i + 1; j < n && ans.size() < m; j++) {
23            if (i != v && j != v && i != x && j != x) {
24                ans.emplace_back(i, j);
25            }
26        }
27    }
28    for (auto [x, y] : ans) {
29        cout << x + 1 << " " << y + 1 << "\n";
```

```

30     }
31     return 0;
32 }
```

**348: Dijkstra?**

- Time limit: 1 second
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

You are given a weighted undirected graph. The vertices are enumerated from 1 to n. Your task is to find the shortest path between the vertex 1 and the vertex n.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<vector<pair<int, int>>> adj(n);
10    for (int i = 0; i < m; i++) {
11        int u, v, w;
12        cin >> u >> v >> w;
13        u--;
14        v--;
15        adj[u].emplace_back(v, w);
16        adj[v].emplace_back(u, w);
17    }
18    vector<i64> dis(n, -1);
19    vector<int> pre(n, -1);
20    priority_queue<tuple<i64, int, int>> h;
21    h.emplace(0, 0, -1);
22    while (!h.empty()) {
23        auto [d, x, t] = h.top();
24        h.pop();
25        if (dis[x] != -1) {
26            continue;
27        }
28        dis[x] = -d;
29        pre[x] = t;
30        for (auto [y, w] : adj[x]) {
31            h.emplace(d - w, y, x);
32        }
33    }
34    if (dis[n - 1] == -1) {
35        cout << -1 << "\n";
36    }
37    vector<int> a;
38    for (int x = n - 1; x != -1; x = pre[x]) {
39        a.push_back(x);
40    }
41    reverse(a.begin(), a.end());
42    for (int x : a) {
43        cout << x << " ";
44    }
45}
```

```

40     }
41     reverse(a.begin(), a.end());
42     for (auto x : a) {
43         cout << x + 1 << " \n"[x == n - 1];
44     }
45     return 0;
46 }
```

### 349: Two Paths

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

As you know, Bob's brother lives in Flatland. In Flatland there are  $n$  cities, connected by  $n - 1$  two-way roads. The cities are numbered from 1 to  $n$ . You can get from one city to another moving along the roads.

The Two Paths company, where Bob's brother works, has won a tender to repair two paths in Flatland. A path is a sequence of different cities, connected sequentially by roads. The company is allowed to choose by itself the paths to repair. The only condition they have to meet is that the two paths shouldn't cross (i.e. shouldn't have common cities).

It is known that the profit, the Two Paths company will get, equals the product of the lengths of the two paths. Let's consider the length of each road equals 1, and the length of a path equals the amount of roads in it. Find the maximum possible profit for the company.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     int ans = 0;
10    vector<vector<int>> adj(n);
11    for (int i = 1; i < n; i++) {
12        int u, v;
13        cin >> u >> v;
14        u--, v--;
15        adj[u].push_back(v);
16        adj[v].push_back(u);
17    }
```

```

18     vector<int> d(n), dia(n);
19     function<void(int, int)> dfs = [&](int x, int p) {
20         d[x] = dia[x] = 0;
21         for (auto y : adj[x]) {
22             if (y != p) {
23                 dfs(y, x);
24                 dia[x] = max({dia[x], dia[y], d[x] + d[y] + 1});
25                 d[x] = max(d[x], d[y] + 1);
26             }
27         }
28     };
29     for (auto u = 0; u < n; u++) {
30         for (auto v : adj[u]) {
31             if (u < v) {
32                 dfs(u, v);
33                 dfs(v, u);
34                 ans = max(ans, dia[u] * dia[v]);
35             }
36         }
37     }
38     cout << ans << "\n";
39     return 0;
40 }
```

### 350: A Wide, Wide Graph

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a tree (a connected graph without cycles) with  $n$  vertices.

Consider a fixed integer  $k$ . Then, the graph  $G_k$  is an undirected graph with  $n$  vertices, where an edge between vertices  $u$  and  $v$  exists if and only if the distance between vertices  $u$  and  $v$  in the given tree is at least  $k$ .

For each  $k$  from 1 to  $n$ , print the number of connected components in the graph  $G_k$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<vector<int>> adj(n);
10    for (int i = 1; i < n; i++) {
```

```

11     int u, v;
12     cin >> u >> v;
13     u--, v--;
14     adj[u].push_back(v);
15     adj[v].push_back(u);
16 }
17 vector<int> d(n, -1);
18 auto bfs = [&](int x) {
19     queue<int> q;
20     d.assign(n, -1);
21     q.push(x);
22     d[x] = 0;
23     while (!q.empty()) {
24         int x = q.front();
25         q.pop();
26         for (auto y : adj[x]) {
27             if (d[y] == -1) {
28                 d[y] = d[x] + 1;
29                 q.push(y);
30             }
31         }
32     }
33     return max_element(d.begin(), d.end()) - d.begin();
34 };
35 int x = bfs(0);
36 int y = bfs(x);
37 auto dx = d;
38 bfs(y);
39 auto dy = d;
40 vector<int> ans(n + 1);
41 for (int i = 0; i < n; i++) {
42     ans[max(dx[i], dy[i]) + 1] += 1;
43 }
44 ans[0] += 1;
45 ans[dx[y] + 1] -= 1;
46 for (int i = 1; i <= n; i++) {
47     ans[i] += ans[i - 1];
48 }
49 for (int i = 1; i <= n; i++) {
50     cout << ans[i] << " \n"[i == n];
51 }
52 return 0;
53 }

```

### 351: Directed Roads

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

ZS the Coder and Chris the Baboon has explored Udayland for quite some time. They realize that it consists of  $n$  towns numbered from 1 to  $n$ .

There are  $n$  directed roads in the Udayland.  $i$ -th of them goes from town  $i$  to some other town  $a_i$  ( $a_i \neq i$ ). ZS the Coder can flip the direction of any road in Udayland, i.e. if it goes from town A to town B before

the flip, it will go from town B to town A after.

ZS the Coder considers the roads in the Udayland confusing, if there is a sequence of distinct towns  $A_1, A_2, \dots, A_k$  ( $k > 1$ ) such that for every  $1 \leq i < k$  there is a road from town  $A_i$  to town  $A_{i+1}$  and another road from town  $A_k$  to town  $A_1$ . In other words, the roads are confusing if some of them form a directed cycle of some towns.

Now ZS the Coder wonders how many sets of roads (there are  $2^n$  variants) in initial configuration can he choose to flip such that after flipping each road in the set exactly once, the resulting network will not be confusing.

Note that it is allowed that after the flipping there are more than one directed road from some town and possibly some towns with no roads leading out of it, or multiple roads between any pair of cities.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int n;
38     cin >> n;
39     vector<int> a(n);
40     for (int i = 0; i < n; i++) {

```

```

41         cin >> a[i];
42         a[i]--;
43     }
44     Z ans = power(Z(2), n);
45     vector<int> vis(n, -1);
46     for (int i = 0; i < n; i++) {
47         int j = i;
48         while (vis[j] == -1) {
49             vis[j] = i;
50             j = a[j];
51         }
52         if (vis[j] == i) {
53             int v = j;
54             int len = 0;
55             do {
56                 j = a[j];
57                 len += 1;
58             } while (v != j);
59             ans /= power(Z(2), len);
60             ans *= power(Z(2), len) - 2;
61         }
62     }
63     cout << ans << "\n";
64     return 0;
65 }
```

### 352: Game on Axis

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  points  $1, 2, \dots, n$ , each point  $i$  has a number  $a_i$  on it. You're playing a game on them. Initially, you are at point 1. When you are at point  $i$ , take following steps:

If  $1 \leq i \leq n$ , go to  $i + a_i$ ,

Otherwise, the game ends.

Before the game begins, you can choose two integers  $x$  and  $y$  satisfying  $1 \leq x \leq n$ ,  $-n \leq y \leq n$  and replace  $a_x$  with  $y$  (set  $a_x := y$ ). Find the number of distinct pairs  $(x, y)$  such that the game that you start after making the change ends in a finite number of steps.

Notice that you do not have to satisfy  $a_x \neq y$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 1E18;
```

```

5  # struct DSU;
6  void solve() {
7      int n;
8      cin >> n;
9      vector<int> a(n);
10     for (int i = 0; i < n; i++) {
11         cin >> a[i];
12         a[i] += i;
13     }
14     int x = 0;
15     i64 ans = 0;
16     DSU dsu(n);
17     vector<bool> vis(n);
18     vector<int> b;
19     while (0 <= x && x < n && !vis[x]) {
20         b.push_back(x);
21         vis[x] = true;
22         x = a[x];
23     }
24     vector<bool> cyc(n);
25     for (int i = 0; i < n; i++) {
26         if (0 <= a[i] && a[i] < n && !vis[i]) {
27             if (!dsu.merge(i, a[i])) {
28                 cyc[i] = true;
29             }
30         }
31     }
32     int sum = 0;
33     for (int i = 0; i < n; i++) {
34         if (cyc[i]) {
35             sum += dsu.size(i);
36         }
37     }
38     if (0 <= x && x < n) {
39         for (int i = 0; i < n; i++) {
40             if (0 <= a[i] && a[i] < n && vis[i]) {
41                 if (!dsu.merge(i, a[i])) {
42                     cyc[i] = true;
43                 }
44             }
45         }
46         sum += dsu.size(0);
47         ans = 1LL * b.size() * sum;
48         ans += 1LL * (n - b.size()) * (2 * n + 1);
49     } else {
50         for (auto x : b) {
51             ans += sum + dsu.size(0);
52             if (0 <= a[x] && a[x] < n) {
53                 dsu.merge(x, a[x]);
54             }
55         }
56     }
57     ans = 1LL * n * (2 * n + 1) - ans;
58     cout << ans << "\n";
59 }
60 int main() {
61     ios::sync_with_stdio(false);
62     cin.tie(nullptr);
63     int t;
64     cin >> t;
65     while (t--) {
66         solve();
67     }
68     return 0;

```

```
69 }
```

### 353: Friendly Spiders

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Mars is home to an unusual species of spiders - Binary spiders.

Right now, Martian scientists are observing a colony of  $n$  spiders, the  $i$ -th of which has  $a_i$  legs.

Some of the spiders are friends with each other. Namely, the  $i$ -th and  $j$ -th spiders are friends if  $\gcd(a_i, a_j) \neq 1$ , i. e., there is some integer  $k \geq 2$  such that  $a_i$  and  $a_j$  are simultaneously divided by  $k$  without a remainder. Here  $\gcd(x, y)$  denotes the greatest common divisor (GCD) of integers  $x$  and  $y$ .

Scientists have discovered that spiders can send messages. If two spiders are friends, then they can transmit a message directly in one second. Otherwise, the spider must pass the message to his friend, who in turn must pass the message to his friend, and so on until the message reaches the recipient.

Let's look at an example.

Suppose a spider with eight legs wants to send a message to a spider with 15 legs. He can't do it directly, because  $\gcd(8, 15) = 1$ . But he can send a message through the spider with six legs because  $\gcd(8, 6) = 2$  and  $\gcd(6, 15) = 3$ . Thus, the message will arrive in two seconds.

Right now, scientists are observing how the  $s$ -th spider wants to send a message to the  $t$ -th spider. The researchers have a hypothesis that spiders always transmit messages optimally. For this reason, scientists would need a program that could calculate the minimum time to send a message and also deduce one of the optimal routes.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     int m = 0;
10    vector<int> a(n);
```

```

11     for (int i = 0; i < n; i++) {
12         cin >> a[i];
13         m = max(m, a[i]);
14     }
15     vector<int> minp(m + 1), primes;
16     for (int i = 2; i <= m; i++) {
17         if (!minp[i]) {
18             minp[i] = i;
19             primes.push_back(i);
20         }
21         for (auto p : primes) {
22             if (i * p > m) break;
23             minp[i * p] = p;
24             if (p == minp[i]) break;
25         }
26     }
27     int s, t;
28     cin >> s >> t;
29     s--, t--;
30     vector<vector<int>> f(m + 1);
31     for (int i = 0; i < n; i++) {
32         for (int x = a[i]; x > 1; x /= minp[x]) {
33             f[minp[x]].push_back(i);
34         }
35     }
36     vector<int> dis(n + m + 1, -1), prev(n + m + 1);
37     queue<tuple<int, int, int>> q;
38     q.emplace(s, 0, -1);
39     while (!q.empty()) {
40         auto [u, d, p] = q.front();
41         q.pop();
42         if (dis[u] != -1) continue;
43         dis[u] = d;
44         prev[u] = p;
45         if (u < n) {
46             for (int x = a[u]; x > 1; x /= minp[x]) {
47                 q.emplace(n + minp[x], d + 1, u);
48             }
49         } else {
50             for (auto x : f[u - n]) {
51                 q.emplace(x, d + 1, u);
52             }
53         }
54     }
55     if (dis[t] == -1) {
56         cout << -1 << "\n";
57         return 0;
58     }
59     vector<int> ans;
60     for (int i = t; i != -1; i = prev[i]) {
61         if (i < n) ans.push_back(i);
62     }
63     reverse(ans.begin(), ans.end());
64     cout << ans.size() << "\n";
65     for (int i = 0; i < ans.size(); i++) {
66         cout << ans[i] + 1 << " \n"[i == ans.size() - 1];
67     }
68     return 0;
69 }

```

### 354: SlavicG's Favorite Problem

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a weighted tree with  $n$  vertices. Recall that a tree is a connected graph without any cycles. A weighted tree is a tree in which each edge has a certain weight. The tree is undirected, it doesn't have a root.

Since trees bore you, you decided to challenge yourself and play a game on the given tree.

In a move, you can travel from a node to one of its neighbors (another node it has a direct edge with).

You start with a variable  $x$  which is initially equal to 0. When you pass through edge  $i$ ,  $x$  changes its value to  $x \text{ XOR } w_i$  (where  $w_i$  is the weight of the  $i$ -th edge).

Your task is to go from vertex  $a$  to vertex  $b$ , but you are allowed to enter node  $b$  if and only if after traveling to it, the value of  $x$  will become 0. In other words, you can travel to node  $b$  only by using an edge  $i$  such that  $x \text{ XOR } w_i = 0$ . Once you enter node  $b$  the game ends and you win.

Additionally, you can teleport at most once at any point in time to any vertex except vertex  $b$ . You can teleport from any vertex, even from  $a$ .

Answer with "YES" if you can reach vertex  $b$  from  $a$ , and "NO" otherwise.

Note that XOR represents the bitwise XOR operation.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, a, b;
6     cin >> n >> a >> b;
7     a--, b--;
8     vector<vector<array<int, 2>>> adj(n);
9     for (int i = 1; i < n; i++) {
10         int x, y, z;
11         cin >> x >> y >> z;
12         x--, y--;
13         adj[x].push_back({y, z});
14         adj[y].push_back({x, z});
15     }
16     vector<int> f(n);
17     set<int> s;
18     function<void(int, int, int)> dfs = [&](int x, int t, int p) {
19         if (x != b) {
20             s.insert(f[x]);

```

```

21         }
22         for (auto [y, z] : adj[x]) {
23             if (y == p || y == t) {
24                 continue;
25             }
26             f[y] = f[x] ^ z;
27             dfs(y, t, x);
28         }
29     };
30     f[a] = 0;
31     dfs(a, b, -1);
32     auto t = s;
33     s.clear();
34     f[b] = 0;
35     dfs(b, -1, -1);
36     for (auto x : s) {
37         if (t.count(x)) {
38             cout << "YES\n";
39             return;
40         }
41     }
42     cout << "NO\n";
43 }
44 int main() {
45     ios::sync_with_stdio(false);
46     cin.tie(nullptr);
47     int t;
48     cin >> t;
49     while (t--) {
50         solve();
51     }
52     return 0;
53 }
```

## greedy

### 355: Bicycles

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

All of Slavic's friends are planning to travel from the place where they live to a party using their bikes. And they all have a bike except Slavic. There are  $n$  cities through which they can travel. They all live in the city 1 and want to go to the party located in the city  $n$ . The map of cities can be seen as an undirected graph with  $n$  nodes and  $m$  edges. Edge  $i$  connects cities  $u_i$  and  $v_i$  and has a length of  $w_i$ .

Slavic doesn't have a bike, but what he has is money. Every city has exactly one bike for sale. The bike in the  $i$ -th city has a slowness factor of  $s_i$ . Once Slavic buys a bike, he can use it whenever to travel from the city he is currently in to any neighboring city, by taking  $w_i \cdot s_j$  time, considering he is traversing edge  $i$  using a bike  $j$  he owns.

Slavic can buy as many bikes as he wants as money isn't a problem for him. Since Slavic hates traveling by bike, he wants to get from his place to the party in the shortest amount of time possible. And, since his informatics skills are quite rusty, he asks you for help.

What's the shortest amount of time required for Slavic to travel from city 1 to city  $n$ ? Slavic can't travel without a bike. It is guaranteed that it is possible for Slavic to travel from city 1 to any other city.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<vector<pair<int, int>>> adj(n);
8     for (int i = 0; i < m; i++) {
9         int u, v, w;
10        cin >> u >> v >> w;
11        u--, v--;
12        adj[u].emplace_back(v, w);
13        adj[v].emplace_back(u, w);
14    }
15    vector<int> s(n);
16    for (int i = 0; i < n; i++) {
17        cin >> s[i];
18    }
19    vector<i64> dis(n * n, -1LL);
20    priority_queue<pair<i64, int>, vector<pair<i64, int>>, greater<>> q;
21    q.emplace(0, 0);
22    while (!q.empty()) {
23        auto [d, x] = q.top();
24        q.pop();
25        if (dis[x] != -1) {
26            continue;
27        }
28        dis[x] = d;
29        int c = x / n, b = x % n;
30        if (c == n - 1) {
31            cout << d << "\n";
32            return;
33        }
34        q.emplace(d, c * n + c);
35        for (auto [y, w] : adj[c]) {
36            q.emplace(d + s[b] * w, y * n + b);
37        }
38    }
39 }
40 int main() {
41     ios::sync_with_stdio(false);
42     cin.tie(nullptr);
43     int t;
44     cin >> t;
45     while (t--) {
46         solve();
47     }
48     return 0;
49 }
```

### 356: Split Plus K

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  positive integers  $a_1, a_2, \dots, a_n$  on a blackboard. You are also given a positive integer  $k$ . You can perform the following operation some (possibly 0) times:

choose a number  $x$  on the blackboard;

erase one occurrence of  $x$ ;

write two positive integers  $y, z$  such that  $y + z = x + k$  on the blackboard.

Is it possible to make all the numbers on the blackboard equal? If yes, what is the minimum number of operations you need?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     i64 k;
7     cin >> n >> k;
8     vector<i64> a(n);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11     }
12     i64 g = abs(a[0] - k);
13     for (int i = 1; i < n; i++) {
14         g = gcd(g, a[i] - a[i - 1]);
15     }
16     i64 v = a[0] < k ? k - g : k + g;
17     i64 ans = 0;
18     for (int i = 0; i < n; i++) {
19         if (g == 0) {
20             if (a[i] != v) {
21                 ans = -1;
22                 break;
23             }
24         } else {
25             i64 res = (a[i] - v) / (v - k);
26             if (res < 0) {
27                 ans = -1;
28                 break;
29             }
30             ans += res;
31         }
32     }
33     cout << ans << "\n";
34 }
```

```
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int t;
39     cin >> t;
40     while (t--) {
41         solve();
42     }
43     return 0;
44 }
```

### 357: Programming Competition

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

BerSoft is the biggest IT corporation in Berland. There are  $n$  employees at BerSoft company, numbered from 1 to  $n$ .

The first employee is the head of the company, and he does not have any superiors. Every other employee  $i$  has exactly one direct superior  $p_i$ .

Employee  $x$  is considered to be a superior (direct or indirect) of employee  $y$  if one of the following conditions holds:

employee  $x$  is the direct superior of employee  $y$ ;

employee  $x$  is a superior of the direct superior of employee  $y$ .

The structure of BerSoft is organized in such a way that the head of the company is superior of every employee.

A programming competition is going to be held soon. Two-person teams should be created for this purpose. However, if one employee in a team is the superior of another, they are uncomfortable together. So, teams of two people should be created so that no one is the superior of the other. Note that no employee can participate in more than one team.

Your task is to calculate the maximum possible number of teams according to the aforementioned rules.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> p(n);
8     vector<vector<int>> adj(n);
9     for (int i = 1; i < n; i++) {
10         cin >> p[i];
11         p[i]--;
12         adj[p[i]].push_back(i);
13     }
14     vector<int> siz(n);
15     auto dfs1 = [&](auto self, int x) -> void {
16         siz[x] = 1;
17         for (auto y : adj[x]) {
18             self(self, y);
19             siz[x] += siz[y];
20         }
21     };
22     dfs1(dfs1, 0);
23     auto dfs = [&](auto self, int x) -> int {
24         int max = 0;
25         for (auto y : adj[x]) {
26             max = max(max, siz[y]);
27         }
28         if (max * 2 > siz[x] - 1) {
29             for (auto y : adj[x]) {
30                 if (siz[y] == max) {
31                     int v = self(self, y);
32                     return v + min((siz[x] - 1 - 2 * v) / 2, siz[x] - 1 - max);
33                 }
34             }
35         } else {
36             return (siz[x] - 1) / 2;
37         }
38     };
39     int ans = dfs(dfs, 0);
40     cout << ans << "\n";
41 }
42 int main() {
43     ios::sync_with_stdio(false);
44     cin.tie(nullptr);
45     int t;
46     cin >> t;
47     while (t--) {
48         solve();
49     }
50     return 0;
51 }
```

### 358: Triangle Construction

- Time limit: 1 second
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

You are given a regular  $N$ -sided polygon. Label one arbitrary side as side 1, then label the next sides in clockwise order as side 2, 3, ...,  $N$ . There are  $A_i$  special points on side  $i$ . These points are positioned such that side  $i$  is divided into  $A_i + 1$  segments with equal length.

For instance, suppose that you have a regular 4-sided polygon, i.e., a square. The following illustration shows how the special points are located within each side when  $A = [3, 1, 4, 6]$ . The uppermost side is labelled as side 1.

You want to create as many non-degenerate triangles as possible while satisfying the following requirements. Each triangle consists of 3 distinct special points (not necessarily from different sides) as its corners. Each special point can only become the corner of at most 1 triangle. All triangles must not intersect with each other.

Determine the maximum number of non-degenerate triangles that you can create.

A triangle is non-degenerate if it has a positive area.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int N;
8     cin >> N;
9     vector<int> A(N);
10    for (int i = 0; i < N; i++) {
11        cin >> A[i];
12    }
13    i64 sum = accumulate(A.begin(), A.end(), 0LL);
14    int max = *max_element(A.begin(), A.end());
15    i64 ans = min(sum / 3, sum - max);
16    cout << ans << "\n";
17    return 0;
18 }
```

### 359: Shift and Reverse

- Time limit: 2.0 s
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Given an array of integers  $a_1, a_2, \dots, a_n$ . You can make two types of operations with this array:

Shift: move the last element of array to the first place, and shift all other elements to the right, so you get the array  $a_n, a_1, a_2, \dots, a_{n-1}$ .

Reverse: reverse the whole array, so you get the array  $a_n, a_{n-1}, \dots, a_1$ .

Your task is to sort the array in non-decreasing order using the minimal number of operations, or say that it is impossible.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    int ans = -1;
12    int f = 0;
13    int p = -1;
14    for (int i = 0; i < n; i++) {
15        if (a[i] > a[(i + 1) % n]) {
16            f += 1;
17            p = i + 1;
18        }
19    }
20    if (f == 0) {
21        ans = 0;
22    } else if (f == 1) {
23        int res = min(p + 2, n - p);
24        if (ans == -1 || ans > res) {
25            ans = res;
26        }
27    }
28    f = 0;
29    for (int i = 0; i < n; i++) {
30        if (a[i] < a[(i + 1) % n]) {
31            f += 1;
32            p = i + 1;
33        }
34    }
35    if (f == 0) {
36        ans = 0;
37    } else if (f == 1) {
38        int res = min(p + 1, n - p + 1);
39        if (ans == -1 || ans > res) {
40            ans = res;
41        }
42    }
43    cout << ans << "\n";
44 }
45 int main() {
46     ios::sync_with_stdio(false);
47     cin.tie(nullptr);
48     int t;
49     cin >> t;
50     while (t--) {
51         solve();

```

```

52     }
53     return 0;
54 }
```

### 360: Yet Another Monster Fight

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vasya is a sorcerer that fights monsters. Again. There are  $n$  monsters standing in a row, the amount of health points of the  $i$ -th monster is  $a_i$ .

Vasya is a very powerful sorcerer who knows many overpowered spells. In this fight, he decided to use a chain lightning spell to defeat all the monsters. Let's see how this spell works.

Firstly, Vasya chooses an index  $i$  of some monster ( $1 \leq i \leq n$ ) and the initial power of the spell  $x$ . Then the spell hits monsters exactly  $n$  times, one hit per monster. The first target of the spell is always the monster  $i$ . For every target except for the first one, the chain lightning will choose a random monster who was not hit by the spell and is adjacent to one of the monsters that already was hit. So, each monster will be hit exactly once. The first monster hit by the spell receives  $x$  damage, the second monster receives  $(x - 1)$  damage, the third receives  $(x - 2)$  damage, and so on.

Vasya wants to show how powerful he is, so he wants to kill all the monsters with a single chain lightning spell. The monster is considered dead if the damage he received is not less than the amount of its health points. On the other hand, Vasya wants to show he doesn't care that much, so he wants to choose the minimum initial power of the spell  $x$  such that it kills all monsters, no matter which monster (among those who can get hit) gets hit on each step.

Of course, Vasya is a sorcerer, but the amount of calculations required to determine the optimal spell setup is way above his possibilities, so you have to help him find the minimum spell power required to kill all the monsters.

Note that Vasya chooses the initial target and the power of the spell, other things should be considered random and Vasya wants to kill all the monsters even in the worst possible scenario.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
```

```

3  using i64 = long long;
4  int main() {
5      ios::sync_with_stdio(false);
6      cin.tie(nullptr);
7      int n;
8      cin >> n;
9      vector<int> a(n);
10     for (int i = 0; i < n; i++) {
11         cin >> a[i];
12     }
13     int ans = 2E9;
14     auto pre = a, suf = a;
15     for (int i = 0; i < n; i++) {
16         pre[i] -= i;
17         if (i) {
18             pre[i] = max(pre[i], pre[i - 1]);
19         }
20     }
21     for (int i = n - 1; i >= 0; i--) {
22         suf[i] += i;
23         if (i < n - 1) {
24             suf[i] = max(suf[i], suf[i + 1]);
25         }
26     }
27     for (int i = 0; i < n; i++) {
28         int res = a[i];
29         if (i > 0) {
30             res = max(res, pre[i - 1] + n - 1);
31         }
32         if (i < n - 1) {
33             res = max(res, suf[i + 1]);
34         }
35         ans = min(ans, res);
36     }
37     cout << ans << "\n";
38     return 0;
39 }
```

### 361: Maximum And Queries (easy version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the easy version of the problem. The only difference between the two versions is the constraint on  $n$  and  $q$ , the memory and time limits. You can make hacks only if all versions of the problem are solved.

Theofanis really likes to play with the bits of numbers. He has an array  $a$  of size  $n$  and an integer  $k$ . He can make at most  $k$  operations in the array. In each operation, he picks a single element and increases it by 1.

He found the maximum bitwise AND that array  $a$  can have after at most  $k$  operations.

Theofanis has put a lot of work into finding this value and was very happy with his result. Unfortunately, Ada, being the evil person that he is, decided to bully him by repeatedly changing the value of  $k$ .

Help Theofanis by calculating the maximum possible bitwise AND for  $q$  different values of  $k$ . Note that queries are independent.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, q;
8     cin >> n >> q;
9     vector<i64> f(1 << 20);
10    vector<int> a(n);
11    for (int i = 0; i < n; i++) {
12        cin >> a[i];
13    }
14    for (int k = 0; k < 20; k++) {
15        vector<i64> s(1 << (19 - k)), c(1 << (19 - k));
16        for (int i = 0; i < n; i++) {
17            if (~a[i] >> k & 1) {
18                s[a[i] >> (k + 1)] += a[i] & ((1 << k) - 1);
19                c[a[i] >> (k + 1)] += 1;
20            }
21        }
22        for (int i = 1; i < (1 << (19 - k)); i *= 2) {
23            for (int j = 0; j < (1 << (19 - k)); j += 2 * i) {
24                for (int l = 0; l < i; l++) {
25                    s[j + l] += s[i + j + l];
26                    c[j + l] += c[i + j + l];
27                }
28            }
29        }
30        for (int x = 0; x < (1 << 20); x++) {
31            if (x >> k & 1) {
32                int y = x >> (k + 1);
33                f[x] += c[y] * (x & ((1 << (k + 1)) - 1)) - s[y];
34            }
35        }
36    }
37    for (int i = (1 << 20) - 2; i >= 0; i--) {
38        f[i] = min(f[i], f[i + 1]);
39    }
40    while (q--) {
41        i64 k;
42        cin >> k;
43        i64 ans = upper_bound(f.begin(), f.end(), k) - f.begin() - 1;
44        if (k >= f.back()) {
45            ans += (k - f.back()) / n;
46        }
47        cout << ans << "\n";
48    }
49    return 0;
50 }
```

### 362: Absolute Beauty

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Kirill has two integer arrays  $a_1, a_2, \dots, a_n$  and  $b_1, b_2, \dots, b_n$  of length  $n$ . He defines the absolute beauty of the array  $b$  as

$$\sum_{i=1}^n |a_i - b_i|.$$

Here,  $|x|$  denotes the absolute value of  $x$ .

Kirill can perform the following operation at most once:

select two indices  $i$  and  $j$  ( $1 \leq i < j \leq n$ ) and swap the values of  $b_i$  and  $b_j$ .

Help him find the maximum possible absolute beauty of the array  $b$  after performing at most one swap.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 1E18;
5 void solve() {
6     int n;
7     cin >> n;
8     vector<int> a(n), b(n);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11     }
12     for (int i = 0; i < n; i++) {
13         cin >> b[i];
14     }
15     i64 ans = 0;
16     i64 mx1 = -inf, mx2 = -inf;
17     for (int i = 0; i < n; i++) {
18         int d = abs(a[i] - b[i]);
19         ans += d;
20         mx1 = max(mx1, 1LL * a[i] + b[i] - d);
21         mx2 = max(mx2, 1LL * -a[i] - b[i] - d);
22     }
23     ans += max(0LL, mx1 + mx2);
24     cout << ans << "\n";
25 }
26 int main() {
27     ios::sync_with_stdio(false);
28     cin.tie(nullptr);
29     int t;
30     cin >> t;
31     while (t--) {

```

```

32     solve();
33 }
34 return 0;
35 }
```

### 363: Neutral Tonality

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  consisting of  $n$  integers, as well as an array  $b$  consisting of  $m$  integers.

Let  $\text{LIS}(c)$  denote the length of the longest increasing subsequence of array  $c$ . For example,  $\text{LIS}([2, 1, 1, 3]) = 2$ ,  $\text{LIS}([1, 7, 9]) = 3$ ,  $\text{LIS}([3, 1, 2, 4]) = 3$ .

You need to insert the numbers  $b_1, b_2, \dots, b_m$  into the array  $a$ , at any positions, in any order. Let the resulting array be  $c_1, c_2, \dots, c_{n+m}$ . You need to choose the positions for insertion in order to minimize  $\text{LIS}(c)$ .

Formally, you need to find an array  $c_1, c_2, \dots, c_{n+m}$  that simultaneously satisfies the following conditions:

The array  $a_1, a_2, \dots, a_n$  is a subsequence of the array  $c_1, c_2, \dots, c_{n+m}$ .

The array  $c_1, c_2, \dots, c_{n+m}$  consists of the numbers  $a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_m$ , possibly rearranged.

The value of  $\text{LIS}(c)$  is the minimum possible among all suitable arrays  $c$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<int> a(n), b(m);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    for (int i = 0; i < m; i++) {
12        cin >> b[i];
13    }
14    vector<int> dp(n), f;
15    for (int i = 0; i < n; i++) {
16        auto it = lower_bound(f.begin(), f.end(), a[i]);
17        dp[i] = it - f.begin() + 1;
18    }
19    cout << dp[n-1] << endl;
20 }
```

```

18     if (it == f.end()) {
19         f.push_back(a[i]);
20     } else {
21         *it = a[i];
22     }
23 }
24 int lis = f.size();
25 vector<int> ans;
26 sort(b.begin(), b.end(), greater());
27 int j = 0;
28 for (int i = 0; i < n; i++) {
29     if (dp[i] == lis) {
30         while (j < m && b[j] >= a[i]) {
31             ans.push_back(b[j++]);
32         }
33     }
34     ans.push_back(a[i]);
35 }
36 while (j < m) {
37     ans.push_back(b[j++]);
38 }
39 for (int i = 0; i < n + m; i++) {
40     cout << ans[i] << " \n"[i == n + m - 1];
41 }
42 }
43 int main() {
44     ios::sync_with_stdio(false);
45     cin.tie(nullptr);
46     int t;
47     cin >> t;
48     while (t--) {
49         solve();
50     }
51     return 0;
52 }
```

### 364: Dances (Hard Version)

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. The only difference is that in this version  $m \leq 10^9$ .

You are given two arrays of integers  $a_1, a_2, \dots, a_n$  and  $b_1, b_2, \dots, b_n$ . Before applying any operations, you can reorder the elements of each array as you wish. Then, in one operation, you will perform both of the following actions, if the arrays are not empty:

Choose any element from array  $a$  and remove it (all remaining elements are shifted to a new array  $a$ ),

Choose any element from array  $b$  and remove it (all remaining elements are shifted to a new array  $b$ ).

Let  $k$  be the final size of both arrays. You need to find the minimum number of operations required to satisfy  $a_i < b_i$  for all  $1 \leq i \leq k$ .

This problem was too easy, so the problem author decided to make it more challenging. You are also given a positive integer  $m$ . Now, you need to find the sum of answers to the problem for  $m$  pairs of arrays  $(c[i], b)$ , where  $1 \leq i \leq m$ . Array  $c[i]$  is obtained from  $a$  as follows:

$c[i]_1 = i$ ,

$c[i]_j = a_j$ , for  $2 \leq j \leq n$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<int> a(n), b(n);
8     for (int i = 1; i < n; i++) {
9         cin >> a[i];
10    }
11    for (int i = 0; i < n; i++) {
12        cin >> b[i];
13    }
14    a[0] = 1;
15    auto a0 = a;
16    sort(a.begin(), a.end());
17    sort(b.begin(), b.end());
18    int ans = 0;
19    for (int i = 0, j = 0; i < n; i++) {
20        while (j < n && a[i] >= b[j]) {
21            j++;
22        }
23        ans = max(ans, j - i);
24    }
25    int x = *ranges::partition_point(ranges::iota_view(1, m + 1),
26                                     [&](int v) {
27                                         a = a0;
28                                         a[0] = v;
29                                         int res = 0;
30                                         sort(a.begin(), a.end());
31                                         for (int i = 0, j = 0; i < n; i++) {
32                                             while (j < n && a[i] >= b[j]) {
33                                                 j++;
34                                             }
35                                             res = max(res, j - i);
36                                         }
37                                         return res == ans;
38                                     });
39    i64 sum = 1LL * m * ans + (m - x + 1);
40    cout << sum << "\n";
41 }
42 int main() {
43     ios::sync_with_stdio(false);
44     cin.tie(nullptr);
45     int t;
46     cin >> t;
47     while (t--) {

```

```

48         solve();
49     }
50     return 0;
51 }
```

### 365: Medium Design

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The array  $a_1, a_2, \dots, a_m$  is initially filled with zeroes. You are given  $n$  pairwise distinct segments  $1 \leq l_i \leq r_i \leq m$ . You have to select an arbitrary subset of these segments (in particular, you may select an empty set). Next, you do the following:

For each  $i = 1, 2, \dots, n$ , if the segment  $(l_i, r_i)$  has been selected to the subset, then for each index  $l_i \leq j \leq r_i$  you increase  $a_j$  by 1 (i. e.  $a_j$  is replaced by  $a_j + 1$ ). If the segment  $(l_i, r_i)$  has not been selected, the array does not change.

Next (after processing all values of  $i = 1, 2, \dots, n$ ), you compute  $\max(a)$  as the maximum value among all elements of  $a$ . Analogously, compute  $\min(a)$  as the minimum value.

Finally, the cost of the selected subset of segments is declared as  $\max(a) - \min(a)$ .

Please, find the maximum cost among all subsets of segments.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<int> l(n), r(n);
8     for (int i = 0; i < n; i++) {
9         cin >> l[i] >> r[i];
10        l[i] -= 1;
11    }
12    vector<pair<int, int>> f;
13    for (int i = 0; i < n; i++) {
14        if (l[i] != 0) {
15            f.emplace_back(l[i], 1);
16            f.emplace_back(r[i], -1);
17        }
18    }
19    int ans = 0;
```

```

20     int cur = 0, lst = 0;
21     sort(f.begin(), f.end());
22     for (auto [x, y] : f) {
23         if (x > lst) {
24             ans = max(ans, cur);
25         }
26         cur += y;
27         lst = x;
28     }
29     if (m > lst) {
30         ans = max(ans, cur);
31     }
32     f.clear();
33     for (int i = 0; i < n; i++) {
34         if (r[i] != m) {
35             f.emplace_back(l[i], 1);
36             f.emplace_back(r[i], -1);
37         }
38     }
39     cur = 0, lst = 0;
40     sort(f.begin(), f.end());
41     for (auto [x, y] : f) {
42         if (x > lst) {
43             ans = max(ans, cur);
44         }
45         cur += y;
46         lst = x;
47     }
48     if (m > lst) {
49         ans = max(ans, cur);
50     }
51     cout << ans << "\n";
52 }
53 int main() {
54     ios::sync_with_stdio(false);
55     cin.tie(nullptr);
56     int t;
57     cin >> t;
58     while (t--) {
59         solve();
60     }
61     return 0;
62 }
```

### 366: Tree XOR

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given a tree with  $n$  vertices labeled from 1 to  $n$ . An integer  $a_i$  is written on vertex  $i$  for  $i = 1, 2, \dots, n$ . You want to make all  $a_i$  equal by performing some (possibly, zero) spells.

Suppose you root the tree at some vertex. On each spell, you can select any vertex  $v$  and any non-negative integer  $c$ . Then for all vertices  $i$  in the subtree<sup>†</sup> of  $v$ , replace  $a_i$  with  $a_i \oplus c$ . The cost of this spell

is  $s \cdot c$ , where  $s$  is the number of vertices in the subtree. Here  $\oplus$  denotes the bitwise XOR operation.

Let  $m_r$  be the minimum possible total cost required to make all  $a_i$  equal, if vertex  $r$  is chosen as the root of the tree. Find  $m_1, m_2, \dots, m_n$ .

<sup>†</sup> Suppose vertex  $r$  is chosen as the root of the tree. Then vertex  $i$  belongs to the subtree of  $v$  if the simple path from  $i$  to  $r$  contains  $v$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<vector<int>> adj(n);
12    for (int i = 1; i < n; i++) {
13        int u, v;
14        cin >> u >> v;
15        u--, v--;
16        adj[u].push_back(v);
17        adj[v].push_back(u);
18    }
19    vector<int> siz(n);
20    vector<i64> add(n);
21    i64 ans = 0;
22    auto dfs1 = [&](auto self, int x, int p) -> void {
23        siz[x] = 1;
24        for (auto y : adj[x]) {
25            if (y == p) {
26                continue;
27            }
28            self(self, y, x);
29            siz[x] += siz[y];
30            ans += 1LL * siz[y] * (a[x] ^ a[y]);
31            add[y] += 1LL * (n - siz[y] * 2) * (a[x] ^ a[y]);
32        }
33    };
34    dfs1(dfs1, 0, -1);
35    auto dfs2 = [&](auto self, int x, int p) -> void {
36        for (auto y : adj[x]) {
37            if (y == p) {
38                continue;
39            }
40            add[y] += add[x];
41            self(self, y, x);
42        }
43    };
44    dfs2(dfs2, 0, -1);
45    for (int i = 0; i < n; i++) {
46        cout << ans + add[i] << " \n"[i == n - 1];
47    }
48 }
49 int main() {
50     ios::sync_with_stdio(false);

```

```

51     cin.tie(nullptr);
52     int t;
53     cin >> t;
54     while (t--) {
55         solve();
56     }
57     return 0;
58 }
```

**367: Prefix Purchase**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have an array  $a$  of size  $n$ , initially filled with zeros ( $a_1 = a_2 = \dots = a_n = 0$ ). You also have an array of integers  $c$  of size  $n$ .

Initially, you have  $k$  coins. By paying  $c_i$  coins, you can add 1 to all elements of the array  $a$  from the first to the  $i$ -th element ( $a_j += 1$  for all  $1 \leq j \leq i$ ). You can buy any  $c_i$  any number of times. A purchase is only possible if  $k \geq c_i$ , meaning that at any moment  $k \geq 0$  must hold true.

Find the lexicographically largest array  $a$  that can be obtained.

An array  $a$  is lexicographically smaller than an array  $b$  of the same length if and only if in the first position where  $a$  and  $b$  differ, the element in array  $a$  is smaller than the corresponding element in  $b$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> c(n);
8     for (int i = 0; i < n; i++) {
9         cin >> c[i];
10    }
11    for (int i = n - 2; i >= 0; i--) {
12        c[i] = min(c[i], c[i + 1]);
13    }
14    int k;
15    cin >> k;
16    vector<int> a(n);
17    int t = k;
18    for (int i = 0; i < n; i++) {
19        int v = c[i] - (i ? c[i - 1] : 0);
```

```

20         if (v > 0) {
21             t = min(t, k / v);
22         }
23         k -= t * v;
24         a[i] = t;
25         cout << a[i] << " \n"[i == n - 1];
26     }
27 }
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(nullptr);
31     int t;
32     cin >> t;
33     while (t--) {
34         solve();
35     }
36     return 0;
37 }
```

### 368: Korney Korneevich and XOR (easy version)

- Time limit: 1.5 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

This is an easier version of the problem with smaller constraints.

Korney Korneevich dag up an array  $a$  of length  $n$ . Korney Korneevich has recently read about the operation bitwise XOR, so he wished to experiment with it. For this purpose, he decided to find all integers  $x \geq 0$  such that there exists an increasing subsequence of the array  $a$ , in which the bitwise XOR of numbers is equal to  $x$ .

It didn't take a long time for Korney Korneevich to find all such  $x$ , and he wants to check his result. That's why he asked you to solve this problem!

A sequence  $s$  is a subsequence of a sequence  $b$  if  $s$  can be obtained from  $b$  by deletion of several (possibly, zero or all) elements.

A sequence  $s_1, s_2, \dots, s_m$  is called increasing if  $s_1 < s_2 < \dots < s_m$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
```

```

6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    constexpr int V = 8192;
14    vector<bitset<V>{}> dp(V + 1, bitset<V>{});
15    vector<vector<int>> f(V + 1);
16    for (int i = 0; i <= V; i++) {
17        dp[i][0] = 1;
18        f[i].push_back(0);
19    }
20    for (auto x : a) {
21        auto b = move(f[x]);
22        for (auto y : b) {
23            if (!dp[x][x ^ y]) {
24                for (int j = x; j <= V && !dp[j][x ^ y]; j++) {
25                    dp[j][x ^ y] = 1;
26                    f[j].push_back(x ^ y);
27                }
28            }
29        }
30    }
31    vector<int> ans;
32    for (int i = 0; i < V; i++) {
33        if (dp[V][i]) {
34            ans.push_back(i);
35        }
36    }
37    cout << ans.size() << "\n";
38    for (auto x : ans) {
39        cout << x << " \n"[x == ans.back()];
40    }
41    return 0;
42 }

```

### 369: Frog Traveler

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Frog Gorf is traveling through Swamp kingdom. Unfortunately, after a poor jump, he fell into a well of  $n$  meters depth. Now Gorf is on the bottom of the well and has a long way up.

The surface of the well's walls vary in quality: somewhere they are slippery, but somewhere have convenient ledges. In other words, if Gorf is on  $x$  meters below ground level, then in one jump he can go up on any integer distance from 0 to  $a_x$  meters inclusive. (Note that Gorf can't jump down, only up).

Unfortunately, Gorf has to take a break after each jump (including jump on 0 meters). And after jumping

up to position  $x$  meters below ground level, he'll slip exactly  $b_x$  meters down while resting.

Calculate the minimum number of jumps Gorf needs to reach ground level.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n + 1), b(n + 1);
10    for (int i = 1; i <= n; i++) {
11        cin >> a[i];
12    }
13    for (int i = 1; i <= n; i++) {
14        cin >> b[i];
15    }
16    vector<int> dis(n + 1, -1);
17    queue<int> q;
18    dis[n] = 0;
19    q.push(n);
20    int L = n;
21    vector<int> prev(n + 1);
22    while (!q.empty()) {
23        int x = q.front();
24        q.pop();
25        int y = x + b[x];
26        while (L > y - a[y]) {
27            L--;
28            prev[L] = x;
29            dis[L] = dis[x] + 1;
30            q.push(L);
31        }
32    }
33    cout << dis[0] << "\n";
34    if (dis[0] != -1) {
35        vector<int> a;
36        for (int i = 0; i != n; i = prev[i]) {
37            a.push_back(i);
38        }
39        reverse(a.begin(), a.end());
40        for (auto x : a) {
41            cout << x << " \n"[x == 0];
42        }
43    }
44    return 0;
45 }
```

### 370: Cyclic Operations

- Time limit: 1 second
- Memory limit: 256 megabytes

- Input file: standard input
- Output file: standard output

Egor has an array  $a$  of length  $n$ , initially consisting of zeros. However, he wanted to turn it into another array  $b$  of length  $n$ .

Since Egor doesn't take easy paths, only the following operation can be used (possibly zero or several times):

choose an array  $l$  of length  $k$  ( $1 \leq l_i \leq n$ , all  $l_i$  are distinct) and change each element  $a_{l_i}$  to  $l_{(i\%k)+1}$  ( $1 \leq i \leq k$ ).

He became interested in whether it is possible to get the array  $b$  using only these operations. Since Egor is still a beginner programmer, he asked you to help him solve this problem.

The operation  $\%$  means taking the remainder, that is,  $a\%b$  is equal to the remainder of dividing the number  $a$  by the number  $b$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> b(n);
8     for (int i = 0; i < n; i++) {
9         cin >> b[i];
10        b[i]--;
11    }
12    if (k == 1) {
13        for (int i = 0; i < n; i++) {
14            if (b[i] != i) {
15                cout << "NO\n";
16                return;
17            }
18        }
19        cout << "YES\n";
20        return;
21    }
22    vector<int> vis(n, -1);
23    for (int i = 0; i < n; i++) {
24        int j = i;
25        while (vis[j] == -1) {
26            vis[j] = i;
27            j = b[j];
28        }
29        if (vis[j] == i) {
30            int len = 0;
31            int x = j;
32            do {
33                len++;
34                x = b[x];
35            } while (x != j);

```

```

36         if (len != k) {
37             cout << "NO\n";
38             return;
39         }
40     }
41 }
42 cout << "YES\n";
43 }
44 int main() {
45     ios::sync_with_stdio(false);
46     cin.tie(nullptr);
47     int t;
48     cin >> t;
49     while (t--) {
50         solve();
51     }
52     return 0;
53 }
```

### 371: Candy Party (Easy Version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the easy version of the problem. The only difference is that in this version everyone must give candies to exactly one person and receive candies from exactly one person. Note that a submission cannot pass both versions of the problem at the same time. You can make hacks only if both versions of the problem are solved.

After Zhongkao examination, Daniel and his friends are going to have a party. Everyone will come with some candies.

There will be  $n$  people at the party. Initially, the  $i$ -th person has  $a_i$  candies. During the party, they will swap their candies. To do this, they will line up in an arbitrary order and everyone will do the following exactly once:

Choose an integer  $p$  ( $1 \leq p \leq n$ ) and a non-negative integer  $x$ , then give his  $2^x$  candies to the  $p$ -th person. Note that one cannot give more candies than currently he has (he might receive candies from someone else before) and he cannot give candies to himself.

Daniel likes fairness, so he will be happy if and only if everyone receives candies from exactly one person. Meanwhile, his friend Tom likes average, so he will be happy if and only if all the people have the same number of candies after all swaps.

Determine whether there exists a way to swap candies, so that both Daniel and Tom will be happy after the swaps.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    i64 sum = accumulate(a.begin(), a.end(), 0LL);
12    if (sum % n != 0) {
13        cout << "No\n";
14        return;
15    }
16    int ave = sum / n;
17    int cnt[31] {};
18    for (int i = 0; i < n; i++) {
19        if (a[i] < ave) {
20            int d = ave - a[i];
21            int l = __builtin_ctz(d);
22            cnt[l]++;
23            d += 1 << l;
24            if (d & (d - 1)) {
25                cout << "No\n";
26                return;
27            }
28            cnt[__builtin_ctz(d)]--;
29        } else if (a[i] > ave) {
30            int d = a[i] - ave;
31            int l = __builtin_ctz(d);
32            cnt[l]--;
33            d += 1 << l;
34            if (d & (d - 1)) {
35                cout << "No\n";
36                return;
37            }
38            cnt[__builtin_ctz(d)]++;
39        }
40    }
41    for (int i = 0; i <= 30; i++) {
42        if (cnt[i]) {
43            cout << "No\n";
44            return;
45        }
46    }
47    cout << "Yes\n";
48 }
49 int main() {
50     ios::sync_with_stdio(false);
51     cin.tie(nullptr);
52     int t;
53     cin >> t;
54     while (t--) {
55         solve();
56     }
57     return 0;
58 }
```

## 372: Manipulating History

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Keine has the ability to manipulate history.

The history of Gensokyo is a string  $s$  of length 1 initially. To fix the chaos caused by Yukari, she needs to do the following operations  $n$  times, for the  $i$ -th time:

She chooses a non-empty substring  $t_{2i-1}$  of  $s$ .

She replaces  $t_{2i-1}$  with a non-empty string,  $t_{2i}$ . Note that the lengths of strings  $t_{2i-1}$  and  $t_{2i}$  can be different.

Note that if  $t_{2i-1}$  occurs more than once in  $s$ , exactly one of them will be replaced.

For example, let  $s$  = “marisa”,  $t_{2i-1}$  = “a”, and  $t_{2i}$  = “z”. After the operation,  $s$  becomes “mzrisa” or “marisz”.

After  $n$  operations, Keine got the final string and an operation sequence  $t$  of length  $2n$ . Just as Keine thinks she has finished, Yukari appears again and shuffles the order of  $t$ . Worse still, Keine forgets the initial history.

Help Keine find the initial history of Gensokyo!

Recall that a substring is a sequence of consecutive characters of the string. For example, for string “abc” its substrings are: “ab”, “c”, “bc” and some others. But the following strings are not its substring: “ac”, “cba”, “acb”.

Hacks

You cannot make hacks in this problem.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     char ans = 0;
8     for (int i = 0; i < 2 * n + 1; i++) {
9         string t;
10        cin >> t;
11        for (auto c : t) {

```

```

12         ans ^= c;
13     }
14   }
15   cout << ans << "\n";
16 }
17 int main() {
18   ios::sync_with_stdio(false);
19   cin.tie(nullptr);
20   int t;
21   cin >> t;
22   while (t--) {
23     solve();
24   }
25   return 0;
26 }
```

### 373: Sorting By Multiplication

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  of length  $n$ , consisting of positive integers.

You can perform the following operation on this array any number of times (possibly zero):

choose three integers  $l, r$  and  $x$  such that  $1 \leq l \leq r \leq n$ , and multiply every  $a_i$  such that  $l \leq i \leq r$  by  $x$ .

Note that you can choose any integer as  $x$ , it doesn't have to be positive.

You have to calculate the minimum number of operations to make the array  $a$  sorted in strictly ascending order (i. e. the condition  $a_1 < a_2 < \dots < a_n$  must be satisfied).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5   int n;
6   cin >> n;
7   vector<int> a(n);
8   for (int i = 0; i < n; i++) {
9     cin >> a[i];
10  }
11  int ans = 0;
12  for (int i = 0; i < n - 1; i++) {
13    ans += (a[i] >= a[i + 1]);
14  }
15  int res = ans;
```

```

16     for (int i = 1; i < n; i++) {
17         res -= (a[i - 1] >= a[i]);
18         ans = min(ans, res + 1);
19         res += (a[i - 1] <= a[i]);
20     }
21     cout << ans << "\n";
22 }
23 int main() {
24     ios::sync_with_stdio(false);
25     cin.tie(nullptr);
26     int t;
27     cin >> t;
28     while (t--) {
29         solve();
30     }
31     return 0;
32 }
```

### 374: Matrix Cascade

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

There is a matrix of size  $n \times n$  which consists of 0s and 1s. The rows are numbered from 1 to  $n$  from top to bottom, the columns are numbered from 1 to  $n$  from left to right. The cell at the intersection of the  $x$ -th row and the  $y$ -th column is denoted as  $(x, y)$ .

AquaMoon wants to turn all elements of the matrix to 0s. In one step she can perform the following operation:

Select an arbitrary cell, let it be  $(i, j)$ , then invert the element in  $(i, j)$  and also invert all elements in cells  $(x, y)$  for  $x > i$  and  $x - i \geq |y - j|$ . To invert a value means to change it to the opposite: 0 changes to 1, 1 changes to 0.

Help AquaMoon determine the minimum number of steps she need to perform to turn all elements of the matrix to 0s. We can show that an answer always exists.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<vector<int>> a(n, vector<int>(n)), b(n, vector<int>(n));
8     for (int i = 0; i < n; i++) {
9         string s;
```

```

10         cin >> s;
11         for (int j = 0; j < n; j++) {
12             a[i][j] = s[j] - '0';
13         }
14     }
15     int ans = 0;
16     for (int i = 0; i < n; i++) {
17         for (int j = 0; j < n; j++) {
18             a[i][j] ^= b[i][j];
19             ans += a[i][j];
20             b[i][j] ^= a[i][j];
21             a[i][j] ^= b[i][j];
22             if (i + 1 < n) {
23                 a[i + 1][j] ^= b[i][j];
24                 if (j - 1 >= 0) {
25                     b[i + 1][j - 1] ^= b[i][j];
26                 } else if (i + 2 < n) {
27                     b[i + 2][j] ^= b[i][j];
28                 }
29                 if (j + 1 < n) {
30                     b[i + 1][j + 1] ^= b[i][j];
31                 } else if (i + 2 < n) {
32                     b[i + 2][j] ^= b[i][j];
33                 }
34                 if (i + 2 < n) {
35                     b[i + 2][j] ^= b[i][j];
36                 }
37             }
38             b[i][j] = 0;
39         }
40     }
41     cout << ans << "\n";
42 }
43 int main() {
44     ios::sync_with_stdio(false);
45     cin.tie(nullptr);
46     int t;
47     cin >> t;
48     while (t--) {
49         solve();
50     }
51     return 0;
52 }
```

## implementation

### 375: Accumulator Apex

- Time limit: 3 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

Allyn is playing a new strategy game called “Accumulator Apex”. In this game, Allyn is given the initial value of an integer  $x$ , referred to as the accumulator, and  $k$  lists of integers. Allyn can make multiple turns. On each turn, Allyn can withdraw the leftmost element from any non-empty list and add it to the

accumulator  $x$  if the resulting  $x$  is non-negative. Allyn can end the game at any moment. The goal of the game is to get the largest possible value of the accumulator  $x$ . Please help Allyn find the largest possible value of the accumulator  $x$  they can get in this game.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     i64 x;
8     int k;
9     cin >> x >> k;
10    vector<array<i64, 2>> all;
11    for (int i = 0; i < k; i++) {
12        int l;
13        cin >> l;
14        vector<int> a(l);
15        for (int j = 0; j < l; j++) {
16            cin >> a[j];
17        }
18        vector<array<i64, 2>> stk;
19        for (int j = l - 1; j >= 0; j--) {
20            i64 x = 0, y = 0;
21            x = min(a[j], 0);
22            y = a[j];
23            while (!stk.empty() && (y <= 0 || y + stk.back()[0] >= 0)) {
24                auto v = stk.back();
25                stk.pop_back();
26                x = min(x, y + v[0]);
27                y += v[1];
28            }
29            if (y > 0) {
30                stk.push_back({x, y});
31            }
32        }
33        all.insert(all.end(), stk.begin(), stk.end());
34    }
35    sort(all.begin(), all.end(), greater());
36    for (auto [a, b] : all) {
37        if (x + a >= 0) {
38            x += b;
39        } else {
40            break;
41        }
42    }
43    cout << x << "\n";
44    return 0;
45 }
```

### 376: The Third Letter

- Time limit: 2 seconds

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In order to win his toughest battle, Mircea came up with a great strategy for his army. He has  $n$  soldiers and decided to arrange them in a certain way in camps. Each soldier has to belong to exactly one camp, and there is one camp at each integer point on the  $x$ -axis (at points  $\dots, -2, -1, 0, 1, 2, \dots$ ).

The strategy consists of  $m$  conditions. Condition  $i$  tells that soldier  $a_i$  should belong to a camp that is situated  $d_i$  meters in front of the camp that person  $b_i$  belongs to. (If  $d_i < 0$ , then  $a_i$ 's camp should be  $-d_i$  meters behind  $b_i$ 's camp.)

Now, Mircea wonders if there exists a partition of soldiers that respects the condition and he asks for your help! Answer “YES” if there is a partition of the  $n$  soldiers that satisfies all of the  $m$  conditions and “NO” otherwise.

Note that two different soldiers may be placed in the same camp.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<i64> f(n);
8     vector<bool> vis(n);
9     vector<vector<pair<int, int>>> adj(n);
10    for (int i = 0; i < m; i++) {
11        int a, b, d;
12        cin >> a >> b >> d;
13        a--;
14        b--;
15        adj[a].emplace_back(b, -d);
16        adj[b].emplace_back(a, d);
17    }
18    for (int i = 0; i < n; i++) {
19        if (!vis[i]) {
20            vis[i] = true;
21            queue<int> q;
22            q.push(i);
23            while (!q.empty()) {
24                int x = q.front();
25                q.pop();
26                for (auto [y, d] : adj[x]) {
27                    if (!vis[y]) {
28                        vis[y] = true;
29                        q.push(y);
30                        f[y] = f[x] + d;
31                    } else if (f[y] != f[x] + d) {
32                        cout << "NO\n";
33                        return;
34                    }
35                }
36            }
37        }
38    }
39}
```

```

35         }
36     }
37   }
38   cout << "YES\n";
39 }
40 int main() {
41   ios::sync_with_stdio(false);
42   cin.tie(nullptr);
43   int t;
44   cin >> t;
45   while (t--) {
46     solve();
47   }
48   return 0;
49 }
```

### 377: Professor Higashikata

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Josuke is tired of his peaceful life in Morioh. Following in his nephew Jotaro's footsteps, he decides to study hard and become a professor of computer science. While looking up competitive programming problems online, he comes across the following one:

Let  $s$  be a binary string of length  $n$ . An operation on  $s$  is defined as choosing two distinct integers  $i$  and  $j$  ( $1 \leq i < j \leq n$ ), and swapping the characters  $s_i, s_j$ .

Consider the  $m$  strings  $t_1, t_2, \dots, t_m$ , where  $t_i$  is the substring <sup>†</sup> of  $s$  from  $l_i$  to  $r_i$ . Define  $t(s) = t_1 + t_2 + \dots + t_m$  as the concatenation of the strings  $t_i$  in that order.

There are  $q$  updates to the string. In the  $i$ -th update  $s_{x_i}$  gets flipped. That is if  $s_{x_i} = 1$ , then  $s_{x_i}$  becomes 0 and vice versa. After each update, find the minimum number of operations one must perform on  $s$  to make  $t(s)$  lexicographically as large<sup>‡</sup> as possible.

Note that no operation is actually performed. We are only interested in the number of operations.

Help Josuke in his dream by solving the problem for him.

<sup>†</sup> A string  $a$  is a substring of a string  $b$  if  $a$  can be obtained from  $b$  by the deletion of several (possibly, zero or all) characters from the beginning and several (possibly, zero or all) characters from the end.

<sup>‡</sup> A string  $a$  is lexicographically larger than a string  $b$  of the same length if and only if the following holds:

in the first position where  $a$  and  $b$  differ, the string  $a$  has a 1, and the string  $b$  has a 0.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, m, q;
9     cin >> n >> m >> q;
10    string s;
11    cin >> s;
12    DSU dsu(n + 1);
13    vector<int> p;
14    for (int i = 0; i < m; i++) {
15        int l, r;
16        cin >> l >> r;
17        l--;
18        for (int j = dsu.find(l); j < r; j = dsu.find(j)) {
19            p.push_back(j);
20            dsu.merge(j + 1, j);
21        }
22    }
23    vector<int> invp(n, -1);
24    for (int i = 0; i < p.size(); i++) {
25        invp[p[i]] = i;
26    }
27    int tot = p.size();
28    int cnt = count(s.begin(), s.end(), '1');
29    int ans = 0;
30    for (int i = 0; i < min(cnt, tot); i++) {
31        ans += (s[p[i]] == '0');
32    }
33    while (q--) {
34        int x;
35        cin >> x;
36        x--;
37        if (s[x] == '1') {
38            cnt--;
39            ans -= (cnt < tot && s[p[cnt]] == '0');
40            s[x] = '0';
41            ans += (invp[x] != -1 && invp[x] < cnt);
42        } else {
43            ans -= (invp[x] != -1 && invp[x] < cnt);
44            s[x] = '1';
45            ans += (cnt < tot && s[p[cnt]] == '0');
46            cnt++;
47        }
48        cout << ans << "\n";
49    }
50    return 0;
51 }
```

### 378: k-th equality

- Time limit: 1 second
- Memory limit: 256 megabytes

- Input file: standard input
- Output file: standard output

Consider all equalities of form  $a + b = c$ , where  $a$  has  $A$  digits,  $b$  has  $B$  digits, and  $c$  has  $C$  digits. All the numbers are positive integers and are written without leading zeroes. Find the  $k$ -th lexicographically smallest equality when written as a string like above or determine that it does not exist.

For example, the first three equalities satisfying  $A = 1, B = 1, C = 2$  are

$1 + 9 = 10,$

$2 + 8 = 10,$

$2 + 9 = 11.$

An equality  $s$  is lexicographically smaller than an equality  $t$  with the same lengths of the numbers if and only if the following holds:

in the first position where  $s$  and  $t$  differ, the equality  $s$  has a smaller digit than the corresponding digit in  $t$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int pw[7] = {1, 10, 100, 1000, 10000, 100000, 1000000};
5 void solve() {
6     int A, B, C;
7     cin >> A >> B >> C;
8     i64 k;
9     cin >> k;
10    for (int x = pw[A - 1]; x < pw[A]; x++) {
11        int lo = max(pw[B - 1], pw[C - 1] - x);
12        int hi = min(pw[B] - 1, pw[C] - 1 - x);
13        int cnt = max(0, hi - lo + 1);
14        if (k <= cnt) {
15            cout << x << " + " << lo + k - 1 << " = " << x + lo + k - 1 << "\n";
16            return;
17        }
18        k -= cnt;
19    }
20    cout << -1 << "\n";
21 }
22 int main() {
23     ios::sync_with_stdio(false);
24     cin.tie(nullptr);
25     int t;
26     cin >> t;
27     while (t--) {
28         solve();
29     }
30     return 0;
31 }
```

### 379: Survey in Class

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Zinaida Viktorovna has  $n$  students in her history class. The homework for today included  $m$  topics, but the students had little time to prepare, so  $i$ -th student learned only topics from  $l_i$  to  $r_i$  inclusive.

At the beginning of the lesson, each student holds their hand at 0. The teacher wants to ask some topics. It goes like this:

The teacher asks the topic  $k$ .

If the student has learned topic  $k$ , then he raises his hand by 1, otherwise he lowers it by 1.

Each topic Zinaida Viktorovna can ask no more than one time.

Find the maximum difference of the heights of the highest and the lowest hand that can be in class after the survey.

Note that the student's hand can go below 0.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<int> l(n), r(n), len(n);
8     for (int i = 0; i < n; i++) {
9         cin >> l[i] >> r[i];
10        l[i]--;
11        len[i] = r[i] - l[i];
12    }
13    int ans = 0;
14    int max = *max_element(len.begin(), len.end());
15    int min = *min_element(len.begin(), len.end());
16    ans = max(ans, max - min);
17    int maxl = *max_element(l.begin(), l.end());
18    int minr = *min_element(r.begin(), r.end());
19    for (int i = 0; i < n; i++) {
20        ans = max(ans, min(len[i], max(maxl - l[i], r[i] - minr)));
21    }
22    cout << 2 * ans << "\n";
23 }
24 int main() {
25     ios::sync_with_stdio(false);

```

```

26     cin.tie(nullptr);
27     int t;
28     cin >> t;
29     while (t--) {
30         solve();
31     }
32     return 0;
33 }
```

### 380: Maximum Xor Secondary

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Bike loves looking for the second maximum element in the sequence. The second maximum element in the sequence of distinct numbers  $x_1, x_2, \dots, x_k$  ( $k > 1$ ) is such maximum element  $x_j$ , that the following inequality holds: .

The lucky number of the sequence of distinct positive integers  $x_1, x_2, \dots, x_k$  ( $k > 1$ ) is the number that is equal to the bitwise excluding OR of the maximum element of the sequence and the second maximum element of the sequence.

You've got a sequence of distinct positive integers  $s_1, s_2, \dots, s_n$  ( $n > 1$ ). Let's denote sequence  $s_l, s_{l+1}, \dots, s_r$  as  $s[l..r]$  ( $1 \leq l < r \leq n$ ). Your task is to find the maximum number among all lucky numbers of sequences  $s[l..r]$ .

Note that as all numbers in sequence  $s$  are distinct, all the given definitions make sense.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    int ans = 0;
14    vector<int> s;
15    for (int i = 0; i < n; i++) {
16        while (!s.empty() && a[i] > a[s.back()]) {
```

```

17         ans = max(a[i] ^ a[s.back()], ans);
18         s.pop_back();
19     }
20     if (!s.empty()) {
21         ans = max(a[i] ^ a[s.back()], ans);
22     }
23     s.push_back(i);
24 }
25 cout << ans << "\n";
26 return 0;
27 }
```

### 381: Nearest Fraction

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given three positive integers  $x, y, n$ . Your task is to find the nearest fraction to  $\frac{x}{y}$  whose denominator is no more than  $n$ .

Formally, you should find such pair of integers  $a, b$  ( $1 \leq b \leq n; 0 \leq a$ ) that the value is as minimal as possible.

If there are multiple “nearest” fractions, choose the one with the minimum denominator. If there are multiple “nearest” fractions with the minimum denominator, choose the one with the minimum numerator.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     i64 x, y, n;
8     cin >> x >> y >> n;
9     i64 a = 0, b = 1;
10    for (int j = 1; j <= n; j++) {
11        i64 t = x * j / y;
12        for (auto i : {t, t + 1}) {
13            if (abs(i * b * y - x * b * j) < abs(a * j * y - x * b * j)) {
14                a = i;
15                b = j;
16            }
17        }
18    }
19    cout << a << "/" << b << "\n";
20    return 0;
21 }
```

### 382: Petya and Divisors

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Little Petya loves looking for numbers' divisors. One day Petya came across the following problem:

You are given  $n$  queries in the form " $x_i \ y_i$ ". For each query Petya should count how many divisors of number  $x_i$  divide none of the numbers  $x_i - y_i, x_i - y_i + 1, \dots, x_i - 1$ . Help him.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> x(n), y(n);
10    for (int i = 0; i < n; i++) {
11        cin >> x[i] >> y[i];
12    }
13    int m = *max_element(x.begin(), x.end());
14    vector<vector<int>> divs(m + 1);
15    for (int i = 1; i <= m; i++) {
16        for (int j = i; j <= m; j += i) {
17            divs[j].push_back(i);
18        }
19    }
20    vector f(m + 1, -1);
21    for (int i = 0; i < n; i++) {
22        int ans = 0;
23        for (auto d : divs[x[i]]) {
24            ans += (f[d] < i - y[i]);
25            f[d] = i;
26        }
27        cout << ans << "\n";
28    }
29    return 0;
30 }
```

### 383: Parking Lot

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Nowadays it is becoming increasingly difficult to park a car in cities successfully. Let's imagine a segment of a street as long as L meters along which a parking lot is located. Drivers should park their cars strictly parallel to the pavement on the right side of the street (remember that in the country the authors of the tasks come from the driving is right side!). Every driver when parking wants to leave for themselves some extra space to move their car freely, that's why a driver is looking for a place where the distance between his car and the one behind his will be no less than b meters and the distance between his car and the one in front of his will be no less than f meters (if there's no car behind then the car can be parked at the parking lot segment edge; the same is true for the case when there're no cars parked in front of the car). Let's introduce an axis of coordinates along the pavement. Let the parking lot begin at point 0 and end at point L. The drivers drive in the direction of the coordinates' increasing and look for the earliest place (with the smallest possible coordinate) where they can park the car. In case there's no such place, the driver drives on searching for his perfect peaceful haven. Sometimes some cars leave the street and free some space for parking. Considering that there never are two moving cars on a street at a time write a program that can use the data on the drivers, entering the street hoping to park there and the drivers leaving it, to model the process and determine a parking lot space for each car.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int L, b, f;
8     cin >> L >> b >> f;
9     set<pair<int, int>> s;
10    s.emplace(-b, -b);
11    s.emplace(L + f, L + f);
12    int n;
13    cin >> n;
14    vector<pair<int, int>> a(n);
15    for (int i = 0; i < n; i++) {
16        int t, x;
17        cin >> t >> x;
18        if (t == 1) {
19            int p = -1;
20            for (auto it = next(s.begin()); it != s.end(); it++) {
21                if (it->first - prev(it)->second >= b + x + f) {
22                    p = prev(it)->second + b;
23                    break;
24                }
25            }
26            if (p != -1) {
27                s.emplace(p, p + x);
28                a[i] = {p, p + x};
29            }
30            cout << p << "\n";
31        }
32    }
33}
```

```

31         } else {
32             x--;
33             s.erase(a[x]);
34         }
35     }
36     return 0;
37 }
```

### 384: Game of chess unfinished

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Once Volodya was at the museum and saw a regular chessboard as a museum piece. And there were only four chess pieces on it: two white rooks, a white king and a black king. “Aha, blacks certainly didn’t win!”, - Volodya said and was right for sure. And your task is to say whether whites had won or not.

Pieces on the chessboard are guaranteed to represent a correct position (every piece occupies one cell, no two pieces occupy the same cell and kings cannot take each other). Thus, your task is only to decide whether whites mate blacks. We would remind you that it means that the black king can be taken by one of the opponent’s pieces at the moment and also it cannot move to an unbeaten position. A rook moves vertically or horizontally by any number of free cells (assuming there are no other pieces on its path), a king - to the adjacent cells (either by corner or by side). Certainly, pieces cannot leave the board. The black king might be able to take opponent’s rooks at his turn (see sample 3).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int x[4], y[4];
8     for (int i = 0; i < 4; i++) {
9         string s;
10        cin >> s;
11        x[i] = s[0] - 'a';
12        y[i] = s[1] - '1';
13    }
14    for (int dx = -1; dx <= 1; dx++) {
15        for (int dy = -1; dy <= 1; dy++) {
16            int nx = x[3] + dx;
```

```

17         int ny = y[3] + dy;
18         if (nx < 0 || nx > 7 || ny < 0 || ny > 7) {
19             continue;
20         }
21         if ((x[0] == nx) ^ (y[0] == ny)) {
22             if (x[2] < min(nx, x[0]) || x[2] > max(nx, x[0])
23                 || y[2] < min(ny, y[0]) || y[2] > max(ny, y[0])) {
24                 continue;
25             }
26         }
27         if ((x[1] == nx) ^ (y[1] == ny)) {
28             if (x[2] < min(nx, x[1]) || x[2] > max(nx, x[1])
29                 || y[2] < min(ny, y[1]) || y[2] > max(ny, y[1])) {
30                 continue;
31             }
32         }
33         if (abs(x[2] - nx) <= 1 && abs(y[2] - ny) <= 1) {
34             continue;
35         }
36         cout << "OTHER\n";
37         return 0;
38     }
39 }
40 cout << "CHECKMATE\n";
41 return 0;
42 }
```

### 385: Computer Game

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vasyas elder brother Petya loves playing computer games. In one of his favourite computer games Petya reached the final level where a fight with the boss take place.

While playing the game Petya found spell scrolls and now he is about to use them. Lets describe the way fighting goes on this level:

- 1) The boss has two parameters: max - the initial amount of health and reg - regeneration rate per second.
- 2) Every scroll also has two parameters: powi - spell power measured in percents - the maximal amount of health counted off the initial one, which allows to use the scroll (i.e. if the boss has more than powi percent of health the scroll cannot be used); and dmgi the damage per second inflicted upon the boss if the scroll is used. As soon as a scroll is used it disappears and another spell is cast upon the boss that inflicts dmgi of damage per second upon him until the end of the game.

During the battle the actions per second are performed in the following order: first the boss gets the damage from all the spells cast upon him, then he regenerates reg of health (at the same time he can't have more than max of health), then the player may use another scroll (no more than one per second).

The boss is considered to be defeated if at the end of a second he has nonpositive ( $\leq 0$ ) amount of health.

Help Petya to determine whether he can win with the set of scrolls available to him and if he can, determine the minimal number of seconds he needs to do it.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 N, hp, reg = map(int, input().split())
2 pw = [0] * N;
3 dmg = [0] * N;
4 for i in range(N) :
5     pw[i], dmg[i] = map(int, input().split())
6 use = [-1] * N
7 tot = 0
8 cur = hp
9 time = 0
10 ans = []
11 while True :
12     cur = min(cur - tot + reg, hp)
13     if cur <= 0 :
14         break
15     i = -1
16     for j in range(N) :
17         if cur * 100 <= pw[j] * hp and (i == -1 or dmg[j] > dmg[i]) and use[j] == -1 :
18             i = j
19     if i == -1 and reg >= tot :
20         print('NO')
21         exit(0)
22     if i != -1 :
23         use[i] = time
24         tot += dmg[i]
25         ans.append((time, i+1))
26     time += 1
27 print('YES')
28 print(time, len(ans))
29 for (a, b) in ans :
30     print(a, b)

```

### 386: Warehouse

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: input.txt

- Output file: output.txt

Once upon a time, when the world was more beautiful, the sun shone brighter, the grass was greener and the sausages tasted better Arlandia was the most powerful country. And its capital was the place where our hero DravDe worked. He couldnt program or make up problems (in fact, few people saw a computer those days) but he was nevertheless happy. He worked in a warehouse where a magical but non-alcoholic drink Ogudar-Olok was kept. We wont describe his work in detail and take a better look at a simplified version of the warehouse.

The warehouse has one set of shelving. It has n shelves, each of which is divided into m sections. The shelves are numbered from top to bottom starting from 1 and the sections of each shelf are numbered from left to right also starting from 1. Each section can contain exactly one box of the drink, and try as he might, DravDe can never put a box in a section that already has one. In the course of his work DravDe frequently notices that he has to put a box in a filled section. In that case his solution is simple. DravDe ignores that section and looks at the next one to the right. If it is empty, he puts the box there. Otherwise he keeps looking for the first empty section to the right. If no empty section is found by the end of the shelf, he looks at the shelf which is under it, then the next one, etc. Also each time he looks at a new shelf he starts from the shelfs beginning. If DravDe still cant find an empty section for the box, he immediately drinks it all up and throws the empty bottles away not to be caught.

After one great party with a lot of Ogudar-Olok drunk DravDe asked you to help him. Unlike him, you can program and therefore modeling the process of counting the boxes in the warehouse will be easy work for you.

The process of counting contains two types of query messages:

+1 x y id (where x, y are integers,  $1 \leq x \leq n$ ,  $1 \leq y \leq m$ , and id is a string of lower case Latin letters - from 1 to 10 characters long). That query means that the warehouse got a box identified as id, which should be put in the section y on the shelf x. If the section is full, use the rules described above. It is guaranteed that every moment of the process the identifiers of all the boxes in the warehouse are different. You dont have to answer this query.

-1 id (where id is a string of lower case Latin letters - from 1 to 10 characters long). That query means that a box identified as id is removed from the warehouse. You have to answer this query (see output format).

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 #ifdef ONLINE_JUDGE
5     ifstream fin("input.txt");
6     ofstream fout("output.txt");
```

```

7 #else
8     #define fin cin
9     #define fout cout
10 #endif
11 int main() {
12     int n, m, k;
13     fin >> n >> m >> k;
14     map<string, int> pos;
15     vector<bool> occu(n * m);
16     while (k--) {
17         string o;
18         fin >> o;
19         if (o == "+1") {
20             int x, y;
21             string id;
22             fin >> x >> y >> id;
23             x--, y--;
24             int p = x * m + y;
25             while (p < n * m && occu[p]) {
26                 p++;
27             }
28             if (p < n * m) {
29                 occu[p] = true;
30                 pos[id] = p;
31             }
32         } else {
33             string id;
34             fin >> id;
35             if (pos.count(id)) {
36                 auto p = pos[id];
37                 pos.erase(id);
38                 occu[p] = false;
39                 fout << p / m + 1 << " " << p % m + 1 << "\n";
40             } else {
41                 fout << -1 << " " << -1 << "\n";
42             }
43         }
44     }
45     return 0;
46 }
```

### 387: The Butcher

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Anton plays his favorite game “Defense of The Ancients 2” for his favorite hero - The Butcher. Now he wants to make his own dinner. To do this he will take a rectangle of height  $h$  and width  $w$ , then make a vertical or horizontal cut so that both resulting parts have integer sides. After that, he will put one of the parts in the box and cut the other again, and so on.

More formally, a rectangle of size  $h \times w$  can be cut into two parts of sizes  $x \times w$  and  $(h - x) \times w$ , where  $x$  is an integer from 1 to  $(h - 1)$ , or into two parts of sizes  $h \times y$  and  $h \times (w - y)$ , where  $y$  is an integer

from 1 to  $(w - 1)$ .

He will repeat this operation  $n - 1$  times, and then put the remaining rectangle into the box too. Thus, the box will contain  $n$  rectangles, of which  $n - 1$  rectangles were put in the box as a result of the cuts, and the  $n$ -th rectangle is the one that the Butcher has left after all  $n - 1$  cuts.

Unfortunately, Butcher forgot the numbers  $h$  and  $w$ , but he still has  $n$  rectangles mixed in random order. Note that Butcher didn't rotate the rectangles, but only shuffled them. Now he wants to know all possible pairs  $(h, w)$  from which this set of rectangles can be obtained. And you have to help him do it!

It is guaranteed that there exists at least one pair  $(h, w)$  from which this set of rectangles can be obtained.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     i64 area = 0;
8     vector<int> a(n), b(n);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i] >> b[i];
11         area += 1LL * a[i] * b[i];
12     }
13     int h = *max_element(a.begin(), a.end());
14     int w = *max_element(b.begin(), b.end());
15     vector<pair<i64, i64>> ans;
16     for (auto [H, W] : {pair(1LL * h, area / h), {area / w, 1LL * w}}) {
17         if (H * W != area) {
18             continue;
19         }
20         priority_queue<pair<int, int>> qh, qw;
21         vector<bool> vis(n);
22         for (int i = 0; i < n; i++) {
23             qh.emplace(a[i], i);
24             qw.emplace(b[i], i);
25         }
26         i64 h = H, w = W;
27         while (h * w > 0) {
28             while (vis[qh.top().second]) {
29                 qh.pop();
30             }
31             while (vis[qw.top().second]) {
32                 qw.pop();
33             }
34             int i = -1;
35             if (qh.top().first == h) {
36                 i = qh.top().second;
37             }
38             if (qw.top().first == w) {
39                 i = qw.top().second;
40             }

```

```

41         if (i == -1) {
42             break;
43         }
44         vis[i] = true;
45         if (a[i] == h) {
46             w -= b[i];
47         } else {
48             h -= a[i];
49         }
50     }
51     if (h * w == 0) {
52         ans.emplace_back(h, w);
53     }
54 }
55 sort(ans.begin(), ans.end());
56 ans.erase(unique(ans.begin(), ans.end()), ans.end());
57 cout << ans.size() << "\n";
58 for (auto [h, w] : ans) {
59     cout << h << " " << w << "\n";
60 }
61 }
62 int main() {
63     ios::sync_with_stdio(false);
64     cin.tie(nullptr);
65     int t;
66     cin >> t;
67     while (t--) {
68         solve();
69     }
70     return 0;
71 }
```

### 388: Schedule

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

At the beginning of the new semester there is new schedule in the Berland State University. According to this schedule,  $n$  groups have lessons at the room 31. For each group the starting time of the lesson and the finishing time of the lesson are known. It has turned out that it is impossible to hold all lessons, because for some groups periods of their lessons intersect. If at some moment of time one group finishes its lesson, and the other group starts the lesson, their lessons don't intersect.

The dean wants to cancel the lesson in one group so that no two time periods of lessons of the remaining groups intersect. You are to find all ways to do that.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
```

```

2  using namespace std;
3  using i64 = long long;
4  int main() {
5      ios::sync_with_stdio(false);
6      cin.tie(nullptr);
7      int n;
8      cin >> n;
9      vector<int> l(n), r(n);
10     for (int i = 0; i < n; i++) {
11         cin >> l[i] >> r[i];
12     }
13     vector<int> ord(n);
14     iota(ord.begin(), ord.end(), 0);
15     sort(ord.begin(), ord.end(), [&](int i, int j) {
16         return l[i] < l[j];
17     });
18     vector<int> ans;
19     for (int i = 0; i < n; i++) {
20         int x = 0;
21         bool ok = true;
22         for (int j = 0; j < n; j++) {
23             if (i != j) {
24                 if (l[ord[j]] < x) {
25                     ok = false;
26                     break;
27                 }
28                 x = r[ord[j]];
29             }
30         }
31         if (ok) {
32             ans.push_back(ord[i]);
33         }
34     }
35     sort(ans.begin(), ans.end());
36     cout << ans.size() << "\n";
37     for (auto x : ans) {
38         cout << x + 1 << " \n"[x == ans.back()];
39     }
40     return 0;
41 }
```

### 389: Codeforces World Finals

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The king Copa often has been reported about the Codeforces site, which is rapidly getting more and more popular among the brightest minds of the humanity, who are using it for training and competing. Recently Copa understood that to conquer the world he needs to organize the world Codeforces tournament. He hopes that after it the brightest minds will become his subordinates, and the toughest part of conquering the world will be completed.

The final round of the Codeforces World Finals 20YY is scheduled for DD.MM.YY, where DD is the day of

the round, MM is the month and YY are the last two digits of the year. Bob is lucky to be the first finalist from Berland. But there is one problem: according to the rules of the competition, all participants must be at least 18 years old at the moment of the finals. Bob was born on BD.BM.BY. This date is recorded in his passport, the copy of which he has already mailed to the organizers. But Bob learned that in different countries the way, in which the dates are written, differs. For example, in the US the month is written first, then the day and finally the year. Bob wonders if it is possible to rearrange the numbers in his date of birth so that he will be at least 18 years old on the day DD.MM.YY. He can always tell that in his motherland dates are written differently. Help him.

According to another strange rule, eligible participant must be born in the same century as the date of the finals. If the day of the finals is participant's 18-th birthday, he is allowed to participate.

As we are considering only the years from 2001 to 2099 for the year of the finals, use the following rule: the year is leap if its number is divisible by four.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 const int day[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     char dot;
9     int DD, MM, YY;
10    cin >> DD >> dot >> MM >> dot >> YY;
11    int BD, BM, BY;
12    cin >> BD >> dot >> BM >> dot >> BY;
13    auto check = [&](int d, int m, int y) {
14        if (m > 12) {
15            return;
16        }
17        if (d > day[m - 1] + (m == 2 && y % 4 == 0)) {
18            return;
19        }
20        if (YY - y < 18) {
21            return;
22        }
23        if (YY - y > 18) {
24            cout << "YES\n";
25            exit(0);
26        }
27        if (MM < m) {
28            return;
29        }
30        if (MM > m) {
31            cout << "YES\n";
32            exit(0);
33        }
34        if (d <= DD) {
35            cout << "YES\n";
```

```

36         exit(0);
37     }
38 }
39 check(BD, BM, BY);
40 check(BD, BY, BM);
41 check(BM, BD, BY);
42 check(BM, BY, BD);
43 check(BY, BD, BM);
44 check(BY, BM, BD);
45 cout << "NO\n";
46 return 0;
47 }
```

### 390: Mail Stamps

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

One day Bob got a letter in an envelope. Bob knows that when Berland's post officers send a letter directly from city A to city B, they stamp it with A B, or B A. Unfortunately, often it is impossible to send a letter directly from the city of the sender to the city of the receiver, that's why the letter is sent via some intermediate cities. Post officers never send a letter in such a way that the route of this letter contains some city more than once. Bob is sure that the post officers stamp the letters accurately.

There are n stamps on the envelope of Bob's letter. He understands that the possible routes of this letter are only two. But the stamps are numerous, and Bob can't determine himself none of these routes. That's why he asks you to help him. Find one of the possible routes of the letter.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     map<int, vector<int>> adj;
10    for (int i = 0; i < n; i++) {
11        int a, b;
12        cin >> a >> b;
13        adj[a].push_back(b);
14        adj[b].push_back(a);
15    }
16    int x;
17    for (auto [y, e] : adj) {
18        if (e.size() == 1) {
19            x = y;
```

```

1  }
2  }
3  cout << x;
4  int y = adj[x][0];
5  while (true) {
6      cout << " " << y;
7      if (adj[y].size() == 1) {
8          break;
9      }
10     x = adj[y][0] ^ adj[y][1] ^ x;
11     swap(x, y);
12 }
13 cout << "\n";
14 return 0;
15 }
```

### 391: Sequence of points

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given the following points with integer coordinates on the plane: M<sub>0</sub>, A<sub>0</sub>, A<sub>1</sub>, ..., A<sub>n - 1</sub>, where n is odd number. Now we define the following infinite sequence of points M<sub>i</sub>: M<sub>i</sub> is symmetric to M<sub>i - 1</sub> according (for every natural number i). Here point B is symmetric to A according M, if M is the center of the line segment AB. Given index j find the point M<sub>j</sub>.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     i64 j;
9     cin >> n >> j;
10    i64 mx, my;
11    cin >> mx >> my;
12    vector<i64> x(n), y(n);
13    for (int i = 0; i < n; i++) {
14        cin >> x[i] >> y[i];
15    }
16    j %= 2 * n;
17    for (int i = 0; i < j; i++) {
18        mx = 2 * x[i % n] - mx;
19        my = 2 * y[i % n] - my;
20    }
21    cout << mx << " " << my << "\n";
```

```

22     return 0;
23 }
```

### 392: Jabber ID

- Time limit: 0.5 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Jabber ID on the national Berland service Babber has a form @[/resource], where

- is a sequence of Latin letters (lowercase or uppercase), digits or underscores characters \_, the length of is between 1 and 16, inclusive.
- is a sequence of word separated by periods (characters .), where each word should contain only characters allowed for , the length of each word is between 1 and 16, inclusive. The length of is between 1 and 32, inclusive.
- is a sequence of Latin letters (lowercase or uppercase), digits or underscores characters \_, the length of is between 1 and 16, inclusive.

The content of square brackets is optional - it can be present or can be absent.

There are the samples of correct Jabber IDs: [email protected], [email protected]/contest.

Your task is to write program which checks if given string is a correct Jabber ID.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 bool valid(string s) {
5     if (s.size() < 1 || s.size() > 16) {
6         return false;
7     }
8     for (auto c : s) {
9         if (!isalpha(c) && !isdigit(c) && c != '_') {
10             return false;
11         }
12     }
13     return true;
14 }
15 int main() {
16     ios::sync_with_stdio(false);
17     cin.tie(nullptr);
18     string s;
19     cin >> s;
20     int a = s.find('@');
```

```

21     int b = s.find('/');
22     if (a == -1) {
23         cout << "NO\n";
24         return 0;
25     }
26     auto username = s.substr(0, a);
27     if (!valid(username)) {
28         cout << "NO\n";
29         return 0;
30     }
31     string hostname;
32     if (b == -1) {
33         hostname = s.substr(a + 1);
34     } else {
35         hostname = s.substr(a + 1, b - a - 1);
36     }
37     if (hostname.size() < 1 || hostname.size() > 32) {
38         cout << "NO\n";
39         return 0;
40     }
41     int i = 0;
42     while (true) {
43         int j = hostname.find('.', i);
44         string word;
45         if (j == -1) {
46             word = hostname.substr(i);
47         } else {
48             word = hostname.substr(i, j - i);
49         }
50         if (!valid(word)) {
51             cout << "NO\n";
52             return 0;
53         }
54         if (j == -1) {
55             break;
56         }
57         i = j + 1;
58     }
59     if (b != -1) {
60         auto resource = s.substr(b + 1);
61         if (!valid(resource)) {
62             cout << "NO\n";
63             return 0;
64         }
65     }
66     cout << "YES\n";
67     return 0;
68 }
```

**393: Li Hua and Tree**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Li Hua has a tree of  $n$  vertices and  $n - 1$  edges. The root of the tree is vertex 1. Each vertex  $i$  has importance  $a_i$ . Denote the size of a subtree as the number of vertices in it, and the importance as the

sum of the importance of vertices in it. Denote the heavy son of a non-leaf vertex as the son with the largest subtree size. If multiple of them exist, the heavy son is the one with the minimum index.

Li Hua wants to perform  $m$  operations:

“1  $x$ ” ( $1 \leq x \leq n$ ) - calculate the importance of the subtree whose root is  $x$ .

“2  $x$ ” ( $2 \leq x \leq n$ ) - rotate the heavy son of  $x$  up. Formally, denote  $son_x$  as the heavy son of  $x$ ,  $fa_x$  as the father of  $x$ . He wants to remove the edge between  $x$  and  $fa_x$  and connect an edge between  $son_x$  and  $fa_x$ . It is guaranteed that  $x$  is not root, but not guaranteed that  $x$  is not a leaf. If  $x$  is a leaf, please ignore the operation.

Suppose you were Li Hua, please solve this problem.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    vector<int> siz(n);
14    vector<i64> sum(n);
15    vector<vector<int>> adj(n);
16    for (int i = 1; i < n; i++) {
17        int u, v;
18        cin >> u >> v;
19        u--, v--;
20        adj[u].push_back(v);
21        adj[v].push_back(u);
22    }
23    vector<int> parent(n, -1);
24    vector<set<pair<int, int>>> s(n);
25    function<void(int)> dfs = [&](int x) {
26        siz[x] = 1;
27        sum[x] = a[x];
28        for (auto y : adj[x]) {
29            if (y == parent[x]) {
30                continue;
31            }
32            parent[y] = x;
33            dfs(y);
34            sum[x] += sum[y];
35            siz[x] += siz[y];
36            s[x].emplace(-siz[y], y);
37        }
38    };
39    dfs(0);
40    while (m--) {
41        int t, x;
42        cin >> t >> x;

```

```

43     x--;
44     if (t == 1) {
45         cout << sum[x] << "\n";
46     } else {
47         if (s[x].empty()) {
48             continue;
49         }
50         int y = s[x].begin()->second;
51         s[parent[x]].erase({-siz[x], x});
52         s[x].erase({-siz[y], y});
53         siz[x] -= siz[y];
54         siz[y] += siz[x];
55         sum[x] -= sum[y];
56         sum[y] += sum[x];
57         s[y].emplace(-siz[x], x);
58         s[parent[x]].emplace(-siz[y], y);
59         parent[y] = parent[x];
60         parent[x] = y;
61     }
62 }
63 return 0;
64 }
```

### 394: BerOS file system

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

The new operating system BerOS has a nice feature. It is possible to use any number of characters ‘/’ as a delimiter in path instead of one traditional ‘/’. For example, strings //usr///local//nginx/sbin// and /usr/local/nginx//sbin are equivalent. The character ‘/’ (or some sequence of such characters) at the end of the path is required only in case of the path to the root directory, which can be represented as single character ‘/’.

A path called normalized if it contains the smallest possible number of characters ‘/’.

Your task is to transform a given path to the normalized form.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     string s;
8     cin >> s;
9     string t;
10    for (auto c : s) {
```

```

11     if (!t.empty() && t.back() == '/') && c == '/') {
12         continue;
13     }
14     t += c;
15 }
16 while (t.size() > 1 && t.back() == '/') {
17     t.pop_back();
18 }
19 cout << t << "\n";
20 return 0;
21 }

```

**math****395: Mathematical Problem**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The mathematicians of the 31st lyceum were given the following task:

You are given an odd number  $n$ , and you need to find  $n$  different numbers that are squares of integers. But it's not that simple. Each number should have a length of  $n$  (and should not have leading zeros), and the multiset of digits of all the numbers should be the same. For example, for 234 and 432, and 11223 and 32211, the multisets of digits are the same, but for 123 and 112233, they are not.

The mathematicians couldn't solve this problem. Can you?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     if (n == 1) {
8         cout << 1 << "\n";
9         return;
10    }
11    cout << 196 << string(n - 3, '0') << "\n";
12    for (int i = 0; i < n / 2; i++) {
13        cout << 1 << string(i, '0') << 6 << string(i, '0') << 9 << string(n - 3 - 2 * i, '0') << "\n";
14        cout << 9 << string(i, '0') << 6 << string(i, '0') << 1 << string(n - 3 - 2 * i, '0') << "\n";
15    }

```

```

16  }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20     int t;
21     cin >> t;
22     while (t--) {
23         solve();
24     }
25     return 0;
26 }
```

### 396: Divisibility Test

- Time limit: 3 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

Daisy has recently learned divisibility rules for integers and she is fascinated by them. One of the tests she learned is a divisibility test by 3. You can find a sum of all digits of a decimal number and check if the resulting sum is divisible by 3. Moreover, the resulting sum of digits is congruent modulo 3 to the original number - the remainder modulo 3 is preserved. For example,  $75 \equiv 7 + 5 \pmod{3}$ . Daisy is specifically interested in such remainder preserving divisibility tests.

There are more examples like that that she learned for decimal integers (integers base 10):

To test divisibility modulo 11, find an alternating sum of digits. Counting digits from the last (least significant) digit, add digits on odd positions (the last, 3rd to the last, etc) and subtract digits on even positions (2nd to the last, 4th to the last, etc) to get the sum that is congruent modulo 11 to the original number. For example,  $123 \equiv 1 - 2 + 3 \pmod{11}$ .

To test divisibility modulo 4, keep the last two digits. Their value is congruent modulo 4 to the original number. For example,  $876543 \equiv 43 \pmod{4}$ .

To test divisibility modulo 7, find an alternating sum of groups of three digits. For example,  $4389328 \equiv 4 - 389 + 328 \pmod{7}$ .

Similar tests can be found in other bases. For example, to test divisibility modulo 5 for octal numbers (base 8), find an alternating sum of groups of two digits. For example,  $1234_8 \equiv -12_8 + 34_8 \pmod{5}$ .

Daisy wants to find such rules for a given base  $b$ . She is interested in three kinds of divisibility rules:

Kind 1 - take the last  $k$  digits of an integer in base  $b$ .

Kind 2 - take a sum of groups of  $k$  digits of an integer in base  $b$ .

Kind 3 - take an alternating sum of groups of  $k$  digits of an integer in base  $b$ .

It is not always possible to find such a divisibility rule. For example, in base 10 there is no such test for divisibility modulo 6, even though different approaches to testing divisibility by 6 exist.

Given base  $b$  and modulo  $n$ , Daisy wants to know the smallest group size  $k$  for which such a divisibility test exists.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int b, n;
6     cin >> b >> n;
7     int p = 1;
8     for (int k = 1; k <= n; k++) {
9         p = 1LL * p * b % n;
10        if (p == 0) {
11            cout << 1 << " " << k << "\n";
12            return;
13        }
14        if (p == 1) {
15            cout << 2 << " " << k << "\n";
16            return;
17        }
18        if (p == n - 1) {
19            cout << 3 << " " << k << "\n";
20            return;
21        }
22    }
23    cout << 0 << "\n";
24 }
25 int main() {
26     ios::sync_with_stdio(false);
27     cin.tie(nullptr);
28     int t;
29     cin >> t;
30     while (t--) {
31         solve();
32     }
33     return 0;
34 }
```

## 397: XOR Construction

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given  $n - 1$  integers  $a_1, a_2, \dots, a_{n-1}$ .

Your task is to construct an array  $b_1, b_2, \dots, b_n$  such that:

every integer from 0 to  $n - 1$  appears in  $b$  exactly once;

for every  $i$  from 1 to  $n - 1$ ,  $b_i \oplus b_{i+1} = a_i$  (where  $\oplus$  denotes the bitwise XOR operator).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n - 1; i++) {
11        int x;
12        cin >> x;
13        a[i + 1] = a[i] ^ x;
14    }
15    int k = __builtin_ctz(n);
16    vector<int> cnt(2 * n + 1), f(2 * n + 1);
17    for (int i = 0; i < n; i++) {
18        cnt[a[i]] += (k + 1);
19        f[a[i]] += a[i];
20    }
21    int x = -1;
22    for (int i = 0; ; i++) {
23        if (cnt[i] == (1 << k)) {
24            x = f[i] ^ (n - 1);
25            break;
26        }
27    }
28    for (int i = 0; i < n; i++) {
29        a[i] ^= x;
30        cout << a[i] << " \n"[i == n - 1];
31    }
32    return 0;
33 }
```

### 398: Suspicious logarithms

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Let  $f(x)$  be the floor of the binary logarithm of  $x$ . In other words,  $f(x)$  is largest non-negative integer  $y$ , such that  $2^y$  does not exceed  $x$ .

Let  $g(x)$  be the floor of the logarithm of  $x$  with base  $f(x)$ . In other words,  $g(x)$  is the largest non-negative integer  $z$ , such that  $f(x)^z$  does not exceed  $x$ .

You are given  $q$  queries. The  $i$ -th query consists of two integers  $l_i$  and  $r_i$ . The answer to the query is the sum of  $g(k)$  across all integers  $k$ , such that  $l_i \leq k \leq r_i$ . Since the answers might be large, print them modulo  $10^9 + 7$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 Z get(i64 n) {
35     Z ans = 0;
36     for (int x = 2; (1LL << x) <= n; x++) {
37         i64 l = (1LL << x), r = min(n, 2 * l - 1);
38         i64 p;
39         for (i64 p = x; p <= r; p *= x) {
40             i64 L = max(l, p);
41             if (L <= r) {
42                 ans += r - L + 1;
43             }
44             if (p > r / x) {
45                 break;
46             }
47         }
48     }
49     return ans;
50 }
```

```

51 void solve() {
52     i64 l, r;
53     cin >> l >> r;
54     Z ans = 0;
55     ans = get(r) - get(l - 1);
56     cout << ans << "\n";
57 }
58 int main() {
59     ios::sync_with_stdio(false);
60     cin.tie(nullptr);
61     int t;
62     cin >> t;
63     while (t--) {
64         solve();
65     }
66     return 0;
67 }
```

### 399: Vasilije Loves Number Theory

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vasilije is a smart student and his discrete mathematics teacher Sonja taught him number theory very well.

He gave Ognjen a positive integer  $n$ .

Denote  $d(n)$  as the number of positive integer divisors of  $n$ , and denote  $\gcd(a, b)$  as the largest integer  $g$  such that  $a$  is divisible by  $g$  and  $b$  is divisible by  $g$ .

After that, he gave Ognjen  $q$  queries, and there are 2 types of queries.

1,  $x$  - set  $n$  to  $n \cdot x$ , and then answer the following question: does there exist a positive integer  $a$  such that  $\gcd(a, n) = 1$ , and  $d(n \cdot a) = n$ ?

2 - reset  $n$  to its initial value (before any queries).

Note that  $n$  does not get back to its initial value after the type 1 query.

Since Ognjen is afraid of number theory, Vasilije promised him that after each query,  $d(n) \leq 10^9$ , however, even with that constraint, he still needs your help with this problem.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
```

```
4  vector<int> minp, primes;
5  constexpr int V = 1E6;
6  int cnt[V + 1];
7  void sieve(int n) {
8      minp.assign(n + 1, 0);
9      primes.clear();
10     for (int i = 2; i <= n; i++) {
11         if (minp[i] == 0) {
12             minp[i] = i;
13             primes.push_back(i);
14         }
15         for (auto p : primes) {
16             if (i * p > n) {
17                 break;
18             }
19             minp[i * p] = p;
20             if (p == minp[i]) {
21                 break;
22             }
23         }
24     }
25 }
26 void solve() {
27     int n, q;
28     cin >> n >> q;
29     vector<int> s;
30     int d = 1;
31     auto add = [&](int x, int t = 1) {
32         while (x > 1) {
33             int p = minp[x];
34             x /= p;
35             d /= cnt[p] + 1;
36             cnt[p] += t;
37             d *= cnt[p] + 1;
38         }
39     };
40     s.push_back(n);
41     add(n);
42     while (q--) {
43         int o;
44         cin >> o;
45         if (o == 1) {
46             int x;
47             cin >> x;
48             s.push_back(x);
49             add(x);
50             int v = 1;
51             for (auto x : s) {
52                 v = 1LL * v * x % d;
53             }
54             if (v == 0) {
55                 cout << "YES\n";
56             } else {
57                 cout << "NO\n";
58             }
59         } else {
60             while (s.size() > 1) {
61                 add(s.back(), -1);
62                 s.pop_back();
63             }
64         }
65     }
66     while (s.size() > 0) {
67         add(s.back(), -1);
```

```

68         s.pop_back();
69     }
70     cout << "\n";
71 }
72 int main() {
73     ios::sync_with_stdio(false);
74     cin.tie(nullptr);
75     sieve(V);
76     int t;
77     cin >> t;
78     while (t--) {
79         solve();
80     }
81     return 0;
82 }
```

## 400: Sum of XOR Functions

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  of length  $n$  consisting of non-negative integers.

You have to calculate the value of  $\sum_{l=1}^n \sum_{r=l}^n f(l, r) \cdot (r-l+1)$ , where  $f(l, r)$  is  $a_l \oplus a_{l+1} \oplus \dots \oplus a_{r-1} \oplus a_r$  (the character  $\oplus$  denotes bitwise XOR).

Since the answer can be very large, print it modulo 998244353.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
}
```

```

20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int n;
38     cin >> n;
39     vector<int> s(n + 1);
40     for (int i = 0; i < n; i++) {
41         int a;
42         cin >> a;
43         s[i + 1] = s[i] ^ a;
44     }
45     Z ans = 0;
46     Z s1[30][2], s2[30][2];
47     for (int i = 0; i <= n; i++) {
48         for (int j = 0; j < 30; j++) {
49             int x = s[i] >> j & 1;
50             ans += (i * s1[j][!x] - s2[j][!x]) * (1 << j);
51             s1[j][x] += 1;
52             s2[j][x] += i;
53         }
54     }
55     cout << ans << "\n";
56     return 0;
57 }
```

## 401: Selling a Menagerie

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are the owner of a menagerie consisting of  $n$  animals numbered from 1 to  $n$ . However, maintaining the menagerie is quite expensive, so you have decided to sell it!

It is known that each animal is afraid of exactly one other animal. More precisely, animal  $i$  is afraid of animal  $a_i$  ( $a_i \neq i$ ). Also, the cost of each animal is known, for animal  $i$  it is equal to  $c_i$ .

You will sell all your animals in some fixed order. Formally, you will need to choose some permutation<sup>†</sup>  $p_1, p_2, \dots, p_n$ , and sell animal  $p_1$  first, then animal  $p_2$ , and so on, selling animal  $p_n$  last.

When you sell animal  $i$ , there are two possible outcomes:

If animal  $a_i$  was sold before animal  $i$ , you receive  $c_i$  money for selling animal  $i$ .

If animal  $a_i$  was not sold before animal  $i$ , you receive  $2 \cdot c_i$  money for selling animal  $i$ . (Surprisingly, animals that are currently afraid are more valuable).

Your task is to choose the order of selling the animals in order to maximize the total profit.

For example, if  $a = [3, 4, 4, 1, 3]$ ,  $c = [3, 4, 5, 6, 7]$ , and the permutation you choose is  $[4, 2, 5, 1, 3]$ , then:

The first animal to be sold is animal 4. Animal  $a_4 = 1$  was not sold before, so you receive  $2 \cdot c_4 = 12$  money for selling it.

The second animal to be sold is animal 2. Animal  $a_2 = 4$  was sold before, so you receive  $c_2 = 4$  money for selling it.

The third animal to be sold is animal 5. Animal  $a_5 = 3$  was not sold before, so you receive  $2 \cdot c_5 = 14$  money for selling it.

The fourth animal to be sold is animal 1. Animal  $a_1 = 3$  was not sold before, so you receive  $2 \cdot c_1 = 6$  money for selling it.

The fifth animal to be sold is animal 3. Animal  $a_3 = 4$  was sold before, so you receive  $c_3 = 5$  money for selling it.

Your total profit, with this choice of permutation, is  $12 + 4 + 14 + 6 + 5 = 41$ . Note that 41 is not the maximum possible profit in this example.

<sup>†</sup> A permutation of length  $n$  is an array consisting of  $n$  distinct integers from 1 to  $n$  in any order. For example,  $[2, 3, 1, 5, 4]$  is a permutation, but  $[1, 2, 2]$  is not a permutation (2 appears twice in the array) and  $[1, 3, 4]$  is also not a permutation ( $n = 3$ , but 4 is present in the array).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10        a[i]--;
11    }
12    vector<int> c(n);
13    for (int i = 0; i < n; i++) {

```

```

14         cin >> c[i];
15     }
16     vector<int> deg(n);
17     for (int i = 0; i < n; i++) {
18         deg[a[i]]++;
19     }
20     vector<int> ans;
21     vector<bool> vis(n);
22     auto dfs = [&](auto self, int x) -> void {
23         ans.push_back(x);
24         vis[x] = true;
25         if (--deg[a[x]] == 0) {
26             self(self, a[x]);
27         }
28     };
29     for (int i = 0; i < n; i++) {
30         if (deg[i] == 0 && !vis[i]) {
31             dfs(dfs, i);
32         }
33     }
34     for (int i = 0; i < n; i++) {
35         if (vis[i]) {
36             continue;
37         }
38         vector<int> b, ca;
39         for (int j = i; !vis[j]; j = a[j]) {
40             vis[j] = true;
41             b.push_back(j);
42             ca.push_back(c[j]);
43         }
44         int p = min_element(ca.begin(), ca.end()) - ca.begin();
45         for (int j = 0; j < b.size(); j++) {
46             ans.push_back(b[(j + 1 + p) % b.size()]);
47         }
48     }
49     for (int i = 0; i < n; i++) {
50         cout << ans[i] + 1 << " \n"[i == n - 1];
51     }
52 }
53 int main() {
54     ios::sync_with_stdio(false);
55     cin.tie(nullptr);
56     int t;
57     cin >> t;
58     while (t--) {
59         solve();
60     }
61     return 0;
62 }
```

## 402: Yet Another Sorting Problem

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Petya has an array of integers  $a_1, a_2, \dots, a_n$ . He only likes sorted arrays. Unfortunately, the given array

could be arbitrary, so Petya wants to sort it.

Petya likes to challenge himself, so he wants to sort array using only 3-cycles. More formally, in one operation he can pick 3 pairwise distinct indices  $i, j$ , and  $k$  ( $1 \leq i, j, k \leq n$ ) and apply  $i \rightarrow j \rightarrow k \rightarrow i$  cycle to the array  $a$ . It simultaneously places  $a_i$  on position  $j$ ,  $a_j$  on position  $k$ , and  $a_k$  on position  $i$ , without changing any other element.

For example, if  $a$  is  $[10, 50, 20, 30, 40, 60]$  and he chooses  $i = 2, j = 1, k = 5$ , then the array becomes  $[50, 40, 20, 30, 10, 60]$ .

Petya can apply arbitrary number of 3-cycles (possibly, zero). You are to determine if Petya can sort his array  $a$ , i. e. make it non-decreasing.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n), cnt(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10        a[i]--;
11        cnt[a[i]]++;
12    }
13    for (int i = 0; i < n; i++) {
14        if (!cnt[i]) {
15            cout << "YES\n";
16            return;
17        }
18    }
19    int res = n & 1;
20    vector<bool> vis(n);
21    for (int i = 0; i < n; i++) {
22        if (vis[i]) {
23            continue;
24        }
25        for (int j = i; !vis[j]; j = a[j]) {
26            vis[j] = true;
27        }
28        res ^= 1;
29    }
30    cout << (res ? "NO" : "YES") << "\n";
31 }
32 int main() {
33     ios::sync_with_stdio(false);
34     cin.tie(nullptr);
35     int t;
36     cin >> t;
37     while (t--) {
38         solve();
39     }
40     return 0;

```

```
41 }
```

## 403: Madoka and the Best School in Russia

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Madoka is going to enroll in “TSUNS PTU”. But she stumbled upon a difficult task during the entrance computer science exam:

A number is called good if it is a multiple of  $d$ .

A number is called beatiful if it is good and it cannot be represented as a product of two good numbers.

Notice that a beautiful number must be good.

Given a good number  $x$ , determine whether it can be represented in at least two different ways as a product of several (possibly, one) beautiful numbers. Two ways are different if the sets of numbers used are different.

Solve this problem for Madoka and help her to enroll in the best school in Russia!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 bool isprime(int n) {
5     if (n <= 1) {
6         return false;
7     }
8     for (int i = 2; i * i <= n; i++) {
9         if (n % i == 0) {
10             return false;
11         }
12     }
13     return true;
14 }
15 void solve() {
16     int x, d;
17     cin >> x >> d;
18     bool ok;
19     if (x % (1LL * d * d) != 0) {
20         ok = false;
21     } else {
22         x /= d * d;
23         if (isprime(d)) {
```

```

24         while (x % d == 0) {
25             x /= d;
26         }
27         if (x == 1 || isprime(x)) {
28             ok = false;
29         } else {
30             ok = true;
31         }
32     } else {
33         int s = sqrt(d);
34         if (s * s == d && isprime(s)) {
35             int t = 4;
36             while (x % s == 0) {
37                 x /= s;
38                 t++;
39             }
40             if (x == 1) {
41                 if (t == 4 || t == 5 || t == 7) {
42                     ok = false;
43                 } else {
44                     ok = true;
45                 }
46             } else {
47                 if (t == 4 && isprime(x)) {
48                     ok = false;
49                 } else {
50                     ok = true;
51                 }
52             }
53         } else {
54             if (x == 1 || isprime(x)) {
55                 ok = false;
56             } else {
57                 ok = true;
58             }
59         }
60     }
61     cout << (ok ? "YES" : "NO") << "\n";
62 }
63 int main() {
64     ios::sync_with_stdio(false);
65     cin.tie(nullptr);
66     int t;
67     cin >> t;
68     while (t--) {
69         solve();
70     }
71     return 0;
72 }
73 }
```

**404: Remove Extra One**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a permutation  $p$  of length  $n$ . Remove one element from permutation to make the number of records the maximum possible.

We remind that in a sequence of numbers  $a_1, a_2, \dots, a_k$  the element  $a_i$  is a record if for every integer  $j$  ( $1 \leq j < i$ ) the following holds:  $a_j < a_i$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> p(n);
10    for (int i = 0; i < n; i++) {
11        cin >> p[i];
12    }
13    int mx[2] {-1, -1};
14    vector<int> v(n);
15    for (int i = 0; i < n; i++) {
16        if (mx[0] == -1 || p[i] > p[mx[0]]) {
17            v[i]--;
18            mx[1] = mx[0];
19            mx[0] = i;
20        } else if (mx[1] == -1 || p[i] > p[mx[1]]) {
21            v[mx[0]]++;
22            mx[1] = i;
23        }
24    }
25    int k = 0;
26    for (int i = 0; i < n; i++) {
27        if (v[i] > v[k] || (v[i] == v[k] && p[i] < p[k])) {
28            k = i;
29        }
30    }
31    cout << p[k] << "\n";
32    return 0;
33 }
```

## 405: XK Segments

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

While Vasya finished eating his piece of pizza, the lesson has already started. For being late for the lesson, the teacher suggested Vasya to solve one interesting problem. Vasya has an array  $a$  and integer

x. He should find the number of different ordered pairs of indexes  $(i, j)$  such that  $a_i \leq a_j$  and there are exactly k integers y such that  $a_i \leq y \leq a_j$  and y is divisible by x.

In this problem it is meant that pair  $(i, j)$  is equal to  $(j, i)$  only if  $i$  is equal to  $j$ . For example pair  $(1, 2)$  is not the same as  $(2, 1)$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, x, k;
8     cin >> n >> x >> k;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    sort(a.begin(), a.end());
14    i64 ans = 0;
15    for (int i = 0, l = 0, r = 0; i < n; i++) {
16        while (l < n && a[i] / x - (a[l] - 1) / x > k) {
17            l++;
18        }
19        while (r < n && a[i] / x - (a[r] - 1) / x >= k && a[r] <= a[i]) {
20            r++;
21        }
22        ans += r - l;
23    }
24    cout << ans << "\n";
25    return 0;
26 }
```

## 406: Marco and GCD Sequence

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In a dream Marco met an elderly man with a pair of black glasses. The man told him the key to immortality and then disappeared with the wind of time.

When he woke up, he only remembered that the key was a sequence of positive integers of some length  $n$ , but forgot the exact sequence. Let the elements of the sequence be  $a_1, a_2, \dots, a_n$ . He remembered that he calculated  $\gcd(a_i, a_{i+1}, \dots, a_j)$  for every  $1 \leq i \leq j \leq n$  and put it into a set  $S$ .  $\gcd$  here means the greatest common divisor.

Note that even if a number is put into the set S twice or more, it only appears once in the set.

Now Marco gives you the set S and asks you to help him figure out the initial sequence. If there are many solutions, print any of them. It is also possible that there are no sequences that produce the set S, in this case print -1.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int m;
8     cin >> m;
9     vector<int> s(m);
10    for (int i = 0; i < m; i++) {
11        cin >> s[i];
12    }
13    for (int i = 0; i < m; i++) {
14        if (s[i] % s[0] != 0) {
15            cout << -1 << "\n";
16            return 0;
17        }
18    }
19    cout << 2 * m << "\n";
20    for (int i = 0; i < m; i++) {
21        cout << s[0] << " " << s[i] << " \n"[i == m - 1];
22    }
23    return 0;
24 }
```

## 407: Ralph And His Magic Field

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Ralph has a magic field which is divided into  $n$   $m$  blocks. That is to say, there are  $n$  rows and  $m$  columns on the field. Ralph can put an integer in each block. However, the magic field doesn't always work properly. It works only if the product of integers in each row and each column equals to  $k$ , where  $k$  is either 1 or -1.

Now Ralph wants you to figure out the number of ways to put numbers in each block in such a way that the magic field works properly. Two ways are considered different if and only if there exists at least one block where the numbers in the first way and in the second way are different. You are asked to output the answer modulo  $1000000007 = 10^9 + 7$ .

Note that there is no range of the numbers to put in the blocks, but we can prove that the answer is not infinity.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     i64 n, m, k;
38     cin >> n >> m >> k;
39     if (k == -1 && (n + m) % 2 != 0) {
40         cout << 0 << "\n";
41         return 0;
42     }
43     Z ans = power(Z(2), (n - 1) % (P - 1) * ((m - 1) % (P - 1)));
44     cout << ans << "\n";
45     return 0;
46 }
```

### 408: Dual (Hard Version)

- Time limit: 1 second
- Memory limit: 256 megabytes

- Input file: standard input
- Output file: standard output

The only difference between the two versions of this problem is the constraint on the maximum number of operations. You can make hacks only if all versions of the problem are solved.

You are given an array  $a_1, a_2, \dots, a_n$  of integers (positive, negative or 0). You can perform multiple operations on the array (possibly 0 operations).

In one operation, you choose  $i, j$  ( $1 \leq i, j \leq n$ , they can be equal) and set  $a_i := a_i + a_j$  (i.e., add  $a_j$  to  $a_i$ ).

Make the array non-decreasing (i.e.,  $a_i \leq a_{i+1}$  for  $1 \leq i \leq n - 1$ ) in at most 31 operations. You do not need to minimize the number of operations.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 vector<pair<int, int>> work(vector<int> a) {
5     int n = a.size();
6     auto [mn, mx] = minmax_element(a.begin(), a.end());
7     if (*mx + *mn < 0) {
8         for (auto &x : a) {
9             x = -x;
10        }
11        reverse(a.begin(), a.end());
12        auto ans = work(a);
13        for (auto &[x, y] : ans) {
14            x = n - 1 - x;
15            y = n - 1 - y;
16        }
17        return ans;
18    }
19    vector<pair<int, int>> ans;
20    int k = mx - a.begin();
21    int cntneg = 0;
22    for (int i = 0; i < n; i++) {
23        if (a[i] < 0) {
24            cntneg++;
25        }
26    }
27    if (cntneg <= 12) {
28        for (int i = 0; i < n; i++) {
29            if (a[i] < 0) {
30                a[i] += a[k];
31                ans.emplace_back(i, k);
32            }
33        }
34        for (int i = 1; i < n; i++) {
35            a[i] += a[i - 1];
36            ans.emplace_back(i, i - 1);
37        }
38    }
39    return ans;
40    int x = mn - a.begin();

```

```

41     for (int i = 0; i < 5; i++) {
42         a[x] += a[x];
43         ans.emplace_back(x, x);
44     }
45     for (int i = 0; i < n; i++) {
46         if (a[i] > 0) {
47             a[i] += a[x];
48             ans.emplace_back(i, x);
49         }
50     }
51     for (int i = n - 2; i >= 0; i--) {
52         a[i] += a[i + 1];
53         ans.emplace_back(i, i + 1);
54     }
55     return ans;
56 }
57 void solve() {
58     int n;
59     cin >> n;
60     vector<int> a(n);
61     for (int i = 0; i < n; i++) {
62         cin >> a[i];
63     }
64     auto ans = work(a);
65     cout << ans.size() << "\n";
66     for (auto [x, y] : ans) {
67         cout << x + 1 << " " << y + 1 << "\n";
68     }
69 }
70 int main() {
71     ios::sync_with_stdio(false);
72     cin.tie(nullptr);
73     int t;
74     cin >> t;
75     while (t--) {
76         solve();
77     }
78     return 0;
79 }
```

## 409: Lisa and the Martians

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Lisa was kidnapped by martians! It's okay, because she has watched a lot of TV shows about aliens, so she knows what awaits her. Let's call integer martian if it is a non-negative integer and strictly less than  $2^k$ , for example, when  $k = 12$ , the numbers 51, 1960, 0 are martian, and the numbers  $\pi$ ,  $-1$ ,  $\frac{21}{8}$ , 4096 are not.

The aliens will give Lisa  $n$  martian numbers  $a_1, a_2, \dots, a_n$ . Then they will ask her to name any martian number  $x$ . After that, Lisa will select a pair of numbers  $a_i, a_j$  ( $i \neq j$ ) in the given sequence and count

$(a_i \oplus x) \& (a_j \oplus x)$ . The operation  $\oplus$  means Bitwise exclusive OR, the operation  $\&$  means Bitwise And. For example,  $(5 \oplus 17) \& (23 \oplus 17) = (00101_2 \oplus 10001_2) \& (10111_2 \oplus 10001_2) = 10100_2 \& 00110_2 = 00100_2 = 4$ .

Lisa is sure that the higher the calculated value, the higher her chances of returning home. Help the girl choose such  $i, j, x$  that maximize the calculated value.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<int> p(n);
12    iota(p.begin(), p.end(), 0);
13    sort(p.begin(), p.end(),
14        [&](int i, int j) {
15            return a[i] < a[j];
16        });
17    int i = -1, j = -1, x = -1;
18    int ans = 1 << k;
19    for (int t = 1; t < n; t++) {
20        if (ans > (a[p[t]] ^ a[p[t - 1]])) {
21            ans = (a[p[t]] ^ a[p[t - 1]]);
22            i = p[t - 1] + 1;
23            j = p[t] + 1;
24            x = ((1 << k) - 1) ^ (a[p[t]] | a[p[t - 1]]);
25        }
26    }
27    cout << i << " " << j << " " << x << "\n";
28 }
29 int main() {
30     ios::sync_with_stdio(false);
31     cin.tie(nullptr);
32     int t;
33     cin >> t;
34     while (t--) {
35         solve();
36     }
37     return 0;
38 }
```

## 410: Imbalanced Arrays

- Time limit: 2 seconds
- Memory limit: 256 megabytes

- Input file: standard input
- Output file: standard output

Ntarsis has come up with an array  $a$  of  $n$  non-negative integers.

Call an array  $b$  of  $n$  integers imbalanced if it satisfies the following:

$$-n \leq b_i \leq n, b_i \neq 0,$$

there are no two indices  $(i, j)$  ( $1 \leq i, j \leq n$ ) such that  $b_i + b_j = 0$ ,

for each  $1 \leq i \leq n$ , there are exactly  $a_i$  indices  $j$  ( $1 \leq j \leq n$ ) such that  $b_i + b_j > 0$ , where  $i$  and  $j$  are not necessarily distinct.

Given the array  $a$ , Ntarsis wants you to construct some imbalanced array. Help him solve this task, or determine it is impossible.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<int> p(n);
12    iota(p.begin(), p.end(), 0);
13    sort(p.begin(), p.end(), [&](int i, int j) {
14        return a[i] < a[j];
15    });
16    int l = 0, r = n - 1;
17    int pos = 0;
18    vector<int> ans(n);
19    for (int i = n; i > 0; i--) {
20        if (a[p[l]] - pos == 0) {
21            ans[p[l]] = -i;
22            l++;
23        } else if (a[p[r]] - pos == i) {
24            ans[p[r]] = i;
25            pos++;
26            r--;
27        } else {
28            cout << "NO\n";
29            return;
30        }
31    }
32    cout << "YES\n";
33    for (int i = 0; i < n; i++) {
34        cout << ans[i] << " \n"[i == n - 1];
35    }
36 }
```

```

37 int main() {
38     ios::sync_with_stdio(false);
39     cin.tie(nullptr);
40     int t;
41     cin >> t;
42     while (t--) {
43         solve();
44     }
45     return 0;
46 }
```

### 411: Ntarsis' Set

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Ntarsis has been given a set  $S$ , initially containing integers  $1, 2, 3, \dots, 10^{1000}$  in sorted order. Every day, he will remove the  $a_1$ -th,  $a_2$ -th,  $\dots$ ,  $a_n$ -th smallest numbers in  $S$  simultaneously.

What is the smallest element in  $S$  after  $k$  days?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10        a[i]--;
11    }
12    i64 ans = 0;
13    for (int i = 0; i < n; i++) {
14        if (ans >= a[i]) {
15            if (ans >= a[i] + 1LL * i * (k - 1)) {
16                ans += k;
17            } else {
18                ans += (ans - a[i]) / i + 1;
19            }
20        }
21    }
22    cout << ans + 1 << "\n";
23 }
24 int main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
27     int t;
```

```

28     cin >> t;
29     while (t--) {
30         solve();
31     }
32     return 0;
33 }
```

## 412: Vika and Price Tags

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vika came to her favorite cosmetics store “Golden Pear”. She noticed that the prices of  $n$  items have changed since her last visit.

She decided to analyze how much the prices have changed and calculated the difference between the old and new prices for each of the  $n$  items.

Vika enjoyed calculating the price differences and decided to continue this process.

Let the old prices be represented as an array of non-negative integers  $a$ , and the new prices as an array of non-negative integers  $b$ . Both arrays have the same length  $n$ .

In one operation, Vika constructs a new array  $c$  according to the following principle:  $c_i = |a_i - b_i|$ . Then, array  $c$  renamed into array  $b$ , and array  $b$  renamed into array  $a$  at the same time, after which Vika repeats the operation with them.

For example, if  $a = [1, 2, 3, 4, 5, 6, 7]$ ;  $b = [7, 6, 5, 4, 3, 2, 1]$ , then  $c = [6, 4, 2, 0, 2, 4, 6]$ . Then,  $a = [7, 6, 5, 4, 3, 2, 1]$ ;  $b = [6, 4, 2, 0, 2, 4, 6]$ .

Vika decided to call a pair of arrays  $a, b$  dull if after some number of such operations all elements of array  $a$  become zeros.

Output “YES” if the original pair of arrays is dull, and “NO” otherwise.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n), b(n);
```

```

8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    for (int i = 0; i < n; i++) {
12        cin >> b[i];
13    }
14    int t = -1;
15    for (int i = 0; i < n; i++) {
16        if (a[i] == 0 && b[i] == 0) {
17            continue;
18        }
19        int v = 0;
20        int x = a[i], y = b[i];
21        while (x != 0) {
22            y %= (2 * x);
23            tie(x, y) = make_pair(y, abs(x - y));
24            v = (v + 1) % 3;
25        }
26        if (t != -1 && t != v) {
27            cout << "NO\n";
28            return;
29        }
30        t = v;
31    }
32    cout << "YES\n";
33 }
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int t;
38     cin >> t;
39     while (t--) {
40         solve();
41     }
42     return 0;
43 }
```

### 413: Rudolf and Snowflakes (hard version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. The only difference is that in this version  $n \leq 10^{18}$ .

One winter morning, Rudolf was looking thoughtfully out the window, watching the falling snowflakes. He quickly noticed a certain symmetry in the configuration of the snowflakes. And like a true mathematician, Rudolf came up with a mathematical model of a snowflake.

He defined a snowflake as an undirected graph constructed according to the following rules:

Initially, the graph has only one vertex.

Then, more vertices are added to the graph. The initial vertex is connected by edges to  $k$  new vertices ( $k > 1$ ).

Each vertex that is connected to only one other vertex is connected by edges to  $k$  more new vertices. This step should be done at least once.

The smallest possible snowflake for  $k = 4$  is shown in the figure.

After some mathematical research, Rudolf realized that such snowflakes may not have any number of vertices. Help Rudolf check whether a snowflake with  $n$  vertices can exist.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     i64 n;
6     cin >> n;
7     for (int i = 2; i <= 60; i++) {
8         int d;
9         if (i == 2) {
10             d = sqrt(n);
11         } else if (i == 3) {
12             d = cbrt(n);
13         } else {
14             d = pow(n, 1.0 / i);
15         }
16         if (d == 1) {
17             continue;
18         }
19         i64 res = 0;
20         i64 p = 1;
21         for (int t = 0; t <= i; t++) {
22             if (t) {
23                 p *= d;
24             }
25             res += p;
26         }
27         if (res == n) {
28             cout << "YES\n";
29             return;
30         }
31     }
32     cout << "NO\n";
33 }
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     cout << fixed << setprecision(10);
38     int t;
39     cin >> t;
40     while (t--) {
41         solve();
42     }
43     return 0;
44 }
```

## 414: Rating System

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are developing a rating system for an online game. Every time a player participates in a match, the player's rating changes depending on the results.

Initially, the player's rating is 0. There are  $n$  matches; after the  $i$ -th match, the rating change is equal to  $a_i$  (the rating increases by  $a_i$  if  $a_i$  is positive, or decreases by  $|a_i|$  if it's negative. There are no zeros in the sequence  $a$ ).

The system has an additional rule: for a fixed integer  $k$ , if a player's rating has reached the value  $k$ , it will never fall below it. Formally, if a player's rating at least  $k$ , and a rating change would make it less than  $k$ , then the rating will decrease to exactly  $k$ .

Your task is to determine the value  $k$  in such a way that the player's rating after all  $n$  matches is the maximum possible (among all integer values of  $k$ ). If there are multiple possible answers, you can print any of them.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    i64 seg = 0;
12    i64 sum = 0;
13    i64 ans = 0;
14    i64 max = 0;
15    for (int i = 0; i < n; i++) {
16        sum += a[i];
17        max = max(max, sum);
18        if (sum - max < seg) {
19            seg = sum - max;
20            ans = max;
21        }
22    }
23    cout << ans << "\n";
24 }
25 int main() {
```

```

26     ios::sync_with_stdio(false);
27     cin.tie(nullptr);
28     int t;
29     cin >> t;
30     while (t--) {
31         solve();
32     }
33     return 0;
34 }
```

**misc****415: Collapsing Strings**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given  $n$  strings  $s_1, s_2, \dots, s_n$ , consisting of lowercase Latin letters. Let  $|x|$  be the length of string  $x$ .

Let a collapse  $C(a, b)$  of two strings  $a$  and  $b$  be the following operation:

if  $a$  is empty,  $C(a, b) = b$ ;

if  $b$  is empty,  $C(a, b) = a$ ;

if the last letter of  $a$  is equal to the first letter of  $b$ , then  $C(a, b) = C(a_{1,|a|-1}, b_{2,|b|})$ , where  $s_{l,r}$  is the substring of  $s$  from the  $l$ -th letter to the  $r$ -th one;

otherwise,  $C(a, b) = a + b$ , i. e. the concatenation of two strings.

Calculate  $\sum_{i=1}^n \sum_{j=1}^n |C(s_i, s_j)|$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int N = 1E6 + 10;
5 int tot = 1;
6 int trie[N][26];
7 int cnt[N];
8 int main() {
9     ios::sync_with_stdio(false);
10    cin.tie(nullptr);
11    int n;
```

```

12     cin >> n;
13     vector<string> s(n);
14     i64 ans = 0;
15     for (int i = 0; i < n; i++) {
16         cin >> s[i];
17         ans += s[i].size();
18         int p = 1;
19         for (auto c : s[i]) {
20             int &q = trie[p][c - 'a'];
21             if (!q) {
22                 q = ++tot;
23             }
24             p = q;
25             cnt[p] += 1;
26         }
27     }
28     ans *= 2 * n;
29     for (int i = 0; i < n; i++) {
30         reverse(s[i].begin(), s[i].end());
31         int p = 1;
32         for (auto c : s[i]) {
33             int &q = trie[p][c - 'a'];
34             if (!q) {
35                 break;
36             }
37             p = q;
38             ans -= 2 * cnt[p];
39         }
40     }
41     cout << ans << "\n";
42     return 0;
43 }
```

## 416: Unusual Entertainment

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A tree is a connected graph without cycles.

A permutation is an array consisting of  $n$  distinct integers from 1 to  $n$  in any order. For example,  $[5, 1, 3, 2, 4]$  is a permutation, but  $[2, 1, 1]$  is not a permutation (as 1 appears twice in the array) and  $[1, 3, 2, 5]$  is also not a permutation (as  $n = 4$ , but 5 is present in the array).

After a failed shoot in the BrMeast video, Alex fell into depression. Even his birthday did not make him happy. However, after receiving a gift from Timofey, Alex's mood suddenly improved. Now he spent days playing with the gifted constructor. Recently, he came up with an unusual entertainment.

Alex builds a tree from his constructor, consisting of  $n$  vertices numbered from 1 to  $n$ , with the root at vertex 1. Then he writes down each integer from 1 to  $n$  in some order, obtaining a permutation  $p$ . After

that, Alex comes up with  $q$  triples of integers  $l, r, x$ . For each triple, he tries to determine if there is at least one descendant of vertex  $x$  among the vertices  $p_l, p_{l+1}, \dots, p_r$ .

A vertex  $u$  is a descendant of vertex  $v$  if and only if  $\text{dist}(1, v) + \text{dist}(v, u) = \text{dist}(1, u)$ , where  $\text{dist}(a, b)$  is the distance between vertices  $a$  and  $b$ . In other words, vertex  $v$  must be on the path from the root to vertex  $u$ .

Alex told Zakhar about this entertainment. Now Alex tells his friend  $q$  triples as described above, hoping that Zakhar can check for the presence of a descendant. Zakhar is very sleepy, so he turned to you for help. Help Zakhar answer all of Alex's questions and finally go to sleep.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, q;
6     cin >> n >> q;
7     vector<vector<int>> adj(n);
8     for (int i = 1; i < n; i++) {
9         int u, v;
10        cin >> u >> v;
11        u--, v--;
12        adj[u].push_back(v);
13        adj[v].push_back(u);
14    }
15    vector<int> invp(n);
16    for (int i = 0; i < n; i++) {
17        int p;
18        cin >> p;
19        p--;
20        invp[p] = i;
21    }
22    vector<vector<array<int, 3>>> qry(n);
23    vector<int> ans(q);
24    for (int i = 0; i < q; i++) {
25        int l, r, x;
26        cin >> l >> r >> x;
27        x--, l--;
28        qry[x].push_back({l, r, i});
29    }
30    vector<set<int>> s(n);
31    auto dfs = [&](auto self, int x, int p) -> void {
32        s[x].insert(invp[x]);
33        for (auto y : adj[x]) {
34            if (y == p) {
35                continue;
36            }
37            self(self, y, x);
38            if (s[y].size() > s[x].size()) {
39                swap(s[x], s[y]);
40            }
41            s[x].merge(s[y]);
42        }
43    };
}
```

```

43     for (auto [l, r, i] : qry[x]) {
44         auto it = s[x].lower_bound(l);
45         if (it != s[x].end() && *it < r) {
46             ans[i] = 1;
47         }
48     }
49 }
50 dfs(dfs, 0, -1);
51 for (int i = 0; i < q; i++) {
52     if (ans[i]) {
53         cout << "YES\n";
54     } else {
55         cout << "NO\n";
56     }
57 }
58 }
59 int main() {
60     ios::sync_with_stdio(false);
61     cin.tie(nullptr);
62     int t;
63     cin >> t;
64     while (t--) {
65         solve();
66     }
67     return 0;
68 }
```

### 417: Nephren gives a riddle

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Nephren is playing a game with little leprechauns.

She gives them an infinite array of strings, f0... .

f0 is “What are you doing at the end of the world? Are you busy? Will you save us?”.

She wants to let more people know about it, so she defines fi = “What are you doing while sending”fi - 1”? Are you busy? Will you send “fi - 1”? for all i >= 1.

For example, f1 is

“What are you doing while sending”What are you doing at the end of the world? Are you busy? Will you save us?”? Are you busy? Will you send”What are you doing at the end of the world? Are you busy? Will you save us?”?. Note that the quotes in the very beginning and in the very end are for clarity and are not a part of f1.

It can be seen that the characters in fi are letters, question marks, (possibly) quotation marks and spaces.

Nephren will ask the little leprechauns q times. Each time she will let them find the k-th character of fn. The characters are indexed starting from 1. If fn consists of less than k characters, output '?' (without quotes).

Can you answer her queries?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 string f = "What are you doing at the end of the world? Are you busy? Will you save us?";
5 string a = "What are you doing while sending \'";
6 string b = "? Are you busy? Will you send \'";
7 string c = "\'?";
8 int main() {
9     ios::sync_with_stdio(false);
10    cin.tie(nullptr);
11    int q;
12    cin >> q;
13    constexpr int N = 1E5;
14    constexpr i64 inf = 1E18;
15    vector<i64> len(N + 1);
16    len[0] = f.size();
17    for (int i = 1; i <= N; i++) {
18        len[i] = min(inf, 2 * len[i - 1] + int(a.size() + b.size() + c.size()));
19    }
20    auto get = [&](auto self, int n, i64 k) {
21        if (k >= len[n]) {
22            return '.';
23        }
24        if (n == 0) {
25            return f[k];
26        }
27        if (k < a.size()) {
28            return a[k];
29        }
30        k -= a.size();
31        if (k < len[n - 1]) {
32            return self(self, n - 1, k);
33        }
34        k -= len[n - 1];
35        if (k < b.size()) {
36            return b[k];
37        }
38        k -= b.size();
39        if (k < len[n - 1]) {
40            return self(self, n - 1, k);
41        }
42        k -= len[n - 1];
43        return c[k];
44    };
45    for (int i = 0; i < q; i++) {
46        int n;
47        i64 k;
48        cin >> n >> k;
49        k--;
50        cout << get(get, n, k);
51    }
52    cout << "\n";

```

```

53     return 0;
54 }
```

## 418: Maximum Subsequence

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  consisting of  $n$  integers, and additionally an integer  $m$ . You have to choose some sequence of indices  $b_1, b_2, \dots, b_k$  ( $1 \leq b_1 < b_2 < \dots < b_k \leq n$ ) in such a way that the value of  $\sum a[b_i]$  is maximized. Chosen sequence can be empty.

Print the maximum possible value of  $\sum a[b_i]$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    int ans = 0;
14    int n1 = n / 2;
15    vector<int> x, y;
16    auto dfs = [&](auto self, int i, int goal, int sum, auto &x) {
17        if (i == goal) {
18            x.push_back(sum);
19            return;
20        }
21        self(self, i + 1, goal, sum, x);
22        self(self, i + 1, goal, (sum + a[i]) % m, x);
23    };
24    dfs(dfs, 0, n1, 0, x);
25    dfs(dfs, n1, n, 0, y);
26    sort(x.begin(), x.end());
27    sort(y.begin(), y.end());
28    int i = y.size() - 1;
29    for (auto a : x) {
30        ans = max(ans, a + y.back() - m);
31        while (i >= 0 && a + y[i] >= m) {
32            i--;
33        }
34    }
35}
```

```

34         if (i >= 0) {
35             ans = max(ans, a + y[i]);
36         }
37     }
38     cout << ans << "\n";
39     return 0;
40 }
```

**419: Fancy Number**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A car number in Berland consists of exactly  $n$  digits. A number is called beautiful if it has at least  $k$  equal digits. Vasya wants to change the digits in his car's number so that the number became beautiful. To replace one of  $n$  digits Vasya has to pay the sum of money, equal to the absolute difference between the old digit and the new one.

Help Vasya: find the minimum sum of money he should pay to make the number of his car beautiful. You should also find the resulting beautiful number. If there are several such numbers, then print the lexicographically minimum one.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, k;
8     cin >> n >> k;
9     string s;
10    cin >> s;
11    string t;
12    int ans = 1E9;
13    vector<int> f[10];
14    for (int i = 0; i < n; i++) {
15        f[s[i] - '0'].push_back(i);
16    }
17    for (int x = 0; x <= 9; x++) {
18        string a = s;
19        int cnt = 0;
20        for (int d = 0; d <= 9; d++) {
21            if (x + d <= 9) {
22                for (auto i : f[x + d]) {
23                    if (cnt < k) {
```

```

24             cnt++;
25             a[i] = '0' + x;
26         }
27     }
28 }
29 if (x >= d && d) {
30     reverse(f[x - d].begin(), f[x - d].end());
31     for (auto i : f[x - d]) {
32         if (cnt < k) {
33             cnt++;
34             a[i] = '0' + x;
35         }
36     }
37     reverse(f[x - d].begin(), f[x - d].end());
38 }
39 int res = 0;
40 for (int i = 0; i < n; i++) {
41     res += abs(s[i] - a[i]);
42 }
43 if (res < ans || (res == ans && a < t)) {
44     ans = res;
45     t = a;
46 }
47 }
48 cout << ans << "\n";
49 cout << t << "\n";
50 return 0;
51 }
```

## 420: LuoTianyi and the Floating Islands (Easy Version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the easy version of the problem. The only difference is that in this version  $k \leq \min(n, 3)$ . You can make hacks only if both versions of the problem are solved.

LuoTianyi now lives in a world with  $n$  floating islands. The floating islands are connected by  $n - 1$  undirected air routes, and any two of them can reach each other by passing the routes. That means, the  $n$  floating islands form a tree.

One day, LuoTianyi wants to meet her friends: Chtholly, Nephren, William, .... Totally, she wants to meet  $k$  people. She doesn't know the exact positions of them, but she knows that they are in pairwise distinct islands. She define an island is good if and only if the sum of the distances<sup>†</sup> from it to the islands with  $k$  people is the minimal among all the  $n$  islands.

Now, LuoTianyi wants to know that, if the  $k$  people are randomly set in  $k$  distinct of the  $n$  islands, then what is the expect number of the good islands? You just need to tell her the expect number modulo  $10^9 + 7$ .

<sup>†</sup>The distance between two islands is the minimum number of air routes you need to take to get from one island to the other.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int n, k;
39     cin >> n >> k;
40     vector<vector<int>> adj(n);
41     for (int i = 1; i < n; i++) {
42         int u, v;
43         cin >> u >> v;
44         u--, v--;
45         adj[u].push_back(v);
46         adj[v].push_back(u);
47     }
48     Z ans = 1;
49     if (k % 2 == 0) {
50         Z inv = 1 / comb.binom(n, k);
51         vector<int> siz(n);
52         function<void(int, int)> dfs = [&](int x, int p) {
53             siz[x] = 1;
54             for (auto y : adj[x]) {
55                 if (y == p) {
56                     continue;
57                 }
58                 if (siz[y] < 0) {
59                     siz[y] = -siz[x];
60                     dfs(y, x);
61                 } else {
62                     siz[y] -= siz[x];
63                 }
64             }
65         };
66         dfs(0, -1);
67         cout << inv * ans;
68     } else {
69         cout << 0;
70     }
71 }
```

```

57         }
58         dfs(y, x);
59         siz[x] += siz[y];
60         ans += comb.binom(siz[y], k / 2) * comb.binom(n - siz[y], k / 2) * inv;
61     }
62 }
63 dfs(0, -1);
64 cout << ans << "\n";
65 return 0;
66 }
67 }
```

## 421: Xenia and Bit Operations

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Xenia the beginner programmer has a sequence  $a$ , consisting of  $2n$  non-negative integers:  $a_1, a_2, \dots, a_{2n}$ . Xenia is currently studying bit operations. To better understand how they work, Xenia decided to calculate some value  $v$  for  $a$ .

Namely, it takes several iterations to calculate value  $v$ . At the first iteration, Xenia writes a new sequence  $a_1$  or  $a_2$ ,  $a_3$  or  $a_4$ , ...,  $a_{2n-1}$  or  $a_{2n}$ , consisting of  $2n-1$  elements. In other words, she writes down the bit-wise OR of adjacent elements of sequence  $a$ . At the second iteration, Xenia writes the bitwise exclusive OR of adjacent elements of the sequence obtained after the first iteration. At the third iteration Xenia writes the bitwise OR of the adjacent elements of the sequence obtained after the second iteration. And so on; the operations of bitwise exclusive OR and bitwise OR alternate. In the end, she obtains a sequence consisting of one element, and that element is  $v$ .

Let's consider an example. Suppose that sequence  $a = (1, 2, 3, 4)$ . Then let's write down all the transformations  $(1, 2, 3, 4)$  ( $1 \text{ or } 2 = 3$ ,  $3 \text{ or } 4 = 7$ ) ( $3 \text{ xor } 7 = 4$ ). The result is  $v = 4$ .

You are given Xenia's initial sequence. But to calculate value  $v$  for a given sequence would be too easy, so you are given additional  $m$  queries. Each query is a pair of integers  $p, b$ . Query  $p, b$  means that you need to perform the assignment  $a_p = b$ . After each query, you need to print the new value  $v$  for the new sequence  $a$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
```

```

3  using i64 = long long;
4  int main() {
5      ios::sync_with_stdio(false);
6      cin.tie(nullptr);
7      int n, m;
8      cin >> n >> m;
9      vector<int> a(2 << n);
10     for (int i = 0; i < (1 << n); i++) {
11         cin >> a[1 << n | i];
12     }
13     for (int i = (1 << n) - 1; i; i--) {
14         if ((n - __lg(i)) % 2) {
15             a[i] = a[2 * i] | a[2 * i + 1];
16         } else {
17             a[i] = a[2 * i] ^ a[2 * i + 1];
18         }
19     }
20     while (m--) {
21         int p, b;
22         cin >> p >> b;
23         p--;
24         p += 1 << n;
25         a[p] = b;
26         while (p /= 2) {
27             if ((n - __lg(p)) % 2) {
28                 a[p] = a[2 * p] | a[2 * p + 1];
29             } else {
30                 a[p] = a[2 * p] ^ a[2 * p + 1];
31             }
32         }
33         cout << a[1] << "\n";
34     }
35     return 0;
36 }
```

## 422: Comb

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Having endured all the hardships, Lara Croft finally found herself in a room with treasures. To her surprise she didn't find golden mountains there. Lara looked around and noticed on the floor a painted table  $n \times m$  panels in size with integers written on the panels. There also was a huge number of stones lying by the wall. On the pillar near the table Lara found a guidance note which said that to get hold of the treasures one has to choose some non-zero number of the first panels in each row of the table and put stones on all those panels to push them down. After that she will receive a number of golden coins equal to the sum of numbers written on the chosen panels. Lara quickly made up her mind on how to arrange the stones and was about to start when she noticed an addition to the note in small font below. According to the addition, for the room ceiling not to crush and smash the adventurer, the chosen panels should form a comb. It was explained that the chosen panels form a comb when

the sequence  $c_1, c_2, \dots, c_n$  made from the quantities of panels chosen in each table line satisfies the following property:  $c_1 > c_2 < c_3 > c_4 < \dots$ , i.e. the inequation mark interchanges between the neighboring elements. Now Lara is bewildered and doesn't know what to do. Help her to determine the largest number of coins she can get and survive at the same time.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<i64> a(n, vector<int>(m));
10    for (int i = 0; i < n; i++) {
11        for (int j = 0; j < m; j++) {
12            cin >> a[i][j];
13            if (j) {
14                a[i][j] += a[i][j-1];
15            }
16        }
17    }
18    vector<i64> dp(m);
19    for (int i = 0; i < n; i++) {
20        if (i % 2 == 0) {
21            for (int j = m-1; j; j--) {
22                dp[j] = dp[j-1] + a[i][j];
23            }
24            dp[0] = -1E18;
25            for (int j = m-1; j; j--) {
26                dp[j-1] = max(dp[j-1], dp[j]);
27            }
28        } else {
29            for (int j = 0; j < m-1; j++) {
30                dp[j] = dp[j+1] + a[i][j];
31            }
32            dp[m-1] = -1E18;
33            for (int j = 0; j < m-1; j++) {
34                dp[j+1] = max(dp[j+1], dp[j]);
35            }
36        }
37    }
38    cout << *max_element(dp.begin(), dp.end()) << "\n";
39    return 0;
40 }
```

### 423: Magic Triples (Easy Version)

- Time limit: 4 seconds
- Memory limit: 256 megabytes

- Input file: standard input
- Output file: standard output

This is the easy version of the problem. The only difference is that in this version,  $a_i \leq 10^6$ .

For a given sequence of  $n$  integers  $a$ , a triple  $(i, j, k)$  is called magic if:

$1 \leq i, j, k \leq n$ .

$i, j, k$  are pairwise distinct.

there exists a positive integer  $b$  such that  $a_i \cdot b = a_j$  and  $a_j \cdot b = a_k$ .

Kolya received a sequence of integers  $a$  as a gift and now wants to count the number of magic triples for it. Help him with this task!

Note that there are no constraints on the order of integers  $i, j$  and  $k$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 vector<int> minp, primes;
5 void sieve(int n) {
6     minp.assign(n + 1, 0);
7     primes.clear();
8     for (int i = 2; i <= n; i++) {
9         if (minp[i] == 0) {
10             minp[i] = i;
11             primes.push_back(i);
12         }
13         for (auto p : primes) {
14             if (i * p > n) {
15                 break;
16             }
17             minp[i * p] = p;
18             if (p == minp[i]) {
19                 break;
20             }
21         }
22     }
23 }
24 vector<vector<int>> divs;
25 void solve() {
26     int n;
27     cin >> n;
28     vector<int> a(n);
29     for (int i = 0; i < n; i++) {
30         cin >> a[i];
31     }
32     sort(a.begin(), a.end());
33     i64 ans = 0;
34     for (int i = 0; i < n; i++) {
35         int x = a[i];
36         for (int j = i + 1; j < n; j++) {
37             if (x * a[j] > n) {
38                 break;
39             }
40             for (int k = j + 1; k < n; k++) {
41                 if (x * a[j] * a[k] > n) {
42                     break;
43                 }
44                 ans++;
45             }
46         }
47     }
48 }
```

```

36     int s = 1;
37     for (auto v : primes) {
38         if (v * v * v > x) {
39             break;
40         }
41         int t = 0;
42         while (x % v == 0) {
43             x /= v;
44             t++;
45             if (t % 2 == 0) {
46                 s *= v;
47             }
48         }
49     }
50     if (x > 1) {
51         int u = sqrt(x);
52         if (u * u == x) {
53             s *= u;
54         }
55     }
56     for (auto b : divs[s]) {
57         auto [l1, r1] = equal_range(a.begin(), a.end(), a[i] / b);
58         auto [l2, r2] = equal_range(a.begin(), a.end(), a[i] / (b*b));
59         ans += 1LL * (r1-l1) * (r2-l2);
60     }
61 }
62 for (int i = 0, j = 0; i < n; i = j) {
63     while (j < n && a[i] == a[j]) {
64         j++;
65     }
66     ans += 1LL * (j-i) * (j-i-1) * (j-i-2);
67 }
68 cout << ans << "\n";
69 }
70 int main() {
71     ios::sync_with_stdio(false);
72     cin.tie(nullptr);
73     int n = 4E4;
74     sieve(n);
75     divs.resize(n+1);
76     for (int i = 2; i <= n; i++) {
77         for (int j = i; j <= n; j += i) {
78             divs[j].push_back(i);
79         }
80     }
81     int t;
82     cin >> t;
83     while (t--) {
84         solve();
85     }
86     return 0;
87 }

```

## 424: The Union of k-Segments

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given  $n$  segments on the coordinate axis  $Ox$  and the number  $k$ . The point is satisfied if it belongs to at least  $k$  segments. Find the smallest (by the number of segments) set of segments on the coordinate axis  $Ox$  which contains all satisfied points and no others.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, k;
8     cin >> n >> k;
9     vector<pair<int, int>> a;
10    for (int i = 0; i < n; i++) {
11        int l, r;
12        cin >> l >> r;
13        a.emplace_back(2 * l, 1);
14        a.emplace_back(2 * r + 1, -1);
15    }
16    sort(a.begin(), a.end());
17    int lst = 0, v = 0;
18    vector<pair<int, int>> ans;
19    for (auto [x, t] : a) {
20        if (x > lst && v >= k) {
21            if (!ans.empty() && ans.back().second == lst) {
22                ans.back().second = x;
23            } else {
24                ans.emplace_back(lst, x);
25            }
26        }
27        v += t;
28        lst = x;
29    }
30    cout << ans.size() << "\n";
31    for (auto [l, r] : ans) {
32        cout << l / 2 << " " << (r - 1) / 2 << "\n";
33    }
34    return 0;
35 }
```

## 425: Phone Number

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Alas, finding one's true love is not easy. Masha has been unsuccessful in that yet. Her friend Dasha told Masha about a way to determine the phone number of one's Prince Charming through arithmancy.

The phone number is divined like that. First one needs to write down one's own phone numbers. For

example, let's suppose that Masha's phone number is 12345. After that one should write her favorite digit from 0 to 9 under the first digit of her number. That will be the first digit of the needed number. For example, Masha's favorite digit is 9. The second digit is determined as a half sum of the second digit of Masha's number and the already written down first digit from her beloved one's number. In this case the arithmetic average equals to  $(2 + 9) / 2 = 5.5$ . Masha can round the number up or down, depending on her wishes. For example, she chooses the digit 5. Having written down the resulting digit under the second digit of her number, Masha moves to finding the third digit in the same way, i.e. finding the half sum the the third digit of her number and the second digit of the new number. The result is  $(5 + 3) / 2 = 4$ . In this case the answer is unique. Thus, every  $i$ -th digit is determined as an arithmetic average of the  $i$ -th digit of Masha's number and the  $i - 1$ -th digit of her true love's number. If needed, the digit can be rounded up or down. For example, Masha can get:

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     string s;
8     cin >> s;
9     int n = s.size();
10    vector<int> a(n);
11    for (int i = 0; i < n; i++) {
12        a[i] = s[i] - '0';
13    }
14    array<i64, 10> dp{};
15    dp.fill(1);
16    for (int i = 1; i < n; i++) {
17        array<i64, 10> g{};
18        for (int x = 0; x < 10; x++) {
19            g[(x + a[i]) / 2] += dp[x];
20            if ((x + a[i]) % 2 == 1) {
21                g[(x + a[i]) / 2 + 1] += dp[x];
22            }
23        }
24        dp = g;
25    }
26    auto ans = accumulate(dp.begin(), dp.end(), 0LL);
27    bool ok = true;
28    for (int i = 1; i < n; i++) {
29        if (abs(a[i] - a[i-1]) > 1) {
30            ok = false;
31        }
32    }
33    ans -= ok;
34    cout << ans << "\n";
35    return 0;
36 }
```

## 426: Hyperdrive

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In a far away galaxy there are  $n$  inhabited planets, numbered with numbers from 1 to  $n$ . They are located at large distances from each other, that's why the communication between them was very difficult until on the planet number 1 a hyperdrive was invented. As soon as this significant event took place,  $n - 1$  spaceships were built on the planet number 1, and those ships were sent to other planets to inform about the revolutionary invention.

Paradoxical thought it may be, but the hyperspace is represented as simple three-dimensional Euclidean space. The inhabited planets may be considered fixed points in it, and no two points coincide and no three points lie on the same straight line. The movement of a ship with a hyperdrive between two planets is performed along a straight line at the constant speed, the same for all the ships. That's why the distance in the hyperspace are measured in hyperyears (a ship with a hyperdrive covers a distance of  $s$  hyperyears in  $s$  years).

When the ship reaches an inhabited planet, the inhabitants of the planet dissemble it, make  $n - 2$  identical to it ships with a hyperdrive and send them to other  $n - 2$  planets (except for the one from which the ship arrived). The time to make a new ship compared to the time in which they move from one planet to another is so small that it can be disregarded. New ships are absolutely identical to the ones sent initially: they move at the same constant speed along a straight line trajectory and, having reached a planet, perform the very same mission, i.e. are dissembled to build new  $n - 2$  ships and send them to all the planets except for the one from which the ship arrived. Thus, the process of spreading the important news around the galaxy continues.

However the hyperdrive creators hurried to spread the news about their invention so much that they didn't study completely what goes on when two ships collide in the hyperspace. If two moving ships find themselves at one point, they provoke an explosion of colossal power, leading to the destruction of the galaxy!

Your task is to find the time the galaxy will continue to exist from the moment of the ships' launch from the first planet.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
```

```

2  using namespace std;
3  using i64 = long long;
4  int main() {
5      ios::sync_with_stdio(false);
6      cin.tie(nullptr);
7      int n;
8      cin >> n;
9      vector<int> x(n), y(n), z(n);
10     for (int i = 0; i < n; i++) {
11         cin >> x[i] >> y[i] >> z[i];
12     }
13     auto dist = [&](int i, int j) {
14         return sqrt((x[i] - x[j]) * (x[i] - x[j]) + (y[i] - y[j]) * (y[i] - y[j]) + (z[i]
15             - z[j]) * (z[i] - z[j]));
16     };
17     double ans = 1E9;
18     for (int i = 1; i < n; i++) {
19         for (int j = i+1; j < n; j++) {
20             ans = min(ans, dist(0, i) + dist(i, j) + dist(j, 0));
21         }
22     }
23     ans /= 2;
24     cout << fixed << setprecision(10) << ans << "\n";
25 }
```

## 427: Let's Go Rolling!

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

On a number axis directed from the left rightwards,  $n$  marbles with coordinates  $x_1, x_2, \dots, x_n$  are situated. Let's assume that the sizes of the marbles are infinitely small, that is in this task each of them is assumed to be a material point. You can stick pins in some of them and the cost of sticking in the marble number  $i$  is equal to  $c_i$ , number  $c_i$  may be negative. After you choose and stick the pins you need, the marbles will start to roll left according to the rule: if a marble has a pin stuck in it, then the marble doesn't move, otherwise the marble rolls all the way up to the next marble which has a pin stuck in it and stops moving there. If there is no pinned marble on the left to the given unpinned one, it is concluded that the marble rolls to the left to infinity and you will pay an infinitely large fine for it. If no marble rolled infinitely to the left, then the fine will consist of two summands:

the sum of the costs of stuck pins;

the sum of the lengths of the paths of each of the marbles, that is the sum of absolute values of differences between their initial and final positions.

Your task is to choose and pin some marbles in the way that will make the fine for you to pay as little as possible.

Problem: [link](#)

Solution: [link](#)

```

1 n = int(input())
2 a = []
3 for i in range(n) :
4     a.append(tuple(map(int, input().split())))
5 a.sort()
6 dp = [10 ** 18] * (n+1)
7 dp[0] = 0
8 for i in range(n) :
9     s = 0
10    for j in range(i, n) :
11        s += a[j][0]
12        dp[j+1] = min(dp[j+1], s - (j-i+1) * a[i][0] + a[i][1] + dp[i])
13 print(dp[n])

```

## 428: Old Berland Language

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Berland scientists know that the Old Berland language had exactly  $n$  words. Those words had lengths of  $l_1, l_2, \dots, l_n$  letters. Every word consisted of two letters, 0 and 1. Ancient Berland people spoke quickly and didn't make pauses between the words, but at the same time they could always understand each other perfectly. It was possible because no word was a prefix of another one. The prefix of a string is considered to be one of its substrings that starts from the initial symbol.

Help the scientists determine whether all the words of the Old Berland language can be reconstructed and if they can, output the words themselves.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 n = int(input())
2 l = list(map(int, input().split()))
3 o = list(range(n))
4 o.sort(key = lambda i : -l[i])
5 ans = [''] * n
6 s = 0
7 lst = 10 ** 3
8 for i in o :
9     s >>= lst - l[i]
10    lst = l[i]
11    if i != o[0] :
12        s += 1

```

```

13     if s >= (1 << l[i]) :
14         print('NO')
15         exit()
16     ans[i] = bin(s)[2:]
17     ans[i] = '0' * (l[i] - len(ans[i])) + ans[i]
18 print('YES')
19 print('\n'.join(ans))

```

## 429: Black Cells

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are playing with a really long strip consisting of  $10^{18}$  white cells, numbered from left to right as 0, 1, 2 and so on. You are controlling a special pointer that is initially in cell 0. Also, you have a “Shift” button you can press and hold.

In one move, you can do one of three actions:

move the pointer to the right (from cell  $x$  to cell  $x + 1$ );

press and hold the “Shift” button;

release the “Shift” button: the moment you release “Shift”, all cells that were visited while “Shift” was pressed are colored in black.

Your goal is to color at least  $k$  cells, but there is a restriction: you are given  $n$  segments  $[l_i, r_i]$  - you can color cells only inside these segments, i. e. you can color the cell  $x$  if and only if  $l_i \leq x \leq r_i$  for some  $i$ .

What is the minimum number of moves you need to make in order to color at least  $k$  cells black?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> l(n), r(n);
8     for (int i = 0; i < n; i++) {
9         cin >> l[i];
10    }
11    for (int i = 0; i < n; i++) {

```

```

12     cin >> r[i];
13 }
14 int sum = 0;
15 int ans = 2E9;
16 priority_queue<int, vector<int>, greater<>> h;
17 for (int i = 0; i < n; i++) {
18     sum += r[i] - l[i] + 1;
19     h.push(r[i] - l[i] + 1);
20     while (sum >= k) {
21         int t = k - (sum - (r[i] - l[i] + 1)) + l[i] - 1;
22         ans = min(ans, 2 * int(h.size()) + t);
23         sum -= h.top();
24         h.pop();
25     }
26 }
27 if (ans == 2E9) {
28     ans = -1;
29 }
30 cout << ans << "\n";
31 }
32 int main() {
33     ios::sync_with_stdio(false);
34     cin.tie(nullptr);
35     int t;
36     cin >> t;
37     while (t--) {
38         solve();
39     }
40     return 0;
41 }

```

## 430: Animals

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: input.txt
- Output file: output.txt

Once upon a time DravDe, an outstanding person famous for his professional achievements (as you must remember, he works in a warehouse storing Ogudar-Olok, a magical but non-alcoholic drink) came home after a hard day. That day he had to drink 9875 boxes of the drink and, having come home, he went to bed at once.

DravDe dreamt about managing a successful farm. He dreamt that every day one animal came to him and asked him to let it settle there. However, DravDe, being unimaginably kind, could send the animal away and it went, rejected. There were exactly  $n$  days in DravDes dream and the animal that came on the  $i$ -th day, ate exactly  $c_i$  tons of food daily starting from day  $i$ . But if one day the animal could not get the food it needed, it got really sad. At the very beginning of the dream there were exactly  $X$  tons of food on the farm.

DravDe woke up terrified...

When he retold the dream to you, he couldn't remember how many animals were on the farm by the end of the  $n$ -th day any more, but he did remember that nobody got sad (as it was a happy farm) and that there was the maximum possible amount of the animals. That's the number he wants you to find out.

It should be noticed that the animals arrived in the morning and DravDe only started to feed them in the afternoon, so that if an animal willing to join them is rejected, it can't eat any farm food. But if the animal does join the farm, it eats daily from that day to the  $n$ -th.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 #ifdef ONLINE_JUDGE
5     ifstream fin("input.txt");
6     ofstream fout("output.txt");
7 #else
8     #define fin cin
9     #define fout cout
10 #endif
11 int main() {
12     int n, X;
13     fin >> n >> X;
14     vector<int> c(n);
15     for (int i = 0; i < n; i++) {
16         fin >> c[i];
17         c[i] *= (n - i);
18     }
19     sort(c.begin(), c.end());
20     int ans = 0;
21     for (auto x : c) {
22         if (x <= X) {
23             ans += 1;
24             X -= x;
25         }
26     }
27     fout << ans << "\n";
28     return 0;
29 }
```

### 431: Wonderful Randomized Sum

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Learn, learn and learn again - Valera has to do this every day. He is studying at mathematical school, where math is the main discipline. The mathematics teacher loves her discipline very much and tries to

cultivate this love in children. That's why she always gives her students large and difficult homework. Despite that Valera is one of the best students, he failed to manage with the new homework. That's why he asks for your help. He has the following task. A sequence of  $n$  numbers is given. A prefix of a sequence is the part of the sequence (possibly empty), taken from the start of the sequence. A suffix of a sequence is the part of the sequence (possibly empty), taken from the end of the sequence. It is allowed to sequentially make two operations with the sequence. The first operation is to take some prefix of the sequence and multiply all numbers in this prefix by -1. The second operation is to take some suffix and multiply all numbers in it by -1. The chosen prefix and suffix may intersect. What is the maximum total sum of the sequence that can be obtained by applying the described operations?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    array<int, 3> dp {};
14    for (int i = 0; i < n; i++) {
15        for (int j = 0; j < 3; j++) {
16            dp[j] += a[i] * (j == 1 ? 1 : -1);
17        }
18        for (int j = 1; j < 3; j++) {
19            dp[j] = max(dp[j], dp[j - 1]);
20        }
21    }
22    cout << dp[2] << "\n";
23    return 0;
24 }
```

## 432: String Problem

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Boy Valera likes strings. And even more he likes them, when they are identical. That's why in his spare time Valera plays the following game. He takes any two strings, consisting of lower case Latin letters,

and tries to make them identical. According to the game rules, with each move Valera can change one arbitrary character  $A_i$  in one of the strings into arbitrary character  $B_i$ , but he has to pay for every move a particular sum of money, equal to  $W_i$ . He is allowed to make as many moves as he needs. Since Valera is a very economical boy and never wastes his money, he asked you, an experienced programmer, to help him answer the question: what minimum amount of money should Valera have to get identical strings.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int inf = 1E9;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     string s, t;
9     cin >> s >> t;
10    if (s.size() != t.size()) {
11        cout << -1 << "\n";
12        return 0;
13    }
14    int n = s.size();
15    int m;
16    cin >> m;
17    vector dp(26, vector(26, inf));
18    for (int i = 0; i < 26; i++) {
19        dp[i][i] = 0;
20    }
21    for (int i = 0; i < m; i++) {
22        char a, b;
23        int c;
24        cin >> a >> b >> c;
25        int x = a - 'a', y = b - 'a';
26        dp[x][y] = min(dp[x][y], c);
27    }
28    for (int k = 0; k < 26; k++) {
29        for (int i = 0; i < 26; i++) {
30            for (int j = 0; j < 26; j++) {
31                dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j]);
32            }
33        }
34    }
35    int ans = 0;
36    string c;
37    for (int i = 0; i < n; i++) {
38        int res = inf;
39        int x = s[i] - 'a', y = t[i] - 'a';
40        int z = 0;
41        for (int j = 0; j < 26; j++) {
42            if (dp[x][j] + dp[y][j] < res) {
43                res = dp[x][j] + dp[y][j];
44                z = j;
45            }
46        }
47        c += 'a' + z;

```

```

48         ans = min(inf, ans + res);
49     }
50     if (ans == inf) {
51         cout << -1 << "\n";
52         return 0;
53     }
54     cout << ans << "\n";
55     cout << c << "\n";
56     return 0;
57 }
```

**433: Flea**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

It is known that fleas in Berland can jump only vertically and horizontally, and the length of the jump is always equal to  $s$  centimeters. A flea has found herself at the center of some cell of the checked board of the size  $n \times m$  centimeters (each cell is 1 × 1 centimeters). She can jump as she wishes for an arbitrary number of times, she can even visit a cell more than once. The only restriction is that she cannot jump out of the board.

The flea can count the amount of cells that she can reach from the starting position  $(x, y)$ . Let's denote this amount by  $dx, dy$ . Your task is to find the number of such starting positions  $(x, y)$ , which have the maximum possible value of  $dx, dy$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m, s;
8     cin >> n >> m >> s;
9     i64 ans = 1LL * ((n - 1) / s + 1) * ((n - 1) % s + 1) * ((m - 1) / s + 1) * ((m - 1) %
10        s + 1);
11    cout << ans << "\n";
12 }
```

**434: Shooting Gallery**

- Time limit: 2 seconds

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

One warm and sunny day king Copa decided to visit the shooting gallery, located at the Central Park, and try to win the main prize - big pink plush panda. The king is not good at shooting, so he invited you to help him.

The shooting gallery is an infinite vertical plane with Cartesian coordinate system on it. The targets are points on this plane. Each target is described by its coordinates  $x_i$ , and  $y_i$ , by the time of its appearance  $t_i$  and by the number  $p_i$ , which gives the probability that Copa hits this target if he aims at it.

A target appears and disappears instantly, so Copa can hit the target only if at the moment  $t_i$  his gun sight aimed at  $(x_i, y_i)$ . Speed of movement of the gun sight on the plane is equal to 1. Copa knows all the information about the targets beforehand (remember, he is a king!). He wants to play in the optimal way, which maximizes the expected value of the amount of hit targets. He can aim at any target at the moment 0.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> x(n), y(n), t(n);
10    vector<double> p(n);
11    for (int i = 0; i < n; i++) {
12        cin >> x[i] >> y[i] >> t[i] >> p[i];
13    }
14    vector<double> dp(n);
15    vector<int> ord(n);
16    iota(ord.begin(), ord.end(), 0);
17    sort(ord.begin(), ord.end(), [&](int i, int j) {
18        return t[i] < t[j];
19    });
20    double ans = 0;
21    for (int i = 0; i < n; i++) {
22        int u = ord[i];
23        dp[u] = p[u];
24        for (int j = 0; j < i; j++) {
25            int v = ord[j];
26            if (1LL * (x[u] - x[v]) * (x[u] - x[v]) + 1LL * (y[u] - y[v]) * (y[u] - y[v])
27                <= 1LL * (t[u] - t[v]) * (t[u] - t[v])) {
28                dp[u] = max(dp[u], dp[v] + p[u]);
29            }
30        }
31        ans = max(ans, dp[u]);
}

```

```

31     }
32     cout << fixed << setprecision(10);
33     cout << ans << "\n";
34     return 0;
35 }
```

### 435: Segments

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given  $n$  segments on the Ox-axis. You can drive a nail in any integer point on the Ox-axis line nail so, that all segments containing this point, are considered nailed down. If the nail passes through endpoint of some segment, this segment is considered to be nailed too. What is the smallest number of nails needed to nail all the segments down?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<pair<int, int>> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i].first >> a[i].second;
12        if (a[i].first < a[i].second) {
13            swap(a[i].first, a[i].second);
14        }
15    }
16    sort(a.begin(), a.end());
17    int last = -1E9;
18    vector<int> ans;
19    for (auto [r, l] : a) {
20        if (l > last) {
21            last = r;
22            ans.push_back(r);
23        }
24    }
25    cout << ans.size() << "\n";
26    for (auto x : ans) {
27        cout << x << " \n"[x == ans.back()];
28    }
29    return 0;
30 }
```

### 436: Checkout Assistant

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Bob came to a cash & carry store, put  $n$  items into his trolley, and went to the checkout counter to pay. Each item is described by its price  $c_i$  and time  $t_i$  in seconds that a checkout assistant spends on this item. While the checkout assistant is occupied with some item, Bob can steal some other items from his trolley. To steal one item Bob needs exactly 1 second. What is the minimum amount of money that Bob will have to pay to the checkout assistant? Remember, please, that it is Bob, who determines the order of items for the checkout assistant.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<i64> dp(2 * n + 1, 1E18);
10    dp[n] = 0;
11    for (int i = 0; i < n; i++) {
12        int t, c;
13        cin >> t >> c;
14        vector<i64> g(2 * n + 1, 1E18);
15        for (int j = 0; j <= 2 * n; j++) {
16            g[min(2 * n, j + t)] = min(g[min(2 * n, j + t)], dp[j] + c);
17            if (j) {
18                g[j - 1] = min(g[j - 1], dp[j]);
19            }
20        }
21        dp = g;
22    }
23    i64 ans = *min_element(dp.begin() + n, dp.end());
24    cout << ans << "\n";
25    return 0;
26 }
```

### 437: Platforms

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

In one one-dimensional world there are  $n$  platforms. Platform with index  $k$  (platforms are numbered from 1) is a segment with coordinates  $[(k - 1)m, (k - 1)m + l]$ , and  $l < m$ . Grasshopper Bob starts to jump along the platforms from point 0, with each jump he moves exactly  $d$  units right. Find out the coordinate of the point, where Bob will fall down. The grasshopper falls down, if he finds himself not on the platform, but if he finds himself on the edge of the platform, he doesn't fall down.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, d, m, l;
8     cin >> n >> d >> m >> l;
9     i64 cur = 0;
10    while (cur < 1LL * m * n && cur % m <= l) {
11        cur += (l - cur % m) / d * d;
12        cur += d;
13    }
14    cout << cur << "\n";
15    return 0;
16 }
```

## 438: Fish

- Time limit: 3 seconds
- Memory limit: 128 megabytes
- Input file: standard input
- Output file: standard output

$n$  fish, numbered from 1 to  $n$ , live in a lake. Every day right one pair of fish meet, and the probability of each other pair meeting is the same. If two fish with indexes  $i$  and  $j$  meet, the first will eat up the second with the probability  $a_{ij}$ , and the second will eat up the first with the probability  $a_{ji} = 1 - a_{ij}$ . The described process goes on until there are at least two fish in the lake. For each fish find out the probability that it will survive to be the last in the lake.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
```

```

3  using i64 = long long;
4  int main() {
5      ios::sync_with_stdio(false);
6      cin.tie(nullptr);
7      int n;
8      cin >> n;
9      vector a(n, vector<double>(n));
10     for (int i = 0; i < n; i++) {
11         for (int j = 0; j < n; j++) {
12             cin >> a[i][j];
13         }
14     }
15     vector<double> dp(1 << n);
16     dp[(1 << n) - 1] = 1;
17     cout << fixed << setprecision(10);
18     for (int s = (1 << n) - 1; s > 0; s--) {
19         int c = __builtin_popcount(s);
20         for (int i = 0; i < n; i++) {
21             if (s >> i & 1) {
22                 for (int j = i + 1; j < n; j++) {
23                     if (s >> j & 1) {
24                         dp[s ^ (1 << j)] += dp[s] * a[i][j] / (c * (c - 1) / 2);
25                         dp[s ^ (1 << i)] += dp[s] * (1 - a[i][j]) / (c * (c - 1) / 2);
26                     }
27                 }
28             }
29         }
30     }
31     for (int i = 0; i < n; i++) {
32         cout << dp[1 << i] << " \n"[i == n - 1];
33     }
34     return 0;
35 }
```

### 439: Monitor

- Time limit: 0.5 second
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

Reca company makes monitors, the most popular of their models is AB999 with the screen size  $a \times b$  centimeters. Because of some production peculiarities a screen parameters are integer numbers. Recently the screen sides ratio  $x: y$  became popular with users. That's why the company wants to reduce monitor AB999 size so that its screen sides ratio becomes  $x: y$ , at the same time they want its total area to be maximal of all possible variants. Your task is to find the screen parameters of the reduced size model, or find out that such a reduction can't be performed.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int a, b, x, y;
8     cin >> a >> b >> x >> y;
9     int g = gcd(x, y);
10    x /= g, y /= g;
11    int t = min(a / x, b / y);
12    cout << t * x << " " << t * y << "\n";
13    return 0;
14 }

```

#### 440: Laser

- Time limit: 1 second
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

Petya is the most responsible worker in the Research Institute. So he was asked to make a very important experiment: to melt the chocolate bar with a new laser device. The device consists of a rectangular field of  $n \times m$  cells and a robotic arm. Each cell of the field is a  $1 \times 1$  square. The robotic arm has two lasers pointed at the field perpendicularly to its surface. At any one time lasers are pointed at the centres of some two cells. Since the lasers are on the robotic hand, their movements are synchronized - if you move one of the lasers by a vector, another one moves by the same vector.

The following facts about the experiment are known:

initially the whole field is covered with a chocolate bar of the size  $n \times m$ , both lasers are located above the field and are active;

the chocolate melts within one cell of the field at which the laser is pointed;

all moves of the robotic arm should be parallel to the sides of the field, after each move the lasers should be pointed at the centres of some two cells;

at any one time both lasers should be pointed at the field. Petya doesn't want to become a second Gordon Freeman.

You are given  $n$ ,  $m$  and the cells  $(x_1, y_1)$  and  $(x_2, y_2)$ , where the lasers are initially pointed at  $(x_i$  is a column number,  $y_i$  is a row number). Rows are numbered from 1 to  $m$  from top to bottom and columns are numbered from 1 to  $n$  from left to right. You are to find the amount of cells of the field on which the chocolate can't be melted in the given conditions.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m, x1, y1, x2, y2;
6     cin >> n >> m >> x1 >> y1 >> x2 >> y2;
7     int dx = abs(x1 - x2);
8     int dy = abs(y1 - y2);
9     i64 ans = 2LL * (n - dx) * (m - dy);
10    ans -= 1LL * max(0, n - 2 * dx) * max(0, m - 2 * dy);
11    ans = 1LL * n * m - ans;
12    cout << ans << "\n";
13 }
14 int main() {
15     ios::sync_with_stdio(false);
16     cin.tie(nullptr);
17     int t;
18     cin >> t;
19     while (t--) {
20         solve();
21     }
22     return 0;
23 }
```

## 441: Long Legs

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A robot is placed in a cell  $(0, 0)$  of an infinite grid. This robot has adjustable length legs. Initially, its legs have length 1.

Let the robot currently be in the cell  $(x, y)$  and have legs of length  $m$ . In one move, it can perform one of the following three actions:

jump into the cell  $(x + m, y)$ ;

jump into the cell  $(x, y + m)$ ;

increase the length of the legs by 1, i. e. set it to  $m + 1$ .

What's the smallest number of moves robot has to make to reach a cell  $(a, b)$ ?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int a, b;
6     cin >> a >> b;
7     int ans = 2E9;
8     for (int i = 1; i <= 1E5; i++) {
9         ans = min(ans, i + 1 + (a - 1) / i + (b - 1) / i);
10    }
11    cout << ans << "\n";
12 }
13 int main() {
14     ios::sync_with_stdio(false);
15     cin.tie(nullptr);
16     int t;
17     cin >> t;
18     while (t--) {
19         solve();
20     }
21     return 0;
22 }
```

## 442: Climbing the Tree

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The snails are climbing a tree. The tree height is  $h$  meters, and snails start at position 0.

Each snail has two attributes  $a$  and  $b$  ( $a > b$ ). Starting from the 1-st day, one snail climbs the tree like this: during the daylight hours of the day, he climbs up  $a$  meters; during the night, the snail rests, and he slides down  $b$  meters. If on the  $n$ -th day, the snail reaches position  $h$  for the first time (that is, the top of the tree), he will finish climbing, and we say that the snail spends  $n$  days climbing the tree. Note that on the last day of climbing, the snail doesn't necessarily climb up  $a$  meters, in case his distance to the top is smaller than  $a$ .

Unfortunately, you don't know the exact tree height  $h$  at first, but you know that  $h$  is a positive integer. There are  $q$  events of two kinds.

Event of type 1: a snail with attributes  $a, b$  comes and claims that he spent  $n$  days climbing the tree. If this message contradicts previously adopted information (i. e. there is no tree for which all previously adopted statements and this one are true), ignore it. Otherwise, adopt it.

Event of type 2: a snail with attributes  $a, b$  comes and asks you how many days he will spend if he climbs the tree. You can only give the answer based on the information you have adopted so far. If you cannot determine the answer precisely, report that.

You need to deal with all the events in order.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int q;
6     cin >> q;
7     i64 lo = 1, hi = 1E18;
8     while (q--) {
9         int t;
10        cin >> t;
11        if (t == 1) {
12            int a, b, n;
13            cin >> a >> b >> n;
14            i64 x, y;
15            if (n == 1) {
16                x = 1;
17                y = a;
18            } else {
19                x = 1LL * a * (n - 1) - 1LL * b * (n - 2) + 1;
20                y = 1LL * a * n - 1LL * b * (n - 1);
21            }
22            if (x <= hi && lo <= y) {
23                lo = max(lo, x);
24                hi = min(hi, y);
25                cout << 1 << " ";
26            } else {
27                cout << 0 << " ";
28            }
29        } else {
30            int a, b;
31            cin >> a >> b;
32            i64 t1 = lo <= a ? 1LL : (lo - b + a - b - 1) / (a - b);
33            i64 t2 = hi <= a ? 1LL : (hi - b + a - b - 1) / (a - b);
34            if (t1 == t2) {
35                cout << t1 << " ";
36            } else {
37                cout << -1 << " ";
38            }
39        }
40    }
41    cout << "\n";
42 }
43 int main() {
44     ios::sync_with_stdio(false);
45     cin.tie(nullptr);
46     int t;
47     cin >> t;
48     while (t--) {
49         solve();
50     }
51     return 0;
52 }
```

### 443: Unlucky Numbers

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In this problem, unlike problem A, you need to look for unluckiest number, not the luckiest one.

Note that the constraints of this problem differ from such in problem A.

Olympus City recently launched the production of personal starships. Now everyone on Mars can buy one and fly to other planets inexpensively.

Each starship has a number -some positive integer  $x$ . Let's define the luckiness of a number  $x$  as the difference between the largest and smallest digits of that number. For example, 142857 has 8 as its largest digit and 1 as its smallest digit, so its luckiness is  $8 - 1 = 7$ . And the number 111 has all digits equal to 1, so its luckiness is zero.

Hateehc is a famous Martian blogger who often flies to different corners of the solar system. To release interesting videos even faster, he decided to buy himself a starship. When he came to the store, he saw starships with numbers from  $l$  to  $r$  inclusively. While in the store, Hateehc wanted to find a starship with the unluckiest number.

Since there are a lot of starships in the store, and Hateehc can't program, you have to help the blogger and write a program that answers his question.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int luckiness(i64 x) {
5     auto s = to_string(x);
6     auto [pmin, pmax] = minmax_element(s.begin(), s.end());
7     return *pmax - *pmin;
8 }
9 void solve() {
10     i64 l, r;
11     cin >> l >> r;
12     i64 ans = l;
13     i64 p = 1;
14     for (int i = 0; i <= 18; i++) {
15         for (int x = 0; x < 10; x++) {
16             i64 v = l / p * p + (p - 1) / 9 * x;
17             if (l <= v && v <= r && luckiness(v) < luckiness(ans)) {
18                 ans = v;
19             }
20             v = r / p * p + (p - 1) / 9 * x;
21             if (l <= v && v <= r && luckiness(v) < luckiness(ans)) {

```

```

22             ans = v;
23         }
24     }
25     p *= 10;
26 }
27 cout << ans << "\n";
28 }
29 int main() {
30     ios::sync_with_stdio(false);
31     cin.tie(nullptr);
32     int t;
33     cin >> t;
34     while (t--) {
35         solve();
36     }
37     return 0;
38 }
```

#### 444: Candy Store

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The store sells  $n$  types of candies with numbers from 1 to  $n$ . One candy of type  $i$  costs  $b_i$  coins. In total, there are  $a_i$  candies of type  $i$  in the store.

You need to pack all available candies in packs, each pack should contain only one type of candies. Formally, for each type of candy  $i$  you need to choose the integer  $d_i$ , denoting the number of type  $i$  candies in one pack, so that  $a_i$  is divided without remainder by  $d_i$ .

Then the cost of one pack of candies of type  $i$  will be equal to  $b_i \cdot d_i$ . Let's denote this cost by  $c_i$ , that is,  $c_i = b_i \cdot d_i$ .

After packaging, packs will be placed on the shelf. Consider the cost of the packs placed on the shelf, in order  $c_1, c_2, \dots, c_n$ . Price tags will be used to describe costs of the packs. One price tag can describe the cost of all packs from  $l$  to  $r$  inclusive if  $c_l = c_{l+1} = \dots = c_r$ . Each of the packs from 1 to  $n$  must be described by at least one price tag. For example, if  $c_1, \dots, c_n = [4, 4, 2, 4, 4]$ , to describe all the packs, a 3 price tags will be enough, the first price tag describes the packs 1, 2, the second: 3, the third: 4, 5.

You are given the integers  $a_1, b_1, a_2, b_2, \dots, a_n, b_n$ . Your task is to choose integers  $d_i$  so that  $a_i$  is divisible by  $d_i$  for all  $i$ , and the required number of price tags to describe the values of  $c_1, c_2, \dots, c_n$  is the minimum possible.

For a better understanding of the statement, look at the illustration of the first test case of the first test:

Let's repeat the meaning of the notation used in the problem:

$a_i$  - the number of candies of type  $i$  available in the store.

$b_i$  - the cost of one candy of type  $i$ .

$d_i$  - the number of candies of type  $i$  in one pack.

$c_i$  - the cost of one pack of candies of type  $i$  is expressed by the formula  $c_i = b_i \cdot d_i$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n), b(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i] >> b[i];
10    }
11    i64 g = 0, m = 1;
12    int ans = 1;
13    for (int i = 0; i < n; i++) {
14        if (gcd(g, 1LL * a[i] * b[i]) % lcm(m, 1LL * b[i]) != 0) {
15            ans += 1;
16            g = 0;
17            m = 1;
18        }
19        g = gcd(g, 1LL * a[i] * b[i]);
20        m = lcm(m, 1LL * b[i]);
21    }
22    cout << ans << "\n";
23 }
24 int main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
27     int t;
28     cin >> t;
29     while (t--) {
30         solve();
31     }
32     return 0;
33 }
```

## violet

### brute force

#### 445: Journey

- Time limit: 2 seconds

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The territory of Berland is represented by a rectangular field  $n \times m$  in size. The king of Berland lives in the capital, located on the upper left square  $(1, 1)$ . The lower right square has coordinates  $(n, m)$ . One day the king decided to travel through the whole country and return back to the capital, having visited every square (except the capital) exactly one time. The king must visit the capital exactly two times, at the very beginning and at the very end of his journey. The king can only move to the side-neighboring squares. However, the royal advise said that the King possibly will not be able to do it. But there is a way out - one can build the system of one way teleporters between some squares so that the king could fulfill his plan. No more than one teleporter can be installed on one square, every teleporter can be used any number of times, however every time it is used, it transports to the same given for any single teleporter square. When the king reaches a square with an installed teleporter he chooses himself whether he is or is not going to use the teleport. What minimum number of teleporters should be installed for the king to complete the journey? You should also compose the journey path route for the king.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     if (n % 2 == 0 && (m > 1 || n == 2)) {
10         cout << 0 << "\n";
11         for (int i = 1; i <= n; i++) {
12             cout << i << " " << 1 << "\n";
13         }
14         for (int i = n; i; i--) {
15             if (i % 2) {
16                 for (int j = m; j > 1; j--) {
17                     cout << i << " " << j << "\n";
18                 }
19             } else {
20                 for (int j = 2; j <= m; j++) {
21                     cout << i << " " << j << "\n";
22                 }
23             }
24         }
25         cout << 1 << " " << 1 << "\n";
26     } else if (m % 2 == 0 && (n > 1 || m == 2)) {
27         cout << 0 << "\n";
28         for (int i = 1; i <= m; i++) {
29             cout << 1 << " " << i << "\n";

```

```

30
31     }
32     for (int i = m; i; i--) {
33         if (i % 2) {
34             for (int j = n; j > 1; j--) {
35                 cout << j << " " << i << "\n";
36             }
37         } else {
38             for (int j = 2; j <= n; j++) {
39                 cout << j << " " << i << "\n";
40             }
41         }
42         cout << 1 << " " << 1 << "\n";
43     } else {
44         cout << 1 << "\n";
45         cout << n << " " << m << " " << 1 << " " << 1 << "\n";
46         for (int i = 1; i <= n; i++) {
47             if (i % 2) {
48                 for (int j = 1; j <= m; j++) {
49                     cout << i << " " << j << "\n";
50                 }
51             } else {
52                 for (int j = m; j; j--) {
53                     cout << i << " " << j << "\n";
54                 }
55             }
56         }
57         cout << 1 << " " << 1 << "\n";
58     }
59     return 0;
60 }
```

## 446: Chocolate Bar

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have a rectangular chocolate bar consisting of  $n \times m$  single squares. You want to eat exactly  $k$  squares, so you may need to break the chocolate bar.

In one move you can break any single rectangular piece of chocolate in two rectangular pieces. You can break only by lines between squares: horizontally or vertically. The cost of breaking is equal to square of the break length.

For example, if you have a chocolate bar consisting of  $2 \times 3$  unit squares then you can break it horizontally and get two  $1 \times 3$  pieces (the cost of such breaking is  $3^2 = 9$ ), or you can break it vertically in two ways and get two pieces:  $2 \times 1$  and  $2 \times 2$  (the cost of such breaking is  $2^2 = 4$ ).

For several given values  $n$ ,  $m$  and  $k$  find the minimum total cost of breaking. You can eat exactly  $k$  squares of chocolate if after all operations of breaking there is a set of rectangular pieces of chocolate

with the total size equal to  $k$  squares. The remaining  $n \cdot m - k$  squares are not necessarily form a single rectangular piece.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int N = 30, K = 50;
5 int dp[N+1][N+1][K+1];
6 void solve() {
7     int n, m, k;
8     cin >> n >> m >> k;
9     cout << dp[n][m][k] << "\n";
10 }
11 int main() {
12     ios::sync_with_stdio(false);
13     cin.tie(nullptr);
14     memset(dp, 0x3f, sizeof(dp));
15     for (int n = 1; n <= N; n++) {
16         for (int m = 1; m <= N; m++) {
17             dp[n][m][0] = 0;
18             if (n*m <= K) {
19                 dp[n][m][n*m] = 0;
20             }
21             for (int x = 1; x < n; x++) {
22                 for (int a = 0; a <= K; a++) {
23                     for (int b = 0; a+b <= K; b++) {
24                         dp[n][m][a+b] = min(dp[n][m][a+b], dp[x][m][a] + dp[n-x][m][b] + m
25                                     *m);
26                     }
27                 }
28                 for (int x = 1; x < m; x++) {
29                     for (int a = 0; a <= K; a++) {
30                         for (int b = 0; a+b <= K; b++) {
31                             dp[n][m][a+b] = min(dp[n][m][a+b], dp[n][x][a] + dp[n][m-x][b] + n
32                                     *n);
33                         }
34                     }
35                 }
36             }
37             int t;
38             cin >> t;
39             while (t--) {
40                 solve();
41             }
42             return 0;
43     }
}

```

#### 447: Number With The Given Amount Of Divisors

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

Given the number  $n$ , find the smallest positive integer which has exactly  $n$  divisors. It is guaranteed that for the given  $n$  the answer will not exceed 1018.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 const int P[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43};
5 i64 ans = 1E18;
6 void dfs(int n, int i, int last, i64 x) {
7     if (n == 1) {
8         ans = min(ans, x);
9         return;
10    }
11    for (int e = 2; e <= n && e <= last; e++) {
12        if (x > 1E18 / P[i]) {
13            break;
14        }
15        x *= P[i];
16        if (n % e == 0) {
17            dfs(n / e, i + 1, e, x);
18        }
19    }
20 }
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     int n;
25     cin >> n;
26     dfs(n, 0, 1E9, 1);
27     cout << ans << "\n";
28     return 0;
29 }
```

## 448: Sum Graph

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an interactive problem.

There is a hidden permutation  $p_1, p_2, \dots, p_n$ .

Consider an undirected graph with  $n$  nodes only with no edges. You can make two types of queries:

Specify an integer  $x$  satisfying  $2 \leq x \leq 2n$ . For all integers  $i$  ( $1 \leq i \leq n$ ) such that  $1 \leq x - i \leq n$ , an edge between node  $i$  and node  $x - i$  will be added.

Query the number of edges in the shortest path between node  $p_i$  and node  $p_j$ . As the answer to this question you will get the number of edges in the shortest path if such a path exists, or  $-1$  if there is no such path.

Note that you can make both types of queries in any order.

Within  $2n$  queries (including type 1 and type 2), guess two possible permutations, at least one of which is  $p_1, p_2, \dots, p_n$ . You get accepted if at least one of the permutations is correct. You are allowed to guess the same permutation twice.

A permutation of length  $n$  is an array consisting of  $n$  distinct integers from 1 to  $n$  in arbitrary order. For example,  $[2, 3, 1, 5, 4]$  is a permutation, but  $[1, 2, 2]$  is not a permutation (2 appears twice in the array), and  $[1, 3, 4]$  is also not a permutation ( $n = 3$  but there is 4 in the array).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void add(int x) {
5     cout << "+" << x << endl;
6     int res;
7     cin >> res;
8 }
9 int query(int i, int j) {
10    cout << "?" << i << " " << j << endl;
11    int res;
12    cin >> res;
13    return res;
14 }
15 void solve() {
16     int n;
17     cin >> n;
18     add(n + 1);
19     add(n);
20     vector<int> a(n), b(n), c(n);
21     for (int i = 1; i < n; i++) {
22         a[i] = query(1, i + 1);
23     }
24     int j = find(a.begin(), a.end(), 1) - a.begin();
25     b[0] = a[j];
26     for (int i = 1; i < n; i++) {
27         b[i] = query(j + 1, i + 1);
28     }
29     for (int i = 0; i < n; i++) {
30         if (a[i] < b[i]) {
31             c[i] = -a[i];
32         } else {
33             c[i] = a[i];
34         }
35     }
36     int min = *min_element(c.begin(), c.end());

```

```

37     for (int i = 0; i < n; i++) {
38         c[i] -= min;
39     }
40     vector<int> p;
41     int l = 1, r = n, t = 0;
42     while (l <= r) {
43         if (!t) {
44             p.push_back(r--);
45         } else {
46             p.push_back(l++);
47         }
48         t ^= 1;
49     }
50     cout << "!";
51     for (int i = 0; i < n; i++) {
52         cout << " " << p[c[i]];
53     }
54     for (int i = 0; i < n; i++) {
55         cout << " " << p[n - 1 - c[i]];
56     }
57     cout << endl;
58     int res;
59     cin >> res;
60 }
61 int main() {
62     ios::sync_with_stdio(false);
63     cin.tie(nullptr);
64     int t;
65     cin >> t;
66     while (t--) {
67         solve();
68     }
69     return 0;
70 }
```

#### 449: Different Arrays

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  consisting of  $n$  integers.

You have to perform the sequence of  $n - 2$  operations on this array:

during the first operation, you either add  $a_2$  to  $a_1$  and subtract  $a_2$  from  $a_3$ , or add  $a_2$  to  $a_3$  and subtract  $a_2$  from  $a_1$ ;

during the second operation, you either add  $a_3$  to  $a_2$  and subtract  $a_3$  from  $a_4$ , or add  $a_3$  to  $a_4$  and subtract  $a_3$  from  $a_2$ ;

...

during the last operation, you either add  $a_{n-1}$  to  $a_{n-2}$  and subtract  $a_{n-1}$  from  $a_n$ , or add  $a_{n-1}$  to  $a_n$  and subtract  $a_{n-1}$  from  $a_{n-2}$ .

So, during the  $i$ -th operation, you add the value of  $a_{i+1}$  to one of its neighbors, and subtract it from the other neighbor.

For example, if you have the array  $[1, 2, 3, 4, 5]$ , one of the possible sequences of operations is:

subtract 2 from  $a_3$  and add it to  $a_1$ , so the array becomes  $[3, 2, 1, 4, 5]$ ;

subtract 1 from  $a_2$  and add it to  $a_4$ , so the array becomes  $[3, 1, 1, 5, 5]$ ;

subtract 5 from  $a_3$  and add it to  $a_5$ , so the array becomes  $[3, 1, -4, 5, 10]$ .

So, the resulting array is  $[3, 1, -4, 5, 10]$ .

An array is reachable if it can be obtained by performing the aforementioned sequence of operations on  $a$ . You have to calculate the number of reachable arrays, and print it modulo 998244353.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 998244353;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
8     if (x < 0) {
9         x += P;
10    }
11    if (x >= P) {
12        x -= P;
13    }
14    return x;
15 }
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25 }
26 # struct Z;
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(nullptr);
30     int n;
31     cin >> n;
32     vector<int> a(n);
33     for (int i = 0; i < n; i++) {
34         cin >> a[i];

```

```

35     }
36     int sum = accumulate(a.begin(), a.end(), 0);
37     vector<Z> dp(2 * sum + 1);
38     dp[a[1] + sum] = 1;
39     for (int i = 2; i < n; i++) {
40         vector<Z> g(2 * sum + 1);
41         for (int j = 0; j <= 2 * sum; j++) {
42             if (!dp[j].val()) continue;
43             g[sum + a[i] - (j - sum)] += dp[j];
44             if (j != sum) g[sum + a[i] + (j - sum)] += dp[j];
45         }
46         swap(dp, g);
47     }
48     auto ans = accumulate(dp.begin(), dp.end(), Z(0));
49     cout << ans << "\n";
50     return 0;
51 }
```

**450: Sheikh (Hard Version)**

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. The only difference is that in this version  $q = n$ .

You are given an array of integers  $a_1, a_2, \dots, a_n$ .

The cost of a subsegment of the array  $[l, r]$ ,  $1 \leq l \leq r \leq n$ , is the value  $f(l, r) = \text{sum}(l, r) - \text{xor}(l, r)$ , where  $\text{sum}(l, r) = a_l + a_{l+1} + \dots + a_r$ , and  $\text{xor}(l, r) = a_l \oplus a_{l+1} \oplus \dots \oplus a_r$  ( $\oplus$  stands for bitwise XOR).

You will have  $q$  queries. Each query is given by a pair of numbers  $L_i, R_i$ , where  $1 \leq L_i \leq R_i \leq n$ . You need to find the subsegment  $[l, r]$ ,  $L_i \leq l \leq r \leq R_i$ , with maximum value  $f(l, r)$ . If there are several answers, then among them you need to find a subsegment with the minimum length, that is, the minimum value of  $r - l + 1$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, q;
6     cin >> n >> q;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<int> p;
```

```

12     for (int i = 0; i < n; i++) {
13         if (a[i]) {
14             p.push_back(i);
15         }
16     }
17     const int m = p.size();
18     vector<i64> sum(m + 1);
19     vector<int> xsum(m + 1);
20     for (int i = 0; i < m; i++) {
21         sum[i + 1] = sum[i] + a[p[i]];
22         xsum[i + 1] = xsum[i] ^ a[p[i]];
23     }
24     for (int i = 0; i < q; i++) {
25         int L, R;
26         cin >> L >> R;
27         L--;
28         int u = L, v = R;
29         int l = lower_bound(p.begin(), p.end(), L) - p.begin();
30         int r = lower_bound(p.begin(), p.end(), R) - p.begin();
31         i64 val = sum[r] - sum[l] - (xsum[r] ^ xsum[l]);
32         if (!val) {
33             v = u + 1;
34         }
35         for (int x = l; x <= min(r, l + 31); x++) {
36             for (int y = max(x + 1, r - 31); y <= r; y++) {
37                 i64 cur = sum[y] - sum[x] - (xsum[y] ^ xsum[x]);
38                 if (cur == val) {
39                     int l = p[x];
40                     int r = p[y - 1] + 1;
41                     if (r - l < v - u) {
42                         u = l;
43                         v = r;
44                     }
45                 }
46             }
47         }
48         cout << u + 1 << " " << v << "\n";
49     }
50 }
51 int main() {
52     ios::sync_with_stdio(false);
53     cin.tie(nullptr);
54     int t;
55     cin >> t;
56     while (t--) {
57         solve();
58     }
59     return 0;
60 }
```

## data structures

### 451: Maximize The Value

- Time limit: 1 second
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

You are given a one-based array consisting of  $N$  integers:  $A_1, A_2, \dots, A_N$ . Initially, the value of each element is set to 0.

There are  $M$  operations (numbered from 1 to  $M$ ). Operation  $i$  is represented by  $\langle L_i, R_i, X_i \rangle$ . If operation  $i$  is executed, all elements  $A_j$  for  $L_i \leq j \leq R_i$  will be increased by  $X_i$ .

You have to answer  $Q$  independent queries. Each query is represented by  $\langle K, S, T \rangle$  which represents the following task. Choose a range  $[l, r]$  satisfying  $S \leq l \leq r \leq T$ , and execute operations  $l, l+1, \dots, r$ . The answer to the query is the maximum value of  $A_K$  after the operations are executed among all possible choices of  $l$  and  $r$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class Info>
5 # struct SegmentTree;
6 constexpr i64 inf = 1E18;
7 # struct Info;
8 Info operator+(const Info &a, const Info &b) {
9     Info c;
10    c.sum = a.sum + b.sum;
11    c.ans = max({a.ans, b.ans, a.suf + b.pre});
12    c.pre = max(a.pre, a.sum + b.pre);
13    c.suf = max(a.suf + b.sum, b.suf);
14    return c;
15 }
16 int main() {
17     ios::sync_with_stdio(false);
18     cin.tie(nullptr);
19     int N, M;
20     cin >> N >> M;
21     vector<vector<pair<int, int>>> modify(N);
22     for (int i = 0; i < M; i++) {
23         int L, R, X;
24         cin >> L >> R >> X;
25         L--;
26         modify[L].emplace_back(i, X);
27         if (R < N) {
28             modify[R].emplace_back(i, 0);
29         }
30     }
31     int Q;
32     cin >> Q;
33     vector<i64> ans(Q);
34     vector<vector<array<int, 3>>> query(N);
35     for (int i = 0; i < Q; i++) {
36         int K, S, T;
37         cin >> K >> S >> T;
38         K--, S--;
39         query[K].push_back({i, S, T});
40     }
41     SegmentTree<Info> seg(vector(M, Info{0, 0, 0, 0, 0}));
42     for (int i = 0; i < N; i++) {
43         for (auto [j, x] : modify[i]) {
44             seg.modify(j, {x, x, x, x});
45         }
46     }

```

```

46         for (auto [j, l, r] : query[i]) {
47             ans[j] = seg.rangeQuery(l, r).ans;
48         }
49     }
50     for (int i = 0; i < Q; i++) {
51         cout << ans[i] << "\n";
52     }
53     return 0;
54 }
```

**452: A Growing Tree**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a rooted tree with the root at vertex 1, initially consisting of a single vertex. Each vertex has a numerical value, initially set to 0. There are also  $q$  queries of two types:

The first type: add a child vertex with the number  $sz + 1$  to vertex  $v$ , where  $sz$  is the current size of the tree. The numerical value of the new vertex will be 0.

The second type: add  $x$  to the numerical values of all vertices in the subtree of vertex  $v$ .

After all queries, output the numerical value of all of the vertices in the final tree.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct HLD;
5 template <typename T>
6 # struct Fenwick;
7 void solve() {
8     int q;
9     cin >> q;
10    vector<array<int, 3>> qry(q);
11    vector<int> p{-1};
12    for (int i = 0; i < q; i++) {
13        int o;
14        cin >> o;
15        if (o == 1) {
16            int x;
17            cin >> x;
18            x--;
19            p.push_back(x);
20            qry[i] = {1, int(p.size()) - 1, x};
21        } else {
22            int x, y;
23            cin >> x >> y;
24            x--;
```

```

25         qry[i] = {2, x, y};
26     }
27 }
28 int n = p.size();
29 HLD t(p.size());
30 for (int i = 1; i < p.size(); i++) {
31     t.addEdge(p[i], i);
32 }
33 t.work();
34 Fenwick<i64> fen(n);
35 for (auto [o, x, y] : qry) {
36     if (o == 1) {
37         i64 v = fen.sum(t.in[x] + 1);
38         fen.add(t.in[x], -v);
39         fen.add(t.out[x], v);
40     } else {
41         fen.add(t.in[x], y);
42         fen.add(t.out[x], -y);
43     }
44 }
45 for (int i = 0; i < n; i++) {
46     cout << fen.sum(t.in[i] + 1) << " \n"[i == n - 1];
47 }
48 }
49 int main() {
50     ios::sync_with_stdio(false);
51     cin.tie(nullptr);
52     int t;
53     cin >> t;
54     while (t--) {
55         solve();
56     }
57     return 0;
58 }
```

### 453: Anya and the Mysterious String

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Anya received a string  $s$  of length  $n$  brought from Rome. The string  $s$  consists of lowercase Latin letters and at first glance does not raise any suspicions. An instruction was attached to the string.

Start of the instruction.

A palindrome is a string that reads the same from left to right and right to left. For example, the strings “anna”, “aboba”, “level” are palindromes, while the strings “gorilla”, “banan”, “off” are not.

A substring  $[l \dots r]$  of string  $s$  is a string  $s_l s_{l+1} \dots s_{r-1} s_r$ . For example, the substring  $[4 \dots 6]$  of the string “generation” is the string “era”.

A string is called beautiful if it does not contain a substring of length at least two that is a palindrome.

For example, the strings “fox”, “abcdef”, and “yioy” are beautiful, while the strings “xyxx”, “yikjkitrb” are not.

When an integer  $x$  is added to the character  $s_i$ , it is replaced  $x$  times with the next character in the alphabet, with “z” being replaced by “a”.

When an integer  $x$  is added to the substring  $[l, r]$  of string  $s$ , it becomes the string  $s_1 s_2 \dots s_{l-1} (s_l + x) (s_{l+1} + x) \dots (s_{r-1} + x) (s_r + x) s_{r+1} \dots s_n$ . For example, when the substring  $[2, 4]$  of the string “abazaba” is added with the number 6, the resulting string is “ahgfaba”.

End of the instruction.

After reading the instruction, Anya resigned herself to the fact that she has to answer  $m$  queries. The queries can be of two types:

Add the number  $x$  to the substring  $[l \dots r]$  of string  $s$ .

Determine whether the substring  $[l \dots r]$  of string  $s$  is beautiful.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class Info, class Tag>
5 # struct LazySegmentTree;
6 # struct Tag;
7 # struct Info;
8 Info operator+(const Info &a, const Info &b) {
9     Info c;
10    c.ok = a.ok && b.ok;
11    if (a.r[0] == b.l[0] && a.r[0] != -1) {
12        c.ok = false;
13    }
14    if (a.r[0] == b.l[1] && a.r[0] != -1) {
15        c.ok = false;
16    }
17    if (a.r[1] == b.l[0] && a.r[1] != -1) {
18        c.ok = false;
19    }
20    if (a.l[0] == -1) {
21        c.l = b.l;
22    } else if (a.l[1] == -1) {
23        c.l[0] = a.l[0];
24        c.l[1] = b.l[0];
25    } else {
26        c.l = a.l;
27    }
28    if (b.r[0] == -1) {
29        c.r = a.r;
30    } else if (b.r[1] == -1) {
31        c.r[0] = b.r[0];
32        c.r[1] = a.r[0];
33    } else {

```

```

34         c.r = b.r;
35     }
36     return c;
37 }
38 void solve() {
39     int n, m;
40     cin >> n >> m;
41     string s;
42     cin >> s;
43     vector<Info> init(n);
44     for (int i = 0; i < n; i++) {
45         init[i].l[0] = init[i].r[0] = s[i] - 'a';
46     }
47     LazySegmentTree<Info, Tag> seg(init);
48     while (m--) {
49         int o, l, r;
50         cin >> o >> l >> r;
51         l--;
52         if (o == 1) {
53             int x;
54             cin >> x;
55             seg.rangeApply(l, r, {x});
56         } else {
57             cout << (seg.rangeQuery(l, r).ok ? "YES" : "NO") << "\n";
58         }
59     }
60 }
61 int main() {
62     ios::sync_with_stdio(false);
63     cin.tie(nullptr);
64     int t;
65     cin >> t;
66     while (t--) {
67         solve();
68     }
69     return 0;
70 }

```

#### 454: Vlad and the Mountains

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vlad decided to go on a trip to the mountains. He plans to move between  $n$  mountains, some of which are connected by roads. The  $i$ -th mountain has a height of  $h_i$ .

If there is a road between mountains  $i$  and  $j$ , Vlad can move from mountain  $i$  to mountain  $j$  by spending  $h_j - h_i$  units of energy. If his energy drops below zero during the transition, he will not be able to move from mountain  $i$  to mountain  $j$ . Note that  $h_j - h_i$  can be negative and then the energy will be restored.

Vlad wants to consider different route options, so he asks you to answer the following queries: is

it possible to construct some route starting at mountain  $a$  and ending at mountain  $b$ , given that he initially has  $e$  units of energy?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 void solve() {
6     int n, m;
7     cin >> n >> m;
8     vector<int> h(n);
9     for (int i = 0; i < n; i++) {
10         cin >> h[i];
11     }
12     vector<array<int, 3>> edges(m);
13     for (int i = 0; i < m; i++) {
14         int u, v;
15         cin >> u >> v;
16         u--, v--;
17         edges[i] = {max(h[u], h[v]), u, v};
18     }
19     sort(edges.begin(), edges.end());
20     int q;
21     cin >> q;
22     vector<array<int, 4>> qry(q);
23     for (int i = 0; i < q; i++) {
24         int a, b, e;
25         cin >> a >> b >> e;
26         a--, b--;
27         qry[i] = {h[a] + e, a, b, i};
28     }
29     sort(qry.begin(), qry.end());
30     DSU dsu(n);
31     int j = 0;
32     vector<bool> ans(q);
33     for (auto [w, u, v, i] : qry) {
34         while (j < m && edges[j][0] <= w) {
35             dsu.merge(edges[j][1], edges[j][2]);
36             j++;
37         }
38         ans[i] = dsu.same(u, v);
39     }
40     for (int i = 0; i < q; i++) {
41         if (ans[i]) {
42             cout << "YES\n";
43         } else {
44             cout << "NO\n";
45         }
46     }
47     cout << "\n";
48 }
49 int main() {
50     ios::sync_with_stdio(false);
51     cin.tie(nullptr);
52     int t;
53     cin >> t;
54     while (t--) {

```

```

55     solve();
56 }
57 return 0;
58 }
```

### 455: Minimum spanning tree for each edge

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Connected undirected weighted graph without self-loops and multiple edges is given. Graph contains n vertices and m edges.

For each edge  $(u, v)$  find the minimal possible weight of the spanning tree that contains the edge  $(u, v)$ .

The weight of the spanning tree is the sum of weights of all edges included in spanning tree.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 # struct HLD;
6 int main() {
7     ios::sync_with_stdio(false);
8     cin.tie(nullptr);
9     int n, m;
10    cin >> n >> m;
11    vector<array<int, 4>> e(m);
12    for (int i = 0; i < m; i++) {
13        int u, v, w;
14        cin >> u >> v >> w;
15        u--;
16        v--;
17        e[i] = {w, u, v, i};
18    }
19    sort(e.begin(), e.end());
20    i64 sum = 0;
21    DSU dsu(2*n-1);
22    int cnt = n;
23    vector<i64> ans(m);
24    HLD t(2*n-1);
25    vector<int> wt(2*n-1);
26    for (auto [w, u, v, i] : e) {
27        if (!dsu.same(u, v)) {
28            t.addEdge(cnt, dsu.find(u));
29            t.addEdge(cnt, dsu.find(v));
30            dsu.merge(cnt, u);
31            dsu.merge(cnt, v);
32            wt[cnt] = w;
33        }
34    }
35    cout << sum << endl;
36 }
```

```

32         cnt++;
33         sum += w;
34     }
35 }
36 t.work(2*n-2);
37 for (auto [w, u, v, i] : e) {
38     ans[i] = sum - wt[t.lca(u, v)] + w;
39 }
40 for (int i = 0; i < m; i++) {
41     cout << ans[i] << "\n";
42 }
43 return 0;
44 }
```

**456: Parade**

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: input.txt
- Output file: output.txt

No Great Victory anniversary in Berland has ever passed without the war parade. This year is not an exception. Thats why the preparations are on in full strength. Tanks are building a line, artillery mounts are ready to fire, soldiers are marching on the main square... And the air forces general Mr. Generalov is in trouble again. This year a lot of sky-scrapers have been built which makes it difficult for the airplanes to fly above the city. It was decided that the planes should fly strictly from south to north. Moreover, there must be no sky scraper on a planes route, otherwise the anniversary will become a tragedy. The Ministry of Building gave the data on  $n$  sky scrapers (the rest of the buildings are rather small and will not be a problem to the planes). When looking at the city from south to north as a geometrical plane, the  $i$ -th building is a rectangle of height  $h_i$ . Its westernmost point has the x-coordinate of  $l_i$  and the easternmost - of  $r_i$ . The terrain of the area is plain so that all the buildings stand on one level. Your task as the Ministry of Defences head programmer is to find an enveloping polyline using the data on the sky-scrapers. The polylines properties are as follows:

If you look at the city from south to north as a plane, then any part of any building will be inside or on the boarder of the area that the polyline encloses together with the land surface.

The polyline starts and ends on the land level, i.e. at the height equal to 0.

The segments of the polyline are parallel to the coordinate axes, i.e. they can only be vertical or horizontal.

The polylines vertices should have integer coordinates.

If you look at the city from south to north the polyline (together with the land surface) must enclose the minimum possible area.

The polyline must have the smallest length among all the polylines, enclosing the minimum possible area with the land.

The consecutive segments of the polyline must be perpendicular.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 #ifdef ONLINE_JUDGE
5     ifstream fin("input.txt");
6     ofstream fout("output.txt");
7 #else
8     #define fin cin
9     #define fout cout
10 #endif
11 int main() {
12     int n;
13     fin >> n;
14     vector<array<int, 3>> a;
15     for (int i = 0; i < n; i++) {
16         int h, l, r;
17         fin >> h >> l >> r;
18         a.push_back({l, 1, h});
19         a.push_back({r, -1, h});
20     }
21     sort(a.begin(), a.end());
22     multiset<int> s{0};
23     vector<pair<int, int>> ans;
24     for (int i = 0, j = 0; i < 2 * n; i = j) {
25         while (j < 2 * n && a[i][0] == a[j][0]) {
26             if (a[j][1] == 1) {
27                 s.insert(a[j][2]);
28             } else {
29                 s.extract(a[j][2]);
30             }
31             j++;
32         }
33         int H = *s.rbegin();
34         if (ans.empty()) {
35             ans.emplace_back(a[i][0], 0);
36             ans.emplace_back(a[i][0], H);
37         } else {
38             if (ans.back().second == ans.end()[-2].second) {
39                 ans.back().first = a[i][0];
40             } else {
41                 ans.emplace_back(a[i][0], ans.back().second);
42             }
43             if (ans.back().second != H) {
44                 ans.emplace_back(a[i][0], H);
45             }
46         }
47     }
48     fout << ans.size() << "\n";
49     for (auto [x, y] : ans) {
50         fout << x << " " << y << "\n";
51     }
52     return 0;
53 }
```

## 457: Sereja and Brackets

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Sereja has a bracket sequence  $s_1, s_2, \dots, s_n$ , or, in other words, a string  $s$  of length  $n$ , consisting of characters "(" and ")".

Sereja needs to answer  $m$  queries, each of them is described by two integers  $l_i, r_i$  ( $1 \leq l_i \leq r_i \leq n$ ). The answer to the  $i$ -th query is the length of the maximum correct bracket subsequence of sequence  $s_{l_i}, s_{l_i+1}, \dots, s_{r_i}$ . Help Sereja answer all queries.

You can find the definitions for a subsequence and a correct bracket sequence in the notes.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T,
5         class Cmp = less<T>>
6 # struct RMQ;
7 int main() {
8     ios::sync_with_stdio(false);
9     cin.tie(nullptr);
10    string s;
11    cin >> s;
12    int n = s.size();
13    vector<int> a(n + 1);
14    for (int i = 0; i < n; i++) {
15        a[i + 1] = a[i] + (s[i] == '(' ? 1 : -1);
16    }
17    RMQ rmq(a);
18    int m;
19    cin >> m;
20    while (m--) {
21        int l, r;
22        cin >> l >> r;
23        l--;
24        int t = rmq(l, r + 1);
25        int ans = r - l - (a[l] - t) - (a[r] - t);
26        cout << ans << "\n";
27    }
28    return 0;
29 }
```

**458: Petr**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Long ago, when Petya was a schoolboy, he was very much interested in the Petr# language grammar. During one lesson Petya got interested in the following question: how many different continuous substrings starting with the sbegin and ending with the send (it is possible sbegin = send), the given string t has. Substrings are different if and only if their contents aren't equal, their positions of occurrence don't matter. Petya wasn't quite good at math, that's why he couldn't count this number. Help him!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     string a, b, c;
8     cin >> a >> b >> c;
9     int na = a.size();
10    int nb = b.size();
11    int nc = c.size();
12    if (na < nb || na < nc) {
13        cout << 0 << "\n";
14        return 0;
15    }
16    vector<int> p, q;
17    for (int i = 0; i + nb <= na; i++) {
18        if (a.substr(i, nb) == b) {
19            p.push_back(i);
20        }
21    }
22    for (int i = 0; i + nc <= na; i++) {
23        if (a.substr(i, nc) == c) {
24            q.push_back(i + nc);
25        }
26    }
27    vector<vector<int>> lcp(na + 1, vector<int>(na + 1, 0));
28    vector<pair<int, int>> x;
29    for (auto l : p) {
30        for (auto r : q) {
31            if (r - l >= max(nb, nc)) {
32                x.emplace_back(l, r);
33            }
34        }
35    }
36    for (int i = na - 1; i >= 0; i--) {
37        for (int j = na - 1; j >= 0; j--) {
38            lcp[i][j] = a[i] == a[j] ? 1 + lcp[i + 1][j + 1] : 0;

```

```

39         }
40     }
41     auto cmp = [&](auto x, auto y) {
42         int len = min({x.second - x.first, lcp[x.first][y.first], y.second - y.first});
43         if (len == y.second - y.first) {
44             return false;
45         }
46         if (len == x.second - x.first) {
47             return true;
48         }
49         return a[x.first + len] < a[y.first + len];
50     };
51     sort(x.begin(), x.end(), cmp);
52     int ans = 0;
53     for (int i = 0; i < x.size(); i++) {
54         if (i == 0 || cmp(x[i - 1], x[i])) {
55             ans += 1;
56         }
57     }
58     cout << ans << "\n";
59     return 0;
60 }
```

## 459: Petya, Petya, Petr, and Palindromes

- Time limit: 1.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Petya and his friend, the robot Petya++, have a common friend - the cyborg Petr#. Sometimes Petr# comes to the friends for a cup of tea and tells them interesting problems.

Today, Petr# told them the following problem.

A palindrome is a sequence that reads the same from left to right as from right to left. For example, [38, 12, 8, 12, 38], [1], and [3, 8, 8, 3] are palindromes.

Let's call the palindromicity of a sequence  $a_1, a_2, \dots, a_n$  the minimum count of elements that need to be replaced to make this sequence a palindrome. For example, the palindromicity of the sequence [38, 12, 8, 38, 38] is 1 since it is sufficient to replace the number 38 at the fourth position with the number 12. And the palindromicity of the sequence [3, 3, 5, 5, 5] is two since you can replace the first two threes with fives, and the resulting sequence [5, 5, 5, 5, 5] is a palindrome.

Given a sequence  $a$  of length  $n$ , and an odd integer  $k$ , you need to find the sum of palindromicity of all subarrays of length  $k$ , i. e., the sum of the palindromicity values for the sequences  $a_i, a_{i+1}, \dots, a_{i+k-1}$  for all  $i$  from 1 to  $n - k + 1$ .

The students quickly solved the problem. Can you do it too?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, k;
8     cin >> n >> k;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12        a[i] = 2 * a[i] - i % 2;
13    }
14    i64 ans = 1LL * (k / 2) * (n - k + 1);
15    int m = *max_element(a.begin(), a.end()) + 1;
16    vector<vector<int>> adj(m);
17    for (int i = 0; i < n; i++) {
18        adj[a[i]].push_back(i);
19    }
20    for (auto v : adj) {
21        for (int i = 0, j = 0, x = v.size(), y = v.size(); i < v.size(); i++) {
22            while (v[i] - v[j] >= k) {
23                j++;
24            }
25            while (x > 0 && v[x - 1] + v[i] >= k - 1) {
26                x--;
27            }
28            while (y > 0 && v[y - 1] + v[i] >= 2 * n - k) {
29                y--;
30            }
31            ans -= max(0, min(i, y) - max(x, j));
32        }
33    }
34    cout << ans << "\n";
35    return 0;
36 }
```

#### 460: Mishka and Interesting sum

- Time limit: 3.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Little Mishka enjoys programming. Since her birthday has just passed, her friends decided to present her with array of non-negative integers  $a_1, a_2, \dots, a_n$  of  $n$  elements!

Mishka loved the array and she instantly decided to determine its beauty value, but she is too little and can't process large arrays. Right because of that she invited you to visit her and asked you to process  $m$  queries.

Each query is processed in the following way:

Two integers  $l$  and  $r$  ( $1 \leq l \leq r \leq n$ ) are specified - bounds of query segment.

Integers, presented in array segment  $[l, r]$  (in sequence of integers  $a_l, a_{l+1}, \dots, a_r$ ) even number of times, are written down.

XOR-sum of written down integers is calculated, and this value is the answer for a query. Formally, if integers written down in point 2 are  $x_1, x_2, \dots, x_k$ , then Mishka wants to know the value , where - operator of exclusive bitwise OR.

Since only the little bears know the definition of array beauty, all you are to do is to answer each of queries presented.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template <typename T>
5 # struct Fenwick;
6 # struct Xor;
7 int main() {
8     ios::sync_with_stdio(false);
9     cin.tie(nullptr);
10    int n;
11    cin >> n;
12    vector<int> a(n);
13    for (int i = 0; i < n; i++) {
14        cin >> a[i];
15    }
16    vector<int> s(n + 1);
17    for (int i = 0; i < n; i++) {
18        s[i + 1] = s[i] ^ a[i];
19    }
20    auto v = a;
21    sort(v.begin(), v.end());
22    vector<int> lst(n, -1), nxt(n, -1);
23    for (int i = n - 1; i >= 0; i--) {
24        a[i] = lower_bound(v.begin(), v.end(), a[i]) - v.begin();
25        nxt[i] = lst[a[i]];
26        lst[a[i]] = i;
27    }
28    int m;
29    cin >> m;
30    vector<int> ans(m);
31    vector<vector<array<int, 2>>> qry(n);
32    for (int i = 0; i < m; i++) {
33        int l, r;
34        cin >> l >> r;
35        l--;
36        qry[l].push_back({r, i});
37    }
38    Fenwick<Xor> fen(n);
39    for (int l = n - 1; l >= 0; l--) {
40        fen.add(l, {v[a[l]]});
41        if (nxt[l] != -1) {
42            fen.add(nxt[l], {v[a[l]]});

```

```

43         }
44         for (auto [r, i] : qry[l]) {
45             ans[i] = s[l] ^ s[r] ^ fen.sum(r).x;
46         }
47     }
48     for (int i = 0; i < m; i++) {
49         cout << ans[i] << "\n";
50     }
51     return 0;
52 }
```

### 461: Hot Start Up (hard version)

- Time limit: 1 second
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

This is a hard version of the problem. The constraints of  $t$ ,  $n$ ,  $k$  are the only difference between versions.

You have a device with two CPUs. You also have  $k$  programs, numbered 1 through  $k$ , that you can run on the CPUs.

The  $i$ -th program ( $1 \leq i \leq k$ ) takes  $cold_i$  seconds to run on some CPU. However, if the last program we ran on this CPU was also program  $i$ , it only takes  $hot_i$  seconds ( $hot_i \leq cold_i$ ). Note that this only applies if we run program  $i$  multiple times consecutively - if we run program  $i$ , then some different program, then program  $i$  again, it will take  $cold_i$  seconds the second time.

You are given a sequence  $a_1, a_2, \dots, a_n$  of length  $n$ , consisting of integers from 1 to  $k$ . You need to use your device to run programs  $a_1, a_2, \dots, a_n$  in sequence. For all  $2 \leq i \leq n$ , you cannot start running program  $a_i$  until program  $a_{i-1}$  has completed.

Find the minimum amount of time needed to run all programs  $a_1, a_2, \dots, a_n$  in sequence.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 1E18;
5 void solve() {
6     int n, k;
7     cin >> n >> k;
8     vector<int> a(n);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11     }
12     vector<int> cold(k + 1), hot(k + 1), profit(k + 1);
```

```

13     i64 sum = 0;
14     for (int i = 1; i <= k; i++) {
15         cin >> cold[i];
16     }
17     for (int i = 1; i <= k; i++) {
18         cin >> hot[i];
19         profit[i] = cold[i] - hot[i];
20     }
21     for (int i = 0; i < n; i++) {
22         sum += cold[a[i]];
23     }
24     vector<i64> dp(k + 1, -inf);
25     dp[0] = 0;
26     i64 max = 0;
27     i64 add = 0;
28     for (int i = 1; i < n; i++) {
29         i64 val = max(max, dp[a[i]] + add + profit[a[i]]);
30         if (a[i] == a[i - 1]) {
31             max += profit[a[i]];
32             add += profit[a[i]];
33         }
34         dp[a[i - 1]] = max(dp[a[i - 1]], val - add);
35         max = max(max, val);
36     }
37     i64 ans = sum - max;
38     cout << ans << "\n";
39 }
40 int main() {
41     ios::sync_with_stdio(false);
42     cin.tie(nullptr);
43     int t;
44     cin >> t;
45     while (t--) {
46         solve();
47     }
48     return 0;
49 }
```

## 462: Koxia and Game

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Koxia and Mahiru are playing a game with three arrays  $a$ ,  $b$ , and  $c$  of length  $n$ . Each element of  $a$ ,  $b$  and  $c$  is an integer between 1 and  $n$  inclusive.

The game consists of  $n$  rounds. In the  $i$ -th round, they perform the following moves:

Let  $S$  be the multiset  $\{a_i, b_i, c_i\}$ .

Koxia removes one element from the multiset  $S$  by her choice.

Mahiru chooses one integer from the two remaining in the multiset  $S$ .

Let  $d_i$  be the integer Mahiru chose in the  $i$ -th round. If  $d$  is a permutation<sup>†</sup>, Koxia wins. Otherwise, Mahiru wins.

Currently, only the arrays  $a$  and  $b$  have been chosen. As an avid supporter of Koxia, you want to choose an array  $c$  such that Koxia will win. Count the number of such  $c$ , modulo 998 244 353.

Note that Koxia and Mahiru both play optimally.

<sup>†</sup> A permutation of length  $n$  is an array consisting of  $n$  distinct integers from 1 to  $n$  in arbitrary order. For example,  $[2, 3, 1, 5, 4]$  is a permutation, but  $[1, 2, 2]$  is not a permutation (2 appears twice in the array), and  $[1, 3, 4]$  is also not a permutation ( $n = 3$  but there is 4 in the array).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 998244353;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
8     if (x < 0) {
9         x += P;
10    }
11    if (x >= P) {
12        x -= P;
13    }
14    return x;
15 }
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25 }
26 # struct Z;
27 void solve() {
28     int n;
29     cin >> n;
30     vector<int> a(n), b(n);
31     for (int i = 0; i < n; i++) {
32         cin >> a[i];
33         a[i]--;
34     }
35     for (int i = 0; i < n; i++) {
36         cin >> b[i];
37         b[i]--;
38     }
39     vector<vector<int>> adj(n);
40     for (int i = 0; i < n; i++) {
41         adj[a[i]].push_back(b[i]);
42         adj[b[i]].push_back(a[i]);
43     }
44     Z ans = 1;

```

```

45     vector<int> vis(n);
46     for (int i = 0; i < n; i++) {
47         if (vis[i]) continue;
48         queue<int> q;
49         q.push(i);
50         vis[i] = 1;
51         int edges = 0, ver = 0;
52         while (!q.empty()) {
53             int x = q.front();
54             q.pop();
55             edges += adj[x].size();
56             ver++;
57             for (auto y : adj[x]) {
58                 if (!vis[y]) {
59                     vis[y] = 1;
60                     q.push(y);
61                 }
62             }
63             edges /= 2;
64             if (edges != ver) {
65                 ans = 0;
66             } else {
67                 ans *= 2;
68             }
69         }
70         for (int i = 0; i < n; i++) {
71             if (a[i] == b[i]) {
72                 ans /= 2;
73                 ans *= n;
74             }
75         }
76         cout << ans << "\n";
77     }
78 }
79 int main() {
80     ios::sync_with_stdio(false);
81     cin.tie(nullptr);
82     int t;
83     cin >> t;
84     while (t--) {
85         solve();
86     }
87     return 0;
88 }
```

### 463: Count Binary Strings

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given an integer  $n$ . You have to calculate the number of binary (consisting of characters 0 and/or 1) strings  $s$  meeting the following constraints.

For every pair of integers  $(i, j)$  such that  $1 \leq i \leq j \leq n$ , an integer  $a_{i,j}$  is given. It imposes the following constraint on the string  $s_i s_{i+1} s_{i+2} \dots s_j$ :

if  $a_{i,j} = 1$ , all characters in  $s_i s_{i+1} s_{i+2} \dots s_j$  should be the same;

if  $a_{i,j} = 2$ , there should be at least two different characters in  $s_i s_{i+1} s_{i+2} \dots s_j$ ;

if  $a_{i,j} = 0$ , there are no additional constraints on the string  $s_i s_{i+1} s_{i+2} \dots s_j$ .

Count the number of binary strings  $s$  of length  $n$  meeting the aforementioned constraints. Since the answer can be large, print it modulo 998244353.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 constexpr int P = 998244353;
4 using i64 = long long;
5 // assume -P <= x < 2P
6 int norm(int x) {
7     if (x < 0) {
8         x += P;
9     }
10    if (x >= P) {
11        x -= P;
12    }
13    return x;
14 }
15 template<class T>
16 T power(T a, i64 b) {
17     T res = 1;
18     for (; b; b /= 2, a *= a) {
19         if (b % 2) {
20             res *= a;
21         }
22     }
23     return res;
24 }
25 # struct Z;
26 int main() {
27     ios::sync_with_stdio(false);
28     cin.tie(nullptr);
29     int n;
30     cin >> n;
31     vector a(n, vector<int>(n));
32     for (int i = 0; i < n; i++) {
33         for (int j = i; j < n; j++) {
34             cin >> a[i][j];
35         }
36     }
37     vector dp(n + 1, vector<Z>(n + 1));
38     dp[0][0] = 1;
39     Z ans = 0;
40     for (int i = 0; i <= n; i++) {
41         for (int j = 0; j <= n; j++) {
42             int x = max(i, j) + 1;
43             if (x > n) {
44                 ans += dp[i][j];
45                 continue;
46             }
47             int ok = 1;

```

```

48         for (int k = 0; k < x; k++) {
49             if (a[k][x - 1] == 1 && k < j) ok = 0;
50             if (a[k][x - 1] == 2 && k >= j) ok = 0;
51         }
52         if (ok) dp[x][j] += dp[i][j];
53         ok = 1;
54         for (int k = 0; k < x; k++) {
55             if (a[k][x - 1] == 1 && k < i) ok = 0;
56             if (a[k][x - 1] == 2 && k >= i) ok = 0;
57         }
58         if (ok) dp[i][x] += dp[i][j];
59     }
60 }
61 cout << ans << "\n";
62 return 0;
63 }
```

#### 464: Hossam and (sub-)palindromic tree

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Hossam has an unweighted tree  $G$  with letters in vertices.

Hossam defines  $s(v, u)$  as a string that is obtained by writing down all the letters on the unique simple path from the vertex  $v$  to the vertex  $u$  in the tree  $G$ .

A string  $a$  is a subsequence of a string  $s$  if  $a$  can be obtained from  $s$  by deletion of several (possibly, zero) letters. For example, “dores”, “cf”, and “for” are subsequences of “codeforces”, while “decor” and “fork” are not.

A palindrome is a string that reads the same from left to right and from right to left. For example, “abacaba” is a palindrome, but “abac” is not.

Hossam defines a sub-palindrome of a string  $s$  as a subsequence of  $s$ , that is a palindrome. For example, “k”, “abba” and “abhba” are sub-palindromes of the string “abhbka”, but “abka” and “cat” are not.

Hossam defines a maximal sub-palindrome of a string  $s$  as a sub-palindrome of  $s$ , which has the maximal length among all sub-palindromes of  $s$ . For example, “abhbka” has only one maximal sub-palindrome - “abhba”. But it may also be that the string has several maximum sub-palindromes: the string “abcd” has 4 maximum sub-palindromes.

Help Hossam find the length of the longest maximal sub-palindrome among all  $s(v, u)$  in the tree  $G$ .

Note that the sub-palindrome is a subsequence, not a substring.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     string s;
8     cin >> s;
9     vector<vector<int>> adj(n);
10    for (int i = 1; i < n; i++) {
11        int u, v;
12        cin >> u >> v;
13        u--, v--;
14        adj[u].push_back(v);
15        adj[v].push_back(u);
16    }
17    vector f(n + n, vector<int>(n + n, -1));
18    vector<int> parent(n, -1);
19    auto dfs = [&](auto dfs, int x) -> void {
20        for (auto y : adj[x]) {
21            if (y == parent[x]) {
22                continue;
23            }
24            parent[y] = x;
25            dfs(dfs, y);
26        }
27    };
28    dfs(dfs, 0);
29    int ans = 1;
30    auto rec = [&](auto rec, int a, int b, int c, int d) {
31        int u = parent[a] == b ? a : b + n;
32        int v = parent[c] == d ? c : d + n;
33        if (f[u][v] != -1) {
34            return f[u][v];
35        }
36        int &res = f[u][v];
37        res = 0;
38        for (auto x : adj[a]) {
39            if (x == b) {
40                continue;
41            }
42            res = max(res, rec(rec, x, a, c, d));
43        }
44        for (auto y : adj[c]) {
45            if (y == d) {
46                continue;
47            }
48            res = max(res, rec(rec, a, b, y, c));
49        }
50        if (s[a] == s[c]) {
51            res = max(res, 1);
52            for (auto x : adj[a]) {
53                if (x == b) {
54                    continue;
55                }
56                for (auto y : adj[c]) {
57                    if (y == d) {
58                        continue;
59                    }
60                    res = max(res, 1 + rec(rec, x, a, y, c));
61                }
62            }
63        }
64    };

```

```

63         }
64     return res;
65 }
66 for (int i = 0; i < n; i++) {
67     for (auto j : adj[i]) {
68         ans = max(ans, 2 * rec(rec, i, j, j, i));
69         for (auto k : adj[i]) {
70             if (j != k) {
71                 ans = max(ans, 2 * rec(rec, j, i, k, i) + 1);
72             }
73         }
74     }
75 }
76 cout << ans << "\n";
77 }
78 int main() {
79     ios::sync_with_stdio(false);
80     cin.tie(nullptr);
81     int t;
82     cin >> t;
83     while (t--) {
84         solve();
85     }
86     return 0;
87 }
```

## 465: Multi-Colored Segments

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Dmitry has  $n$  segments of different colors on the coordinate axis  $Ox$ . Each segment is characterized by three integers  $l_i, r_i$  and  $c_i$  ( $1 \leq l_i \leq r_i \leq 10^9, 1 \leq c_i \leq n$ ), where  $l_i$  and  $r_i$  are the coordinates of the ends of the  $i$ -th segment, and  $c_i$  is its color.

Dmitry likes to find the minimum distances between segments. However, he considers pairs of segments of the same color uninteresting. Therefore, he wants to know for each segment the distance from this segment to the nearest differently colored segment.

The distance between two segments is the minimum of the distances between a point of the first segment and a point of the second segment. In particular, if the segments intersect, then the distance between them is equal to 0.

For example, Dmitry has 5 segments:

The first segment intersects with the second (and these are segments of different colors), so the answers for them are equal to 0.

For the 3-rd segment, the nearest segment of a different color is the 2-nd segment, the distance to which is equal to 2.

For the 4-th segment, the nearest segment of a different color is the 5-th segment, the distance to which is equal to 1.

The 5-th segment lies inside the 2-nd segment (and these are segments of different colors), so the answers for them are equal to 0.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int inf = 1E9;
5 void solve() {
6     int n;
7     cin >> n;
8     vector<array<int, 5>> pts(2 * n);
9     for (int i = 0; i < n; i++) {
10         int l, r, c;
11         cin >> l >> r >> c;
12         pts[2 * i] = {l, r, c, i, 0};
13         pts[2 * i + 1] = {r, l, c, i, 1};
14     }
15     sort(pts.begin(), pts.end());
16     vector<int> ans(n, inf);
17     for (int t = 0; t < 2; t++) {
18         array<int, 2> f[2];
19         f[0] = f[1] = {-inf, -1};
20         for (auto [x, y, c, i, o] : pts) {
21             if (!o) {
22                 array g{y, c};
23                 if (g > f[0]) {
24                     swap(g, f[0]);
25                 }
26                 if ((g > f[1] && g[1] != f[0][1]) || f[0][1] == f[1][1]) {
27                     f[1] = g;
28                 }
29             } else {
30                 for (auto [z, d] : f) {
31                     if (d != c) {
32                         ans[i] = min(ans[i], max(0, y - z));
33                     }
34                 }
35             }
36         }
37         reverse(pts.begin(), pts.end());
38         for (auto &[x, y, c, i, o] : pts) {
39             x = inf - x;
40             y = inf - y;
41             o ^= 1;
42         }
43     }
44     for (int i = 0; i < n; i++) {
45         cout << ans[i] << " \n"[i == n - 1];
46     }
}

```

```

47 }
48 int main() {
49     ios::sync_with_stdio(false);
50     cin.tie(nullptr);
51     int t;
52     cin >> t;
53     while (t--) {
54         solve();
55     }
56     return 0;
57 }
```

**dp****466: Light Bulbs (Easy Version)**

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

The easy and hard versions of this problem differ only in the constraints on  $n$ . In the easy version, the sum of values of  $n^2$  over all test cases does not exceed  $10^6$ . Furthermore,  $n$  does not exceed 1000 in each test case.

There are  $2n$  light bulbs arranged in a row. Each light bulb has a color from 1 to  $n$  (exactly two light bulbs for each color).

Initially, all light bulbs are turned off. You choose a set of light bulbs  $S$  that you initially turn on. After that, you can perform the following operations in any order any number of times:

choose two light bulbs  $i$  and  $j$  of the same color, exactly one of which is on, and turn on the second one;

choose three light bulbs  $i, j, k$ , such that both light bulbs  $i$  and  $k$  are on and have the same color, and the light bulb  $j$  is between them ( $i < j < k$ ), and turn on the light bulb  $j$ .

You want to choose a set of light bulbs  $S$  that you initially turn on in such a way that by performing the described operations, you can ensure that all light bulbs are turned on.

Calculate two numbers:

the minimum size of the set  $S$  that you initially turn on;

the number of sets  $S$  of minimum size (taken modulo 998244353).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
35 void solve() {
36     int n;
37     cin >> n;
38     vector<i64> w(n);
39     vector<int> c(2 * n);
40     for (int i = 0; i < n; i++) {
41         w[i] = rng();
42     }
43     vector<i64> f(2 * n + 1);
44     for (int i = 0; i < 2 * n; i++) {
45         cin >> c[i];
46         c[i]--;
47         f[i + 1] = f[i] ^ w[c[i]];
48     }
49     int ans = count(f.begin(), f.end(), 0) - 1;
50     Z ways = 1;
51     map<i64, int> last;
52     for (int i = 0; i <= 2 * n; i++) {
53         last[f[i]] = i;
54     }
55     for (int i = 0; i < 2 * n; i++) {
56         if (f[i] == 0) {
57             int cnt = 1;
58             int j = i + 1;
59             while (f[j] != 0) {
60                 cnt += 1;
61                 j = last[f[j]];
62                 j += 1;
63             }
64             ways *= cnt;
65         }
66     }
67 }
```

```

63         }
64         ways *= cnt;
65     }
66     cout << ans << " " << ways << "\n";
67 }
68 int main() {
69     ios::sync_with_stdio(false);
70     cin.tie(nullptr);
71     int t;
72     cin >> t;
73     while (t--) {
74         solve();
75     }
76     return 0;
77 }
```

## 467: Array Collapse

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $[p_1, p_2, \dots, p_n]$ , where all elements are distinct.

You can perform several (possibly zero) operations with it. In one operation, you can choose a contiguous subsegment of  $p$  and remove all elements from that subsegment, except for the minimum element on that subsegment. For example, if  $p = [3, 1, 4, 7, 5, 2, 6]$  and you choose the subsegment from the 3-rd element to the 6-th element, the resulting array is  $[3, 1, 2, 6]$ .

An array  $a$  is called reachable if it can be obtained from  $p$  using several (maybe zero) aforementioned operations. Calculate the number of reachable arrays, and print it modulo 998244353.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
```

```

14  constexpr i64 mul(i64 a, i64 b, i64 p) {
15      i64 res = a * b - i64(1.L * a * b / p) * p;
16      res %= p;
17      if (res < 0) {
18          res += p;
19      }
20      return res;
21  }
22  template<i64 P>
23  # struct MLong;
24  template<>
25  i64 MLong<0LL>::Mod = i64(1E18) + 9;
26  template<int P>
27  # struct MInt;
28  template<>
29  int MInt<0>::Mod = 998244353;
30  template<int V, int P>
31  constexpr MInt<P> CInv = MInt<P>(V).inv();
32  constexpr int P = 998244353;
33  using Z = MInt<P>;
34  void solve() {
35      int n;
36      cin >> n;
37      vector<int> p(n + 2);
38      p[0] = 1E9 + 1;
39      p[n + 1] = 1E9 + 2;
40      for (int i = 1; i <= n; i++) {
41          cin >> p[i];
42      }
43      n += 2;
44      vector<int> l(n, -1), r(n, n);
45      vector<int> stk;
46      for (int i = 0; i < n; i++) {
47          while (!stk.empty() && p[i] < p[stk.back()]) {
48              int x = stk.back();
49              stk.pop_back();
50              r[x] = i;
51          }
52          if (!stk.empty()) {
53              l[i] = stk.back();
54          }
55          stk.push_back(i);
56      }
57      vector<Z> dp(n), d(n), s(n);
58      dp[0] = 1;
59      s[0] = 1;
60      for (int i = 0; i < n; i++) {
61          if (i) {
62              dp[i] += s[i - 1];
63              d[i] += d[i - 1];
64              if (l[i] >= 0) {
65                  dp[i] -= s[l[i]];
66              }
67              dp[i] += d[i];
68              s[i] = s[i - 1] + dp[i];
69          }
70          if (i < n - 1) {
71              d[i + 1] += dp[i];
72              if (r[i] < n) {
73                  d[r[i]] -= dp[i];
74              }
75          }
76      }
77      Z ans = dp[n - 1];

```

```

78     cout << ans << "\n";
79 }
80 int main() {
81     ios::sync_with_stdio(false);
82     cin.tie(nullptr);
83     int t;
84     cin >> t;
85     while (t--) {
86         solve();
87     }
88     return 0;
89 }
```

## 468: Count BFS Graph

- Time limit: 1 second
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

You are currently researching a graph traversal algorithm called the Breadth First Search (BFS). Suppose you have an input graph of  $N$  nodes (numbered from 1 to  $N$ ). The graph is represented by an adjacency matrix  $M$ , for which node  $u$  can traverse to node  $v$  if  $M_{u,v}$  is 1, otherwise it is 0. Your algorithm will output the order the nodes are visited in the BFS. The pseudocode of the algorithm is presented as follows.

During your research, you are interested in the following problem. Given an array  $A$  such that  $A$  is a permutation of 1 to  $N$  and  $A_1 = 1$ . How many simple undirected graph with  $N$  nodes and adjacency matrix  $M$  such that  $\text{BFS}(M) = A$ ? Since the answer can be very large, calculate the answer modulo 998 244 353.

A simple graph has no self-loop ( $M_{i,i} = 0$  for  $1 \leq i \leq N$ ) and there is at most one edge that connects a pair of nodes. In an undirected graph, if node  $u$  is adjacent to node  $v$ , then node  $v$  is also adjacent to node  $u$ ; formally,  $M_{u,v} = M_{v,u}$  for  $1 \leq u < v \leq N$ .

Two graphs are considered different if there is an edge that exists in one graph but not the other. In other words, two graphs are considered different if their adjacency matrices are different.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
```

```
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int N;
38     cin >> N;
39     vector<int> A(N);
40     vector<Z> dp(N), pw(N);
41     pw[0] = 1;
42     for (int i = 1; i < N; i++) {
43         pw[i] = pw[i - 1] * 2;
44     }
45     for (int i = 0; i < N; i++) {
46         cin >> A[i];
47     }
48     dp[0] = 1;
49     for (int i = 2; i < N; i++) {
50         Z sum = 0;
51         vector<Z> g(N);
52         for (int j = 0; j < i; j++) {
53             if (A[i] > A[i - 1]) {
54                 sum += dp[j];
55             }
56             g[j] = sum * pw[i - j - 1];
57             if (A[i] < A[i - 1]) {
58                 sum += dp[j];
59             }
60         }
61         dp = move(g);
62     }
63     Z ans = accumulate(dp.begin(), dp.end(), Z(0));
64     cout << ans << "\n";
65     return 0;
66 }
```

## 469: Transitive Graph

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a directed graph  $G$  with  $n$  vertices and  $m$  edges between them.

Initially, graph  $H$  is the same as graph  $G$ . Then you decided to perform the following actions:

If there exists a triple of vertices  $a, b, c$  of  $H$ , such that there is an edge from  $a$  to  $b$  and an edge from  $b$  to  $c$ , but there is no edge from  $a$  to  $c$ , add an edge from  $a$  to  $c$ .

Repeat the previous step as long as there are such triples.

Note that the number of edges in  $H$  can be up to  $n^2$  after performing the actions.

You also wrote some values on vertices of graph  $H$ . More precisely, vertex  $i$  has the value of  $a_i$  written on it.

Consider a simple path consisting of  $k$  distinct vertices with indexes  $v_1, v_2, \dots, v_k$ . The length of such a path is  $k$ . The value of that path is defined as  $\sum_{i=1}^k a_{v_i}$ .

A simple path is considered the longest if there is no other simple path in the graph with greater length.

Among all the longest simple paths in  $H$ , find the one with the smallest value.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct SCC;
5 void solve() {
6     int n, m;
7     cin >> n >> m;
8     vector<int> a(n);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11     }
12     SCC g(n);
13     for (int i = 0; i < m; i++) {
14         int u, v;
15         cin >> u >> v;
16         u--, v--;
17         g.addEdge(u, v);
18     }
19     auto bel = g.work();
20     vector<vector<int>> adj(n);
21     vector<pair<int, i64>> s(g.cnt), dp(n);

```

```

22     for (int i = 0; i < n; i++) {
23         s[bel[i]].first += 1;
24         s[bel[i]].second += -a[i];
25         for (auto j : g.adj[i]) {
26             if (bel[i] != bel[j]) {
27                 adj[bel[i]].push_back(bel[j]);
28             }
29         }
30     }
31     for (int i = g.cnt - 1; i >= 0; i--) {
32         dp[i].first += s[i].first;
33         dp[i].second += s[i].second;
34         for (auto j : adj[i]) {
35             dp[j] = max(dp[j], dp[i]);
36         }
37     }
38     auto ans = *max_element(dp.begin(), dp.end());
39     cout << ans.first << " " << -ans.second << "\n";
40 }
41 int main() {
42     ios::sync_with_stdio(false);
43     cin.tie(nullptr);
44     int t;
45     cin >> t;
46     while (t--) {
47         solve();
48     }
49     return 0;
50 }
```

## 470: Small GCD

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Let  $a$ ,  $b$ , and  $c$  be integers. We define function  $f(a, b, c)$  as follows:

Order the numbers  $a$ ,  $b$ ,  $c$  in such a way that  $a \leq b \leq c$ . Then return  $\gcd(a, b)$ , where  $\gcd(a, b)$  denotes the greatest common divisor (GCD) of integers  $a$  and  $b$ .

So basically, we take the gcd of the 2 smaller values and ignore the biggest one.

You are given an array  $a$  of  $n$  elements. Compute the sum of  $f(a_i, a_j, a_k)$  for each  $i, j, k$ , such that  $1 \leq i < j < k \leq n$ .

More formally, compute

$$\sum_{i=1}^n \sum_{j=i+1}^n \sum_{k=j+1}^n f(a_i, a_j, a_k).$$

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int N = 1E5;
5 vector<int> divs[N + 1];
6 int phi[N + 1];
7 void solve() {
8     int n;
9     cin >> n;
10    vector<int> a(n);
11    for (int i = 0; i < n; i++) {
12        cin >> a[i];
13    }
14    sort(a.begin(), a.end());
15    i64 ans = 0;
16    array<int, N + 1> f{};
17    for (int i = 0; i < n; i++) {
18        for (auto d : divs[a[i]]) {
19            ans += 1LL * phi[d] * f[d] * (n - 1 - i);
20            f[d] += 1;
21        }
22    }
23    cout << ans << "\n";
24 }
25 int main() {
26     ios::sync_with_stdio(false);
27     cin.tie(nullptr);
28     for (int i = 1; i <= N; i++) {
29         for (int j = i; j <= N; j += i) {
30             divs[j].push_back(i);
31         }
32     }
33     for (int i = 1; i <= N; i++) {
34         phi[i] = i;
35     }
36     for (int i = 1; i <= N; i++) {
37         for (int j = 2 * i; j <= N; j += i) {
38             phi[j] -= phi[i];
39         }
40     }
41     int t;
42     cin >> t;
43     while (t--) {
44         solve();
45     }
46     return 0;
47 }
```

## 471: Counting Rhyme

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array of integers  $a_1, a_2, \dots, a_n$ .

A pair of integers  $(i, j)$ , such that  $1 \leq i < j \leq n$ , is called good, if there does not exist an integer  $k$  ( $1 \leq k \leq n$ ) such that  $a_i$  is divisible by  $a_k$  and  $a_j$  is divisible by  $a_k$  at the same time.

Please, find the number of good pairs.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     i64 ans = 0;
8     vector<i64> f(n + 1);
9     vector<int> g(n + 1);
10    for (int i = 0; i < n; i++) {
11        int a;
12        cin >> a;
13        f[a] += 1;
14        g[a] = 1;
15    }
16    for (int i = 1; i <= n; i++) {
17        for (int j = 2 * i; j <= n; j += i) {
18            g[j] |= g[i];
19            f[i] += f[j];
20        }
21    }
22    for (int i = 1; i <= n; i++) {
23        f[i] *= f[i];
24    }
25    for (int i = n; i; i--) {
26        for (int j = 2 * i; j <= n; j += i) {
27            f[i] -= f[j];
28        }
29    }
30    for (int i = 1; i <= n; i++) {
31        if (!g[i]) {
32            ans += f[i];
33        }
34    }
35    ans /= 2;
36    cout << ans << "\n";
37 }
38 int main() {
39     ios::sync_with_stdio(false);
40     cin.tie(nullptr);
41     int t;
42     cin >> t;
43     while (t--) {
44         solve();
45     }
46     return 0;
47 }
```

## 472: Pchelyonok and Segments

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Pchelyonok decided to give Mila a gift. Pchelenok has already bought an array  $a$  of length  $n$ , but gifting an array is too common. Instead of that, he decided to gift Mila the segments of that array!

Pchelyonok wants his gift to be beautiful, so he decided to choose  $k$  non-overlapping segments of the array  $[l_1, r_1], [l_2, r_2], \dots [l_k, r_k]$  such that:

the length of the first segment  $[l_1, r_1]$  is  $k$ , the length of the second segment  $[l_2, r_2]$  is  $k - 1, \dots$ , the length of the  $k$ -th segment  $[l_k, r_k]$  is 1

for each  $i < j$ , the  $i$ -th segment occurs in the array earlier than the  $j$ -th (i.e.  $r_i < l_j$ )

the sums in these segments are strictly increasing (i.e. let  $\text{sum}(l \dots r) = \sum_{i=l}^r a_i$  - the sum of numbers in the segment  $[l, r]$  of the array, then  $\text{sum}(l_1 \dots r_1) < \text{sum}(l_2 \dots r_2) < \dots < \text{sum}(l_k \dots r_k)$ ).

Pchelenok also wants his gift to be as beautiful as possible, so he asks you to find the maximal value of  $k$  such that he can give Mila a gift!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<i64> s(n + 1);
12    for (int i = 0; i < n; i++) {
13        s[i + 1] = s[i] + a[i];
14    }
15    vector<i64> dp(n + 1, 1E18);
16    for (int ans = 1; ; ans++) {
17        for (int i = 0; i <= n; i++) {
18            if (i > n - ans) {
19                dp[i] = 0;
20            } else {
21                dp[i] = dp[i + ans] > s[i + ans] - s[i] ? s[i + ans] - s[i] : 0;
22            }
23        }
24        for (int i = n - 1; i >= 0; i--) {

```

```

25         dp[i] = max(dp[i], dp[i + 1]);
26     }
27     if (dp[0] == 0) {
28         cout << ans - 1 << "\n";
29         return;
30     }
31 }
32 int main() {
33     ios::sync_with_stdio(false);
34     cin.tie(nullptr);
35     int t;
36     cin >> t;
37     while (t--) {
38         solve();
39     }
40     return 0;
41 }
```

### 473: Candy Party (Hard Version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. The only difference is that in this version everyone must give candies to no more than one person and receive candies from no more than one person. Note that a submission cannot pass both versions of the problem at the same time. You can make hacks only if both versions of the problem are solved.

After Zhongkao examination, Daniel and his friends are going to have a party. Everyone will come with some candies.

There will be  $n$  people at the party. Initially, the  $i$ -th person has  $a_i$  candies. During the party, they will swap their candies. To do this, they will line up in an arbitrary order and everyone will do the following no more than once:

Choose an integer  $p$  ( $1 \leq p \leq n$ ) and a non-negative integer  $x$ , then give his  $2^x$  candies to the  $p$ -th person. Note that one cannot give more candies than currently he has (he might receive candies from someone else before) and he cannot give candies to himself.

Daniel likes fairness, so he will be happy if and only if everyone receives candies from no more than one person. Meanwhile, his friend Tom likes average, so he will be happy if and only if all the people have the same number of candies after all swaps.

Determine whether there exists a way to swap candies, so that both Daniel and Tom will be happy after the swaps.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    i64 sum = accumulate(a.begin(), a.end(), 0LL);
12    if (sum % n != 0) {
13        cout << "No\n";
14        return;
15    }
16    int ave = sum / n;
17    int cnt[31] {};
18    int pos[31] {};
19    int neg[31] {};
20    for (int i = 0; i < n; i++) {
21        if (a[i] < ave) {
22            int d = ave - a[i];
23            if (!(d & (d - 1))) {
24                neg[__builtin_ctz(d)]++;
25            }
26            int l = __builtin_ctz(d);
27            cnt[l]++;
28            d += 1 << l;
29            if (d & (d - 1)) {
30                cout << "No\n";
31                return;
32            }
33            cnt[__builtin_ctz(d)]--;
34        } else if (a[i] > ave) {
35            int d = a[i] - ave;
36            if (!(d & (d - 1))) {
37                pos[__builtin_ctz(d)]++;
38            }
39            int l = __builtin_ctz(d);
40            cnt[l]--;
41            d += 1 << l;
42            if (d & (d - 1)) {
43                cout << "No\n";
44                return;
45            }
46            cnt[__builtin_ctz(d)]++;
47        }
48    }
49    for (int i = 30; i > 0; i--) {
50        if (cnt[i] > 0) {
51            int t = min(pos[i - 1], cnt[i]);
52            cnt[i] -= t;
53            cnt[i - 1] += 2 * t;
54        }
55        if (cnt[i] < 0) {
56            int t = min(neg[i - 1], -cnt[i]);
57            cnt[i] += t;
58            cnt[i - 1] -= 2 * t;
59        }
60    }

```

```

61     for (int i = 0; i <= 30; i++) {
62         if (cnt[i]) {
63             cout << "No\n";
64             return;
65         }
66     }
67     cout << "Yes\n";
68 }
69 int main() {
70     ios::sync_with_stdio(false);
71     cin.tie(nullptr);
72     int t;
73     cin >> t;
74     while (t--) {
75         solve();
76     }
77     return 0;
78 }
```

**474: Speedrun**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are playing a video game. The game has  $n$  quests that need to be completed. However, the  $j$ -th quest can only be completed at the beginning of hour  $h_j$  of a game day. The game day is  $k$  hours long. The hours of each game day are numbered  $0, 1, \dots, k - 1$ . After the first day ends, a new one starts, and so on.

Also, there are dependencies between the quests, that is, for some pairs  $(a_i, b_i)$  the  $b_i$ -th quest can only be completed after the  $a_i$ -th quest. It is guaranteed that there are no circular dependencies, as otherwise the game would be unbeatable and nobody would play it.

You are skilled enough to complete any number of quests in a negligible amount of time (i. e. you can complete any number of quests at the beginning of the same hour, even if there are dependencies between them). You want to complete all quests as fast as possible. To do this, you can complete the quests in any valid order. The completion time is equal to the difference between the time of completing the last quest and the time of completing the first quest in this order.

Find the least amount of time you need to complete the game.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
```

```

4  constexpr i64 inf = 1E18;
5  void solve() {
6      int n, m, k;
7      cin >> n >> m >> k;
8      vector<int> h(n);
9      for (int i = 0; i < n; i++) {
10         cin >> h[i];
11     }
12     vector<vector<int>> adj(n);
13     vector<int> deg(n);
14     for (int i = 0; i < m; i++) {
15         int u, v;
16         cin >> u >> v;
17         u--;
18         v--;
19         adj[u].push_back(v);
20         deg[v]++;
21     }
22     vector<int> q;
23     for (int i = 0; i < n; i++) {
24         if (deg[i] == 0) {
25             q.push_back(i);
26         }
27     }
28     for (int i = 0; i < n; i++) {
29         int x = q[i];
30         for (auto y : adj[x]) {
31             if (--deg[y] == 0) {
32                 q.push_back(y);
33             }
34         }
35     }
36     vector<i64> dp(n);
37     for (int i = n - 1; i >= 0; i--) {
38         int x = q[i];
39         for (auto y : adj[x]) {
40             dp[x] = max(dp[x], dp[y] + (h[y] - h[x] + k) % k);
41         }
42     }
43     vector<int> p(n);
44     iota(p.begin(), p.end(), 0);
45     sort(p.begin(), p.end(),
46          [&](int i, int j) {
47              return h[i] < h[j];
48          });
49     i64 ans = inf;
50     for (int i = 0; i < n; i++) {
51         dp[i] += h[i];
52     }
53     i64 res = *max_element(dp.begin(), dp.end());
54     for (auto i : p) {
55         ans = min(ans, res - h[i]);
56         res = max(res, dp[i] + k);
57     }
58     cout << ans << "\n";
59 }
60 int main() {
61     ios::sync_with_stdio(false);
62     cin.tie(nullptr);
63     int t;
64     cin >> t;
65     while (t--) {
66         solve();
67     }
68     return 0;

```

68 }

## 475: Maximum Questions

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vasya wrote down two strings  $s$  of length  $n$  and  $t$  of length  $m$  consisting of small English letters ‘a’ and ‘b’. What is more, he knows that string  $t$  has a form “abab...”, namely there are letters ‘a’ on odd positions and letters ‘b’ on even positions.

Suddenly in the morning, Vasya found that somebody spoiled his string. Some letters of the string  $s$  were replaced by character ‘?’.

Let’s call a sequence of positions  $i, i + 1, \dots, i + m - 1$  as occurrence of string  $t$  in  $s$ , if  $1 \leq i \leq n - m + 1$  and  $t_1 = s_i, t_2 = s_{i+1}, \dots, t_m = s_{i+m-1}$ .

The boy defines the beauty of the string  $s$  as maximum number of disjoint occurrences of string  $t$  in  $s$ . Vasya can replace some letters ‘?’ with ‘a’ or ‘b’ (letters on different positions can be replaced with different letter). Vasya wants to make some replacements in such a way that beauty of string  $s$  is maximum possible. From all such options, he wants to choose one with the minimum number of replacements. Find the number of replacements he should make.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int inf = 1E9;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n;
9     cin >> n;
10    string s;
11    cin >> s;
12    int m;
13    cin >> m;
14    vector<array<int, 2>> dp(n + 1, {-inf, -inf});
15    dp[0] = {0, 0};
16    vector sum(2, vector(2, vector<int>(n + 1)));
17    for (int x = 0; x < 2; x++) {
18        for (int y = 0; y < 2; y++) {
19            for (int i = 0; i < n; i++) {

```

```

20             sum[x][y][i + 1] = sum[x][y][i] + (i % 2 == x && s[i] == 'a' + y);
21         }
22     }
23 }
24 for (int i = 1; i <= n; i++) {
25     dp[i] = dp[i - 1];
26     if (i >= m) {
27         int t = (i - m) % 2;
28         if (sum[t][1][i] - sum[t][1][i - m] > 0) {
29             continue;
30         }
31         if (sum[t ^ 1][0][i] - sum[t ^ 1][0][i - m] > 0) {
32             continue;
33         }
34         int ques = m - (sum[t][0][i] - sum[t][0][i - m]) - (sum[t ^ 1][1][i] - sum[t ^
35             1][1][i - m]);
36         dp[i] = max(dp[i], array{dp[i - m][0] + 1, dp[i - m][1] - ques});
37     }
38     cout << -dp[n][1] << "\n";
39 }
40 }
```

## 476: Unusual Sequences

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Count the number of distinct sequences  $a_1, a_2, \dots, a_n$  ( $1 \leq a_i$ ) consisting of positive integers such that  $\gcd(a_1, a_2, \dots, a_n) = x$  and  $x$ . As this number could be large, print the answer modulo  $10^9 + 7$ .

$\gcd$  here means the greatest common divisor.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
```

```

16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int x, y;
38     cin >> x >> y;
39     if (y % x != 0) {
40         cout << 0 << "\n";
41         return 0;
42     }
43     y /= x;
44     Z ans = 0;
45     vector<int> p;
46     x = y;
47     for (int i = 2; i * i <= x; i++) {
48         if (x % i == 0) {
49             p.push_back(i);
50             while (x % i == 0) {
51                 x /= i;
52             }
53         }
54     }
55     if (x > 1) {
56         p.push_back(x);
57     }
58     auto dfs = [&](auto self, int i, int x, int m) {
59         if (i == p.size()) {
60             ans += power(Z(2), y / x - 1) * m;
61             return;
62         }
63         self(self, i + 1, x, m);
64         self(self, i + 1, x * p[i], -m);
65     };
66     dfs(dfs, 0, 1, 1);
67     cout << ans << "\n";
68     return 0;
69 }
```

**477: Square Subsets**

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

Petya was late for the lesson too. The teacher gave him an additional task. For some array a Petya should find the number of different ways to select non-empty subset of elements from it in such a way that their product is equal to a square of some integer.

Two ways are considered different if sets of indexes of elements chosen by these ways are different.

Since the answer can be very large, you should find the answer modulo  $10^9 + 7$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 vector<int> minp, primes;
35 void sieve(int n) {
36     minp.assign(n + 1, 0);
37     primes.clear();
38     for (int i = 2; i <= n; i++) {
39         if (minp[i] == 0) {
40             minp[i] = i;
41             primes.push_back(i);
42         }
43         for (auto p : primes) {
44             if (i * p > n) {
45                 break;
46             }
47             minp[i * p] = 1;
48         }
49     }
50 }
```

```

46         }
47         minp[i * p] = p;
48         if (p == minp[i]) {
49             break;
50         }
51     }
52 }
53 int main() {
54     ios::sync_with_stdio(false);
55     cin.tie(nullptr);
56     sieve(70);
57     int n;
58     cin >> n;
59     vector<int> t(19);
60     vector<int> a(n);
61     vector<int> id(70, -1);
62     for (int i = 0; i < primes.size(); i++) {
63         id[primes[i]] = i;
64     }
65     Z ans = 1;
66     for (int i = 0; i < n; i++) {
67         cin >> a[i];
68         int mask = 0;
69         int x = a[i];
70         while (x > 1) {
71             int p = minp[x];
72             mask ^= 1 << id[p];
73             x /= p;
74         }
75         for (int j = 0; j < 19; j++) {
76             if (mask >> j & 1) {
77                 if (t[j]) {
78                     mask ^= t[j];
79                 } else {
80                     t[j] = mask;
81                     break;
82                 }
83             }
84         }
85     }
86     if (mask == 0) {
87         ans *= 2;
88     }
89 }
90 ans -= 1;
91 cout << ans << "\n";
92 return 0;
93 }
```

## 478: Counting Arrays

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given two positive integer numbers  $x$  and  $y$ . An array  $F$  is called an  $y$ -factorization of  $x$  iff the following conditions are met:

There are  $y$  elements in  $F$ , and all of them are integer numbers;

.

You have to count the number of pairwise distinct arrays that are  $y$ -factorizations of  $x$ . Two arrays  $A$  and  $B$  are considered different iff there exists at least one index  $i$  ( $1 \leq i \leq y$ ) such that  $A_i \neq B_i$ . Since the answer can be very large, print it modulo  $10^9 + 7$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 vector<int> minp, primes;
35 void sieve(int n) {
36     minp.assign(n + 1, 0);
37     primes.clear();
38     for (int i = 2; i <= n; i++) {
39         if (minp[i] == 0) {
40             minp[i] = i;
41             primes.push_back(i);
42         }
43         for (auto p : primes) {
44             if (i * p > n) {
45                 break;
46             }
47             minp[i * p] = p;
48         }
49     }
50 }
```

```

48         if (p == minp[i]) {
49             break;
50         }
51     }
52 }
53 # struct Comb comb;
54 int main() {
55     ios::sync_with_stdio(false);
56     cin.tie(nullptr);
57     sieve(1E6);
58     int q;
59     cin >> q;
60     for (int i = 0; i < q; i++) {
61         int x, y;
62         cin >> x >> y;
63         Z ans = power(Z(2), y - 1);
64         while (x > 1) {
65             int p = minp[x];
66             int t = 0;
67             while (x % p == 0) {
68                 x /= p;
69                 t++;
70             }
71             ans *= comb.binom(t + y - 1, y - 1);
72         }
73         cout << ans << "\n";
74     }
75     return 0;
76 }
77 }
```

### 479: Ralph and Mushrooms

- Time limit: 2.5 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Ralph is going to collect mushrooms in the Mushroom Forest.

There are  $m$  directed paths connecting  $n$  trees in the Mushroom Forest. On each path grow some mushrooms. When Ralph passes a path, he collects all the mushrooms on the path. The Mushroom Forest has a magical fertile ground where mushrooms grow at a fantastic speed. New mushrooms regrow as soon as Ralph finishes mushroom collection on a path. More specifically, after Ralph passes a path the  $i$ -th time, there regrow  $i$  mushrooms less than there was before this pass. That is, if there is initially  $x$  mushrooms on a path, then Ralph will collect  $x$  mushrooms for the first time,  $x - 1$  mushrooms the second time,  $x - 2$  mushrooms the third time, and so on. However, the number of mushrooms can never be less than 0.

For example, let there be 9 mushrooms on a path initially. The number of mushrooms that can be collected from the path is 9, 8, 6 and 3 when Ralph passes by from first to fourth time. From the fifth

time and later Ralph can't collect any mushrooms from the path (but still can pass it).

Ralph decided to start from the tree s. How many mushrooms can he collect using only described paths?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct SCC;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, m;
9     cin >> n >> m;
10    SCC g(n);
11    vector<int> x(m), y(m), w(m);
12    for (int i = 0; i < m; i++) {
13        cin >> x[i] >> y[i] >> w[i];
14        x[i]--, y[i]--;
15        g.addEdge(x[i], y[i]);
16    }
17    auto bel = g.work();
18    n = g.cnt;
19    vector<i64> dp(n), val(n);
20    vector<vector<array<int, 2>>> adj(n);
21    for (int i = 0; i < m; i++) {
22        x[i] = bel[x[i]];
23        y[i] = bel[y[i]];
24        if (x[i] == y[i]) {
25            int k = sqrt(2 * w[i]);
26            while (k * (k + 1) / 2 < w[i]) {
27                k++;
28            }
29            val[x[i]] += 1LL * w[i] * k - 1LL * (k - 1) * k * (k + 1) / 6;
30        } else {
31            adj[x[i]].push_back({y[i], w[i]});
32        }
33    }
34    int s;
35    cin >> s;
36    s--;
37    for (int i = 0; i < n; i++) {
38        for (auto [j, w] : adj[i]) {
39            dp[i] = max(dp[i], dp[j] + w);
40        }
41        dp[i] += val[i];
42    }
43    cout << dp[bel[s]] << "\n";
44    return 0;
45 }
```

## 480: Cowboys

- Time limit: 2 seconds

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A very tense moment: n cowboys stand in a circle and each one points his colt at a neighbor. Each cowboy can point the colt to the person who follows or precedes him in clockwise direction. Human life is worthless, just like in any real western.

The picture changes each second! Every second the cowboys analyse the situation and, if a pair of cowboys realize that they aim at each other, they turn around. In a second all such pairs of neighboring cowboys aiming at each other turn around. All actions happen instantaneously and simultaneously in a second.

We'll use character "A" to denote a cowboy who aims at his neighbour in the clockwise direction, and character "B" for a cowboy who aims at his neighbour in the counter clockwise direction. Then a string of letters "A" and "B" will denote the circle of cowboys, the record is made from the first of them in a clockwise direction.

For example, a circle that looks like "ABBBABBBA" after a second transforms into "BABBBABBA" and a circle that looks like "BABBA" transforms into "ABABB".

A second passed and now the cowboys' position is described by string s. Your task is to determine the number of possible states that lead to s in a second. Two states are considered distinct if there is a cowboy who aims at his clockwise neighbor in one state and at his counter clockwise neighbor in the other state.

Problem: [link](#)

Solution: [link](#)

```

1  s = input()
2  n = len(s)
3  ans = 1
4  if s == 'A' * n or s == 'B' * n :
5      print(1)
6      exit(0)
7  f = [1, 2]
8  for i in range(n) :
9      f.append(f[-1] + f[-2])
10 cnt = 0
11 for i in range(n) :
12     if s[i] == 'A' and s[(i + 1) % n] == 'B' and s[(i - 1 + n) % n] == 'A' and s[(i + 2) %
13         n] == 'B' and n >= 4:
14         print(0)
15         exit(0)
16     for i in range(n) :
17         if s[i] == 'B' and s[(i + 1) % n] == 'A' and (s[(i - 2 + n) % n] != 'B' or s[(i - 1 +
18             n) % n] != 'A') :
19             cnt += 1
20             j = i
21             while s[j % n] == 'B' and s[(j + 1) % n] == 'A' :
22                 j += 2

```

```

21         l = (j - i) // 2
22         if s[(i - 1 + n) % n] == 'A' :
23             l -= 1
24         if s[j % n] == 'B' :
25             l -= 1
26         ans *= f[max(0, l)]
27     if cnt == 0 :
28         n /= 2
29         if n == 1 :
30             print(1)
31         elif n == 2 :
32             print(2)
33         else :
34             print(f[n - 1] + f[n - 3])
35     else :
36         print(ans)

```

**481: Vanya and Brackets**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vanya is doing his maths homework. He has an expression of form , where  $x_1, x_2, \dots, x_n$  are digits from 1 to 9, and sign represents either a plus '+' or the multiplication sign '\*'. Vanya needs to add one pair of brackets in this expression so that to maximize the value of the resulting expression.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 s = input()
2 n = len(s)
3 ans = eval(s)
4 for i in range(0, n, 2) :
5     for j in range(i, n, 2) :
6         if (i == 0 or s[i - 1] == '*') and (j == n - 1 or s[j + 1] == '*') :
7             ans = max(ans, eval(s[: i] + '(' + s[i : j + 1] + ')' + s[j + 1 :]))
8 print(ans)

```

**482: Ball Sorting**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  colorful balls arranged in a row. The balls are painted in  $n$  distinct colors, denoted by numbers from 1 to  $n$ . The  $i$ -th ball from the left is painted in color  $c_i$ . You want to reorder the balls so that the  $i$ -th ball from the left has color  $i$ . Additionally, you have  $k \geq 1$  balls of color 0 that you can use in the reordering process.

Due to the strange properties of the balls, they can be reordered only by performing the following operations:

Place a ball of color 0 anywhere in the sequence (between any two consecutive balls, before the leftmost ball or after the rightmost ball) while keeping the relative order of other balls. You can perform this operation no more than  $k$  times, because you have only  $k$  balls of color 0.

Choose any ball of non-zero color such that at least one of the balls adjacent to him has color 0, and move that ball (of non-zero color) anywhere in the sequence (between any two consecutive balls, before the leftmost ball or after the rightmost ball) while keeping the relative order of other balls. You can perform this operation as many times as you want, but for each operation you should pay 1 coin.

You can perform these operations in any order. After the last operation, all balls of color 0 magically disappear, leaving a sequence of  $n$  balls of non-zero colors.

What is the minimum amount of coins you should spend on the operations of the second type, so that the  $i$ -th ball from the left has color  $i$  for all  $i$  from 1 to  $n$  after the disappearance of all balls of color zero? It can be shown that under the constraints of the problem, it is always possible to reorder the balls in the required way.

Solve the problem for all  $k$  from 1 to  $n$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> c(n);
8     for (int i = 0; i < n; i++) {
9         cin >> c[i];
10        c[i]--;
11    }
12    vector dp(2 * n + 1, vector(n, 0));
13    for (auto x : c) {
14        vector ndp(2 * n + 1, vector(n, n));
15        for (int i = 0; i <= 2 * n; i++) {
16            for (int j = 0; j < n; j++) {
17                if (i % 2 == 0) {
18                    if (j <= x) {
19                        ndp[i][x] = min(ndp[i][x], dp[i][j]);
20                    }
21                } else {

```

```

22             ndp[i][j] = min(ndp[i][j], dp[i][j] + 1);
23         }
24     }
25 }
26 dp = move(ndp);
27 for (int i = 1; i <= 2 * n; i++) {
28     for (int j = 0; j < n; j++) {
29         dp[i][j] = min(dp[i][j], dp[i - 1][j]);
30     }
31 }
32 for (int k = 1; k <= n; k++) {
33     cout << *min_element(dp[2 * k].begin(), dp[2 * k].end()) << " \n"[k == n];
34 }
35 }
36 }
37 int main() {
38     ios::sync_with_stdio(false);
39     cin.tie(nullptr);
40     int t;
41     cin >> t;
42     while (t--) {
43         solve();
44     }
45     return 0;
46 }
```

### 483: Range Sorting (Easy Version)

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The only difference between this problem and the hard version is the constraints on  $t$  and  $n$ .

You are given an array  $a$ , consisting of  $n$  distinct integers  $a_1, a_2, \dots, a_n$ .

Define the beauty of an array  $p_1, p_2, \dots, p_k$  as the minimum amount of time needed to sort this array using an arbitrary number of range-sort operations. In each range-sort operation, you will do the following:

Choose two integers  $l$  and  $r$  ( $1 \leq l < r \leq k$ ).

Sort the subarray  $p_l, p_{l+1}, \dots, p_r$  in  $r - l$  seconds.

Please calculate the sum of beauty over all subarrays of array  $a$ .

A subarray of an array is defined as a sequence of consecutive elements of the array.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template <typename T>
5 # struct Fenwick;
6 void solve() {
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    auto va = a;
14    sort(va.begin(), va.end());
15    for (auto &x : a) {
16        x = lower_bound(va.begin(), va.end(), x) - va.begin();
17    }
18    i64 ans = 0;
19    for (int i = 1; i <= n; i++) {
20        ans += 1LL * (i - 1) * (n - i + 1);
21    }
22    vector<vector<pair<int, int>>> R(n);
23    vector<int> stk{n};
24    Fenwick<int> f1(n), f2(n);
25    for (int i = n - 1; i >= 0; i--) {
26        while (stk.size() > 1 && a[i] <= a[stk.back()]) {
27            int x = stk.back();
28            stk.pop_back();
29            R[i].emplace_back(a[x], stk.back() - x);
30            f2.add(a[x], x - stk.back());
31        }
32        R[i].emplace_back(a[i], i - stk.back());
33        f2.add(a[i], stk.back() - i);
34        stk.push_back(i);
35    }
36    stk = {-1};
37    i64 res = 0;
38    for (int i = 0; i < n; i++) {
39        while (stk.size() > 1 && a[i] >= a[stk.back()]) {
40            int x = stk.back();
41            stk.pop_back();
42            res += 1LL * f2.rangeSum(a[x], n) * (stk.back() - x);
43            f1.add(a[x], stk.back() - x);
44        }
45        res += 1LL * f2.rangeSum(a[i], n) * (i - stk.back());
46        f1.add(a[i], i - stk.back());
47        stk.push_back(i);
48        for (auto [v, x] : R[i]) {
49            res += 1LL * f1.sum(v + 1) * x;
50            f2.add(v, x);
51        }
52        ans -= res;
53    }
54    cout << ans << "\n";
55 }
56 int main() {
57     ios::sync_with_stdio(false);
58     cin.tie(nullptr);
59     int t;
60     cin >> t;
61     while (t--) {
62         solve();
63     }
64     return 0;

```

65 }

**484: Let's Play Osu!**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You're playing a game called Osu! Here's a simplified version of it. There are  $n$  clicks in a game. For each click there are two outcomes: correct or bad. Let us denote correct as "O", bad as "X", then the whole play can be encoded as a sequence of  $n$  characters "O" and "X".

Using the play sequence you can calculate the score for the play as follows: for every maximal consecutive "O"s block, add the square of its length (the number of characters "O") to the score. For example, if your play can be encoded as "OOXOOOXXOO", then there's three maximal consecutive "O"s block "OO", "OOO", "OO", so your score will be  $2^2 + 3^2 + 2^2 = 17$ . If there are no correct clicks in a play then the score for the play equals to 0.

You know that the probability to click the  $i$ -th ( $1 \leq i \leq n$ ) click correctly is  $p_i$ . In other words, the  $i$ -th character in the play sequence has  $p_i$  probability to be "O",  $1 - p_i$  to be "X". Your task is to calculate the expected score for your play.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<double> p(n);
10    for (int i = 0; i < n; i++) {
11        cin >> p[i];
12    }
13    double ans = 0;
14    cout << fixed << setprecision(10);
15    double s = 0;
16    for (int i = 0; i < n; i++) {
17        s = (s + 1) * p[i];
18        ans += s * 2;
19        ans -= p[i];
20    }
21    cout << ans << "\n";

```

```

22     return 0;
23 }
```

## 485: Petya and Spiders

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Little Petya loves training spiders. Petya has a board  $n \times m$  in size. Each cell of the board initially has a spider sitting on it. After one second Petya chooses a certain action for each spider, and all of them humbly perform its commands. There are 5 possible commands: to stay idle or to move from current cell to some of the four side-neighboring cells (that is, one command for each of the four possible directions). Petya gives the commands so that no spider leaves the field. It is allowed for spiders to pass through each other when they crawl towards each other in opposite directions. All spiders crawl simultaneously and several spiders may end up in one cell. Petya wants to know the maximum possible number of spider-free cells after one second.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int inf = 1E9;
5 void chmin(int &a, int b) {
6     if (a > b) {
7         a = b;
8     }
9 }
10 int main() {
11     ios::sync_with_stdio(false);
12     cin.tie(nullptr);
13     int n, m;
14     cin >> n >> m;
15     if (n < m) {
16         swap(n, m);
17     }
18     vector dp(1 << (2 * m), 0);
19     for (int x = 0; x < n; x++) {
20         auto g = dp;
21         for (int u = 0; u < (1 << m); u++) {
22             for (int v = 0; v < (1 << m); v++) {
23                 dp[u << m | v] = g[v << m | u];
24             }
25         }
26         for (int y = 0; y < m; y++) {
27             vector g(1 << (2 * m), inf);
28             for (int s = 0; s < (1 << (2 * m)); s++) {
29                 if (s >> (m + y) & 1) {
```

```

30         chmin(g[s], dp[s ^ 1 << (m + y)]);
31     } else {
32         chmin(g[s], dp[s | 1 << y | 1 << (m + y)] + 1);
33         if (y) {
34             chmin(g[s], dp[s | 1 << (m + y - 1) | 1 << (y - 1)] + 1);
35             chmin(g[s], dp[s | 1 << (m + y - 1) | 1 << y] + 1);
36             chmin(g[s], dp[s | 1 << (m + y) | 1 << y | 1 << (y - 1)] + 1);
37         }
38         if (y < m - 1) {
39             chmin(g[s], dp[s | 1 << (m + y) | 1 << y | 1 << (y + 1)] + 1);
40         }
41         if (y && y < m - 1) {
42             chmin(g[s], dp[s | 1 << (m + y) | 1 << y | 1 << (y - 1) | 1 << (y + 1)] + 1);
43         }
44         if (y >= 2) {
45             chmin(g[s], dp[s | 1 << (m + y - 1) | 1 << (m + y - 2) | 1 << (y - 1)] + 1);
46         }
47     }
48     swap(dp, g);
49 }
50 }
51 int ans = n * m - dp[0];
52 cout << ans << "\n";
53 return 0;
54 }
```

## greedy

### 486: Freedom of Choice

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Let's define the anti-beauty of a multiset  $\{b_1, b_2, \dots, b_{len}\}$  as the number of occurrences of the number  $len$  in the multiset.

You are given  $m$  multisets, where the  $i$ -th multiset contains  $n_i$  distinct elements, specifically:  $c_{i,1}$  copies of the number  $a_{i,1}$ ,  $c_{i,2}$  copies of the number  $a_{i,2}, \dots, c_{i,n_i}$  copies of the number  $a_{i,n_i}$ . It is guaranteed that  $a_{i,1} < a_{i,2} < \dots < a_{i,n_i}$ . You are also given numbers  $l_1, l_2, \dots, l_m$  and  $r_1, r_2, \dots, r_m$  such that  $1 \leq l_i \leq r_i \leq c_{i,1} + \dots + c_{i,n_i}$ .

Let's create a multiset  $X$ , initially empty. Then, for each  $i$  from 1 to  $m$ , you must perform the following action exactly once:

Choose some  $v_i$  such that  $l_i \leq v_i \leq r_i$

Choose any  $v_i$  numbers from the  $i$ -th multiset and add them to the multiset  $X$ .

You need to choose  $v_1, \dots, v_m$  and the added numbers in such a way that the resulting multiset  $X$  has the minimum possible anti-beauty.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 1E18;
5 void solve() {
6     int m;
7     cin >> m;
8     i64 ans = inf;
9     vector<vector<pair<i64, i64>>> a(m);
10    map<i64, vector<pair<i64, i64>>> occur;
11    i64 L = 0, R = 0;
12    vector<i64> l(m), r(m), cnt(m);
13    for (int i = 0; i < m; i++) {
14        int n;
15        cin >> n >> l[i] >> r[i];
16        a[i].resize(n);
17        for (int j = 0; j < n; j++) {
18            cin >> a[i][j].first;
19        }
20        for (int j = 0; j < n; j++) {
21            cin >> a[i][j].second;
22            cnt[i] += a[i][j].second;
23            occur[a[i][j].first].emplace_back(i, a[i][j].second);
24        }
25        L += l[i], R += r[i];
26    }
27    for (i64 N = L; N <= R && ans > 0; N++) {
28        i64 suml = L, sumr = R;
29        i64 must = 0;
30        if (occur.count(N)) {
31            for (auto [i, c] : occur[N]) {
32                sumr -= r[i];
33                if (cnt[i] - c >= l[i]) {
34                    sumr += min(cnt[i] - c, r[i]);
35                } else {
36                    sumr += l[i];
37                    must += l[i] - (cnt[i] - c);
38                }
39            }
40        }
41        ans = min(ans, must + max(0LL, N - sumr));
42    }
43    cout << ans << "\n";
44 }
45 int main() {
46     ios::sync_with_stdio(false);
47     cin.tie(nullptr);
48     int t;
49     cin >> t;
50     while (t--) {
51         solve();
52     }
53     return 0;
54 }
```

## 487: Autosynthesis

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Chaneka writes down an array  $a$  of  $n$  positive integer elements. Initially, all elements are not circled. In one operation, Chaneka can circle an element. It is possible to circle the same element more than once.

After doing all operations, Chaneka makes a sequence  $r$  consisting of all uncircled elements of  $a$  following the order of their indices.

Chaneka also makes another sequence  $p$  such that its length is equal to the number of operations performed and  $p_i$  is the index of the element that is circled in the  $i$ -th operation.

Chaneka wants to do several operations such that sequence  $r$  is equal to sequence  $p$ . Help her achieve this, or report if it is impossible! Note that if there are multiple solutions, you can print any of them.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12        a[i]--;
13    }
14    vector<vector<int>> adj(n);
15    for (int i = 0; i < n; i++) {
16        adj[a[i]].push_back(i);
17    }
18    vector<int> c(n, -1);
19    for (int i = 0; i < n; i++) {
20        if (c[i] == -1) {
21            int j = i;
22            while (c[j] == -1) {
23                c[j] = 0;
24                j = a[j];
25            }
26            bool good = false;
27            for (int v = 0; v < 2; v++) {
28                bool ok = true;
29                auto dfs = [&](auto self, int x, bool root = false) -> void {
30                    if (x == j && !root) {
31                        c[x] = v;
32                    }
33                    for (int y : adj[x]) {
34                        if (c[y] == -1) {
35                            self(dfs, y, true);
36                        }
37                    }
38                };
39                dfs();
40            }
41            if (good) {
42                cout << "YES" << endl;
43                return 0;
44            }
45        }
46    }
47    cout << "NO" << endl;
48}
```

```

32             return;
33         }
34     int u = 0;
35     for (auto y : adj[x]) {
36         self(self, y);
37         u |= !c[y];
38     }
39     if (root && c[x] != u) {
40         ok = false;
41     }
42     c[x] = u;
43 };
44 dfs(dfs, j, true);
45 if (ok) {
46     good = true;
47     break;
48 }
49 if (!good) {
50     cout << -1 << "\n";
51     return 0;
52 }
53 }
54 }
55 vector<int> ans;
56 for (int i = 0; i < n; i++) {
57     if (!c[i]) {
58         ans.push_back(a[i] + 1);
59     }
60 }
61 cout << ans.size() << "\n";
62 for (int i = 0; i < ans.size(); i++) {
63     cout << ans[i] << " \n"[i == ans.size() - 1];
64 }
65 return 0;
66 }
67 }
```

## 488: Treelabeling

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Eikooc and Sushi play a game.

The game is played on a tree having  $n$  nodes numbered 1 to  $n$ . Recall that a tree having  $n$  nodes is an undirected, connected graph with  $n - 1$  edges.

They take turns alternately moving a token on the tree. Eikooc makes the first move, placing the token on any node of her choice. Sushi makes the next move, followed by Eikooc, followed by Sushi, and so on. In each turn after the first, a player must move the token to a node  $u$  such that

$u$  is adjacent to the node  $v$  the token is currently on

$u$  has not been visited before

$$u \oplus v \leq \min(u, v)$$

Here  $x \oplus y$  denotes the bitwise XOR operation on integers  $x$  and  $y$ .

Both the players play optimally. The player who is unable to make a move loses.

The following are examples which demonstrate the rules of the game.

Before the game begins, Eikooc decides to sneakily relabel the nodes of the tree in her favour. Formally, a relabeling is a permutation  $p$  of length  $n$  (sequence of  $n$  integers wherein each integer from 1 to  $n$  occurs exactly once) where  $p_i$  denotes the new numbering of node  $i$ .

She wants to maximize the number of nodes she can choose in the first turn which will guarantee her a win. Help Eikooc find any relabeling which will help her do so.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<vector<int>> adj(n);
8     for (int i = 1; i < n; i++) {
9         int u, v;
10        cin >> u >> v;
11        u--, v--;
12        adj[u].push_back(v);
13        adj[v].push_back(u);
14    }
15    vector<int> c(n);
16    auto dfs = [&](auto self, int x, int p) -> void {
17        for (auto y : adj[x]) {
18            if (y == p) {
19                continue;
20            }
21            c[y] = c[x] ^ 1;
22            self(self, y, x);
23        }
24    };
25    dfs(dfs, 0, -1);
26    int c0 = count(c.begin(), c.end(), 0);
27    int c1 = n - c0;
28    int i0 = 0, i1 = 0;
29    vector<int> p(n);
30    for (int t = __lg(n); t >= 0; t--) {
31        int v = min(1 << t, n - (1 << t) + 1);
32        if (v <= c0) {
33            c0 -= v;
34            for (int j = 0; j < v; j++) {
35                while (c[i0] != 0) {
36                    i0++;
37                }
38            }
39        }
40    }
41}
```

```

37             }
38             p[i0] = (1 << t) + j;
39             i0++;
40         }
41     } else {
42         c1 -= v;
43         for (int j = 0; j < v; j++) {
44             while (c[i1] != 1) {
45                 i1++;
46             }
47             p[i1] = (1 << t) + j;
48             i1++;
49         }
50     }
51 }
52 for (int i = 0; i < n; i++) {
53     cout << p[i] << " \n"[i == n - 1];
54 }
55 }
56 int main() {
57     ios::sync_with_stdio(false);
58     cin.tie(nullptr);
59     int t;
60     cin >> t;
61     while (t--) {
62         solve();
63     }
64     return 0;
65 }
```

### 489: Ithea Plays With Chtholly

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an interactive problem. Refer to the Interaction section below for better understanding.

Ithea and Chtholly want to play a game in order to determine who can use the kitchen tonight.

Initially, Ithea puts  $n$  clear sheets of paper in a line. They are numbered from 1 to  $n$  from left to right.

This game will go on for  $m$  rounds. In each round, Ithea will give Chtholly an integer between 1 and  $c$ , and Chtholly needs to choose one of the sheets to write down this number (if there is already a number before, she will erase the original one and replace it with the new one).

Chtholly wins if, at any time, all the sheets are filled with a number and the  $n$  numbers are in non-decreasing order looking from left to right from sheet 1 to sheet  $n$ , and if after  $m$  rounds she still doesn't win, she loses the game.

Chtholly really wants to win the game as she wants to cook something for Willem. But she doesn't know how to win the game. So Chtholly finds you, and your task is to write a program to receive numbers

that Ithea gives Chtholly and help her make the decision on which sheet of paper write this number.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m, c;
8     cin >> n >> m >> c;
9     int l = 0, r = n;
10    vector<int> a(n);
11    while (l < r) {
12        int p;
13        cin >> p;
14        if (p <= c / 2) {
15            if (l == 0 || p >= a[l - 1]) {
16                cout << l + 1 << endl;
17                a[l++] = p;
18            } else {
19                int i = 0;
20                while (a[i] <= p) {
21                    i++;
22                }
23                cout << i + 1 << endl;
24                a[i] = p;
25            }
26        } else {
27            if (r == n || p <= a[r]) {
28                cout << r << endl;
29                a[--r] = p;
30            } else {
31                int i = n - 1;
32                while (a[i] >= p) {
33                    i--;
34                }
35                cout << i + 1 << endl;
36                a[i] = p;
37            }
38        }
39    }
40    return 0;
41 }
```

## 490: Gluttony

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  with  $n$  distinct integers. Construct an array  $b$  by permuting  $a$  such that for every non-empty subset of indices  $S = \{x_1, x_2, \dots, x_k\}$  ( $1 \leq x_i \leq n, 0 < k < n$ ) the sums of elements on

that positions in  $a$  and  $b$  are different, i. e.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    vector<int> p(n);
14    iota(p.begin(), p.end(), 0);
15    sort(p.begin(), p.end(),
16          [&](int i, int j) {
17              return a[i] < a[j];
18          });
19    for (int i = 1; i < n; i++) {
20        if (a[p[i]] == a[p[i - 1]]) {
21            cout << -1 << "\n";
22            return 0;
23        }
24    }
25    vector<int> b(n);
26    for (int i = 0; i < n; i++) {
27        b[p[i]] = a[p[(i + 1) % n]];
28    }
29    for (int i = 0; i < n; i++) {
30        cout << b[i] << " \n"[i == n - 1];
31    }
32    return 0;
33 }
```

## 491: Counting Graphs

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Given a tree consisting of  $n$  vertices. A tree is a connected undirected graph without cycles. Each edge of the tree has its weight,  $w_i$ .

Your task is to count the number of different graphs that satisfy all four conditions:

The graph does not have self-loops and multiple edges.

The weights on the edges of the graph are integers and do not exceed  $S$ .

The graph has exactly one minimum spanning tree.

The minimum spanning tree of the graph is the given tree.

Two graphs are considered different if their sets of edges are different, taking into account the weights of the edges.

The answer can be large, output it modulo 998244353.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 # struct DSU;
35 void solve() {
36     int n, S;
37     cin >> n >> S;
38     vector<array<int, 3>> e(n - 1);
39     for (int i = 0; i < n - 1; i++) {
40         int u, v, w;
41         cin >> u >> v >> w;
42         u--, v--;
43         e[i] = {w, u, v};
44     }
45     sort(e.begin(), e.end());
46     DSU dsu(n);
47     Z ans = 1;
48     for (auto [w, u, v] : e) {
49         i64 cnt = 1LL * dsu.size(u) * dsu.size(v) - 1;

```

```

50         ans *= power(Z(S - w + 1), cnt);
51         dsu.merge(u, v);
52     }
53     cout << ans << "\n";
54 }
55 int main() {
56     ios::sync_with_stdio(false);
57     cin.tie(nullptr);
58     int t;
59     cin >> t;
60     while (t--) {
61         solve();
62     }
63     return 0;
64 }
```

## 492: Pairs of Segments

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Two segments  $[l_1, r_1]$  and  $[l_2, r_2]$  intersect if there exists at least one  $x$  such that  $l_1 \leq x \leq r_1$  and  $l_2 \leq x \leq r_2$ .

An array of segments  $[[l_1, r_1], [l_2, r_2], \dots, [l_k, r_k]]$  is called beautiful if  $k$  is even, and it is possible to split the elements of this array into  $\frac{k}{2}$  pairs in such a way that:

every element of the array belongs to exactly one of the pairs;

segments in each pair intersect with each other;

segments in different pairs do not intersect.

For example, the array  $[[2, 4], [9, 12], [2, 4], [7, 7], [10, 13], [6, 8]]$  is beautiful, since it is possible to form 3 pairs as follows:

the first element of the array (segment  $[2, 4]$ ) and the third element of the array (segment  $[2, 4]$ );

the second element of the array (segment  $[9, 12]$ ) and the fifth element of the array (segment  $[10, 13]$ );

the fourth element of the array (segment  $[7, 7]$ ) and the sixth element of the array (segment  $[6, 8]$ ).

As you can see, the segments in each pair intersect, and no segments from different pairs intersect.

You are given an array of  $n$  segments  $[[l_1, r_1], [l_2, r_2], \dots, [l_n, r_n]]$ . You have to remove the minimum possible number of elements from this array so that the resulting array is beautiful.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> l(n), r(n);
8     for (int i = 0; i < n; i++) {
9         cin >> l[i] >> r[i];
10    }
11    vector<int> o(n);
12    iota(o.begin(), o.end(), 0);
13    sort(o.begin(), o.end(),
14        [&](int i, int j) {
15            return r[i] < r[j];
16        });
17    int ans = 0;
18    int cut = -1;
19    int lastr = -1;
20    for (auto i : o) {
21        if (l[i] <= cut) {
22            continue;
23        }
24        if (l[i] <= lastr) {
25            ans++;
26            cut = r[i];
27        } else {
28            lastr = r[i];
29        }
30    }
31    cout << n - 2 * ans << "\n";
32 }
33 int main() {
34     ios::sync_with_stdio(false);
35     cin.tie(nullptr);
36     int t;
37     cin >> t;
38     while (t--) {
39         solve();
40     }
41     return 0;
42 }
```

### 493: Gadgets for dollars and pounds

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Nura wants to buy  $k$  gadgets. She has only  $s$  burles for that. She can buy each gadget for dollars or for pounds. So each gadget is selling only for some type of currency. The type of currency and the cost in that currency are not changing.

Nura can buy gadgets for  $n$  days. For each day you know the exchange rates of dollar and pound, so you know the cost of conversion burles to dollars or to pounds.

Each day (from 1 to  $n$ ) Nura can buy some gadgets by current exchange rate. Each day she can buy any gadgets she wants, but each gadget can be bought no more than once during  $n$  days.

Help Nura to find the minimum day index when she will have  $k$  gadgets. Nura always pays with burles, which are converted according to the exchange rate of the purchase day. Nura can't buy dollars or pounds, she always stores only burles. Gadgets are numbered with integers from 1 to  $m$  in order of their appearing in input.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m, k, s;
8     cin >> n >> m >> k >> s;
9     vector<int> a(n), b(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    for (int i = 0; i < n; i++) {
14        cin >> b[i];
15    }
16    vector<int> t(m), c(m);
17    for (int i = 0; i < m; i++) {
18        cin >> t[i] >> c[i];
19    }
20    auto mina = a, minb = b;
21    for (int i = 1; i < n; i++) {
22        mina[i] = min(mina[i], mina[i-1]);
23        minb[i] = min(minb[i], minb[i-1]);
24    }
25    auto check = [&](int x) {
26        vector<i64> d(m);
27        for (int i = 0; i < m; i++) {
28            d[i] = 1LL * c[i] * (t[i] == 1 ? mina[x-1] : minb[x-1]);
29        }
30        nth_element(d.begin(), d.begin() + k, d.end());
31        i64 sum = accumulate(d.begin(), d.begin() + k, 0LL);
32        return sum <= s;
33    };
34    int lo = 1, hi = n+1;
35    while (lo < hi) {
36        int x = (lo + hi) / 2;
37        if (check(x)) {
38            hi = x;
39        } else {
40            lo = x+1;
41        }
42    }
43    if (lo == n+1) {
44        cout << -1 << "\n";
45        return 0;

```

```

46      }
47      vector<i64> d(m);
48      for (int i = 0; i < m; i++) {
49          d[i] = 1LL * c[i] * (t[i] == 1 ? mina[lo-1] : minb[lo-1]);
50      }
51      vector<int> o(m);
52      iota(o.begin(), o.end(), 0);
53      nth_element(o.begin(), o.begin() + k, o.end(), [&](int i, int j) {
54          return d[i] < d[j];
55      });
56      int daya = min_element(a.begin(), a.begin() + lo) - a.begin() + 1;
57      int dayb = min_element(b.begin(), b.begin() + lo) - b.begin() + 1;
58      cout << lo << "\n";
59      for (int i = 0; i < k; i++) {
60          cout << o[i]+1 << " " << (t[o[i]] == 1 ? daya : dayb) << "\n";
61      }
62      return 0;
63  }

```

#### 494: Rearrange Brackets

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A regular bracket sequence is a bracket sequence that can be transformed into a correct arithmetic expression by inserting characters “1” and “+” between the original characters of the sequence. For example:

bracket sequences “()()” and “((())” are regular (the resulting expressions are: “(1)+(1)” and “((1+1)+1)”);

bracket sequences “)”, “(” and “)” are not.

You are given a regular bracket sequence. In one move, you can remove a pair of adjacent brackets such that the left one is an opening bracket and the right one is a closing bracket. Then concatenate the resulting parts without changing the order. The cost of this move is the number of brackets to the right of the right bracket of this pair.

The cost of the regular bracket sequence is the smallest total cost of the moves required to make the sequence empty.

Actually, you are not removing any brackets. Instead, you are given a regular bracket sequence and an integer  $k$ . You can perform the following operation at most  $k$  times:

extract some bracket from the sequence and insert it back at any position (between any two brackets, at the start or at the end; possibly, at the same place it was before).

After all operations are performed, the bracket sequence has to be regular. What is the smallest possible cost of the resulting regular bracket sequence?

## Problem: [link](#)

## Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void chmin(i64 &a, i64 b) {
5     if (a > b) {
6         a = b;
7     }
8 }
9 void solve() {
10     int k;
11     cin >> k;
12     string s;
13     cin >> s;
14     int n = s.size() / 2;
15     vector<int> pre(2 * n + 1);
16     for (int i = 0; i < 2 * n; i++) {
17         pre[i + 1] = pre[i] + (s[i] == '(' ? 1 : -1);
18     }
19     i64 dp[6][6][6][6] {};
20     memset(dp, 0x3f, sizeof(dp));
21     dp[0][0][0][0] = 0;
22     for (int i = 0; i <= 2 * n; i++) {
23         if (i) {
24             for (int a = k; a >= 0; a--) {
25                 for (int b = k; b >= 0; b--) {
26                     for (int c = k - max(a, b); c >= 0; c--) {
27                         for (int d = k - max(a, b); d >= 0; d--) {
28                             if (b < k && pre[i] - a + (b + 1) + c - d >= 0) {
29                                 chmin(dp[a][b][c][d], dp[a][b][c][d] + pre[i] - a + (b
30                                         + 1) + c - d);
31                             }
32                             if (d < k && pre[i] - a + b + c - (d + 1) >= 0) {
33                                 chmin(dp[a][b][c][d + 1], dp[a][b][c][d] + pre[i] - a + b
34                                         + c - (d + 1));
35                             }
36                             if (pre[i] - a + b + c - d < 0) {
37                                 dp[a][b][c][d] = 0x3f3f3f3f3f3f3f3f;
38                             } else {
39                                 dp[a][b][c][d] += pre[i] - a + b + c - d;
40                             }
41                         }
42                     }
43                 }
44             }
45             for (int a = 0; a <= k; a++) {
46                 for (int b = 0; b <= k; b++) {
47                     for (int c = 0; c <= k - max(a, b); c++) {
48                         for (int d = 0; d <= k - max(a, b); d++) {
49                             if (a < k && pre[i] - (a + 1) + b + c - d >= 0) {
50                                 chmin(dp[a + 1][b][c][d], dp[a][b][c][d] + pre[i] - (a + 1) +
51                                         b + c - d);
52                             }
53                             if (c < k && pre[i] - a + b + (c + 1) - d >= 0) {
54                                 dp[a][b][c + 1][d] = 0x3f3f3f3f3f3f3f3f;
55                             } else {
56                                 dp[a][b][c + 1][d] += pre[i] - a + b + (c + 1) - d;
57                             }
58                         }
59                     }
60                 }
61             }
62         }
63     }
64 }

```

```

52                     chmin(dp[a][b][c + 1][d], dp[a][b][c][d] + pre[i] - a + b + (c
53                         + 1) - d);
54                 }
55             }
56         }
57     }
58     i64 ans = 0x3f3f3f3f3f3f3f3f;
59     for (int a = 0; a <= k; a++) {
60         for (int b = 0; a + b <= k; b++) {
61             ans = min(ans, dp[a][a][b]);
62         }
63     }
64     ans = (ans - n) / 2;
65     cout << ans << "\n";
66 }
67 int main() {
68     ios::sync_with_stdio(false);
69     cin.tie(nullptr);
70     int t;
71     cin >> t;
72     while (t--) {
73         solve();
74     }
75     return 0;
76 }
77 }
```

## 495: C\*++ Calculations

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

C++ language is quite similar to C++. The similarity manifests itself in the fact that the programs written in C++ sometimes behave unpredictably and lead to absolutely unexpected effects. For example, let's imagine an arithmetic expression in C\*++ that looks like this (expression is the main term):

expression ::= summand | expression + summand | expression - summand

summand ::= increment | coefficient\*increment

increment ::= a++ | ++a

coefficient ::= 0|1|2|...|1000

For example, “5a++-3++a+a++” is a valid expression in C\*++.

Thus, we have a sum consisting of several summands divided by signs “+” or “-”. Every summand is an expression “a++” or “++a” multiplied by some integer coefficient. If the coefficient is omitted, it is suggested being equal to 1.

The calculation of such sum in C<sup>++</sup> goes the following way. First all the summands are calculated one after another, then they are summed by the usual arithmetic rules. If the summand contains “a++”, then during the calculation first the value of the “a” variable is multiplied by the coefficient, then value of “a” is increased by 1. If the summand contains “++a”, then the actions on it are performed in the reverse order: first “a” is increased by 1, then - multiplied by the coefficient.

The summands may be calculated in any order, that's why sometimes the result of the calculation is completely unpredictable! Your task is to find its largest possible value.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1  a = int(input())
2  s = '+'+input()
3  t = []
4  p = 0
5  n = len(s)
6  while p < n :
7      pa = s.find('a', p)
8      if pa >= p + 3 and s[pa - 2 : pa] == '++' :
9          np = pa + 1
10     else :
11         np = pa + 3
12     t.append(s[p : np])
13     p = np
14  coef = []
15  for s in t :
16      if s[-1] == 'a' :
17          pre = True
18      else :
19          pre = False
20      v = 1
21      if len(s) > 4 :
22          v = int(s[:-4])
23      elif s[0] == '-' :
24          v = -1
25      coef.append((v, pre))
26  coef.sort()
27  ans = 0
28  for (v, pre) in coef :
29      if pre :
30          a += 1
31      ans += v * a
32      if not pre :
33          a += 1
34  print(ans)

```

## 496: Parquet

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

Once Bob decided to lay a parquet floor in his living room. The living room is of size  $n \times m$  metres. Bob had planks of three types: a planks 1 2 meters, b planks 2 1 meters, and c planks 2 2 meters. Help Bob find out, if it is possible to parquet the living room with such a set of planks, and if it is possible, find one of the possible ways to do so. Bob doesn't have to use all the planks.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 char get(int x, int y) {
5     return 'a' + x % 5 * 5 + y % 5;
6 }
7 int main() {
8     ios::sync_with_stdio(false);
9     cin.tie(nullptr);
10    int n, m, a, b, c;
11    cin >> n >> m >> a >> b >> c;
12    if (n * m % 2 == 1) {
13        cout << "IMPOSSIBLE\n";
14        return 0;
15    }
16    int mina = n % 2 * m / 2;
17    int minb = m % 2 * n / 2;
18    if (a < mina || b < minb) {
19        cout << "IMPOSSIBLE\n";
20        return 0;
21    }
22    if ((a - mina) % 2) {
23        a -= 1;
24    }
25    if ((b - minb) % 2) {
26        b -= 1;
27    }
28    if (2 * a + 2 * b + 4 * c < n * m) {
29        cout << "IMPOSSIBLE\n";
30        return 0;
31    }
32    vector s(n, string(m, '.'));
33    if (n % 2) {
34        for (int j = 0; j < m; j += 2) {
35            s[n - 1][j] = get(n - 1, j);
36            s[n - 1][j + 1] = get(n - 1, j);
37            a -= 1;
38        }
39    }
40    if (m % 2) {
41        for (int i = 0; i < n; i += 2) {
42            s[i][m - 1] = get(i, m - 1);
43            s[i + 1][m - 1] = get(i, m - 1);
44            b -= 1;
45        }
46    }
47    for (int i = 0; i + 1 < n; i += 2) {
48        for (int j = 0; j + 1 < m; j += 2) {

```

```

49         if (c) {
50             s[i][j] = get(i, j);
51             s[i][j + 1] = get(i, j);
52             s[i + 1][j] = get(i, j);
53             s[i + 1][j + 1] = get(i, j);
54             c -= 1;
55         } else if (a) {
56             s[i][j] = get(i, j);
57             s[i][j + 1] = get(i, j);
58             s[i + 1][j] = get(i + 1, j);
59             s[i + 1][j + 1] = get(i + 1, j);
60             a -= 2;
61         } else {
62             s[i][j] = get(i, j);
63             s[i][j + 1] = get(i, j + 1);
64             s[i + 1][j] = get(i, j);
65             s[i + 1][j + 1] = get(i, j + 1);
66             b -= 2;
67         }
68     }
69 }
70 for (int i = 0; i < n; i++) {
71     cout << s[i] << "\n";
72 }
73 return 0;
74 }
```

**497: Seller Bob**

- Time limit: 2 seconds
- Memory limit: 128 megabytes
- Input file: standard input
- Output file: standard output

Last year Bob earned by selling memory sticks. During each of  $n$  days of his work one of the two following events took place:

A customer came to Bob and asked to sell him a  $2x$  MB memory stick. If Bob had such a stick, he sold it and got  $2x$  berllars.

Bob won some programming competition and got a  $2x$  MB memory stick as a prize. Bob could choose whether to present this memory stick to one of his friends, or keep it.

Bob never kept more than one memory stick, as he feared to mix up their capacities, and deceive a customer unintentionally. It is also known that for each memory stick capacity there was at most one customer, who wanted to buy that memory stick. Now, knowing all the customers' demands and all the prizes won at programming competitions during the last  $n$  days, Bob wants to know, how much money he could have earned, if he had acted optimally.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 n = int(input())
2 dp = [0] * (n+1)
3 last = [-1] * 2001
4 for i in range(0,n) :
5     [s, x] = input().split(' ')
6     x = int(x)
7     dp[i + 1] = dp[i]
8     if s == 'win' :
9         last[x] = i
10    elif last[x] != -1 :
11        dp[i + 1] = max(dp[i + 1], dp[last[x]] + 2 ** x)
12 print(dp[n]);

```

## 498: Monsters

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There is an undirected graph with  $n$  vertices and  $m$  edges. Initially, for each vertex  $i$ , there is a monster with danger  $a_i$  on that vertex. For a monster with danger  $a_i$ , you can defeat it if and only if you have defeated at least  $a_i$  other monsters before.

Now you want to defeat all the monsters. First, you choose some vertex  $s$  and defeat the monster on that vertex (since you haven't defeated any monsters before,  $a_s$  has to be 0). Then, you can move through the edges. If you want to move from vertex  $u$  to vertex  $v$ , then the following must hold: either the monster on vertex  $v$  has been defeated before, or you can defeat it now. For the second case, you defeat the monster on vertex  $v$  and reach vertex  $v$ .

You can pass the vertices and the edges any number of times. Determine whether you can defeat all the monsters or not.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 void solve() {
6     int n, m;
7     cin >> n >> m;
8     vector<int> a(n);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11     }
12     vector<vector<int>> adj(n);

```

```

13     for (int i = 0; i < m; i++) {
14         int u, v;
15         cin >> u >> v;
16         u--, v--;
17         if (pair(a[u], u) < pair(a[v], v)) {
18             swap(u, v);
19         }
20         adj[u].push_back(v);
21     }
22     vector<bool> ok(n);
23     vector<int> order(n);
24     iota(order.begin(), order.end(), 0);
25     sort(order.begin(), order.end(), [&](int i, int j) {
26         return pair(a[i], i) < pair(a[j], j);
27     });
28     DSU dsu(n);
29     for (auto x : order) {
30         ok[x] = a[x] == 0;
31         for (auto y : adj[x]) {
32             y = dsu.leader(y);
33             if (ok[y] && dsu.size(y) >= a[x]) {
34                 ok[x] = true;
35             }
36             dsu.merge(x, y);
37         }
38     }
39     for (int i = 0; i < n; i++) {
40         if (!dsu.same(0, i)) {
41             cout << "NO\n";
42             return;
43         }
44     }
45     if (ok[dsu.leader(0)]) {
46         cout << "YES\n";
47     } else {
48         cout << "NO\n";
49     }
50 }
51 int main() {
52     ios::sync_with_stdio(false);
53     cin.tie(nullptr);
54     int t;
55     cin >> t;
56     while (t--) {
57         solve();
58     }
59     return 0;
60 }
```

## 499: Accommodation

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Annie is an amateur photographer. She likes to take pictures of giant residential buildings at night. She just took a picture of a huge rectangular building that can be seen as a table of  $n \times m$  windows. That

means that the building has  $n$  floors and each floor has exactly  $m$  windows. Each window is either dark or bright, meaning there is light turned on in the room behind it.

Annies knows that each apartment in this building is either one-bedroom or two-bedroom. Each one-bedroom apartment has exactly one window representing it on the picture, and each two-bedroom apartment has exactly two consecutive windows on the same floor. Moreover, the value of  $m$  is guaranteed to be divisible by 4 and it is known that each floor has exactly  $\frac{m}{4}$  two-bedroom apartments and exactly  $\frac{m}{2}$  one-bedroom apartments. The actual layout of apartments is unknown and can be different for each floor.

Annie considers an apartment to be occupied if at least one of its windows is bright. She now wonders, what are the minimum and maximum possible number of occupied apartments if judged by the given picture?

Formally, for each of the floors, she comes up with some particular apartments layout with exactly  $\frac{m}{4}$  two-bedroom apartments (two consecutive windows) and  $\frac{m}{2}$  one-bedroom apartments (single window). She then counts the total number of apartments that have at least one bright window. What is the minimum and maximum possible number she can get?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     int smin = 0, smax = 0;
10    for (int i = 0; i < n; i++) {
11        string s;
12        cin >> s;
13        int min = count(s.begin(), s.end(), '1');
14        int two = 0;
15        for (int j = 0; j < m - 1; j++) {
16            if (s[j] == '1' && s[j + 1] == '1') {
17                j++;
18                two++;
19            }
20        }
21        two = min(two, m / 4);
22        smin += min - two;
23        int max = count(s.begin(), s.end(), '1');
24        two = 0;
25        for (int j = 0; j < m - 1; j++) {
26            if (s[j] != '1' || s[j + 1] != '1') {
27                j++;
28                two++;
29            }
30        }
31        two = min(two, m / 4);
32        smax += max - (m / 4 - two);

```

```

33     }
34     cout << smin << " " << smax << "\n";
35     return 0;
36 }
```

## 500: The way home

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

He's going to get there by plane. In total, there are  $m$  flights in the country,  $i$ -th flies from city  $a_i$  to city  $b_i$  and costs  $s_i$  coins. Note that the  $i$ -th flight is one-way, so it can't be used to get from city  $b_i$  to city  $a_i$ . To use it, Borya must be in the city  $a_i$  and have at least  $s_i$  coins (which he will spend on the flight).

After the robbery, he has only  $p$  coins left, but he does not despair! Being in the city  $i$ , he can organize performances every day, each performance will bring him  $w_i$  coins.

Help the magician find out if he will be able to get home, and what is the minimum number of performances he will have to organize.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 1E18;
5 void solve() {
6     int n, m, p;
7     cin >> n >> m >> p;
8     vector<int> w(n);
9     for (int i = 0; i < n; i++) {
10         cin >> w[i];
11     }
12     vector<vector<pair<int, int>>> adj(n);
13     for (int i = 0; i < m; i++) {
14         int u, v, w;
15         cin >> u >> v >> w;
16         u--;
17         v--;
18         adj[u].emplace_back(v, w);
19     }
20     vector<int> order(n);
21     iota(order.begin(), order.end(), 0);
22     sort(order.begin(), order.end(), [&](int i, int j) {
23         return w[i] < w[j];
24     });
25     vector<array<i64, 2>> dp(n, {inf, inf});
26     dp[0] = {0, -p};
27     for (auto x : order) {
28         vector<i64> dis(n, -1);

```

```

29         h.emplace(0, x);
30         while (!h.empty()) {
31             auto [d, x] = h.top();
32             h.pop();
33             if (dis[x] != -1) {
34                 continue;
35             }
36             dis[x] = -d;
37             for (auto [y, w] : adj[x]) {
38                 h.emplace(d - w, y);
39             }
40         }
41         for (int y = 0; y < n; y++) {
42             if (dis[y] != -1) {
43                 i64 t = max(0LL, (dis[y] + dp[x][1] + w[x] - 1) / w[x]);
44                 dp[y] = min(dp[y], array{dp[x][0] + t, dp[x][1] - t * w[x] + dis[y]});
45             }
46         }
47         i64 ans = dp[n - 1][0];
48         if (ans == inf) {
49             ans = -1;
50         }
51         cout << ans << "\n";
52     }
53 int main() {
54     ios::sync_with_stdio(false);
55     cin.tie(nullptr);
56     int t;
57     cin >> t;
58     while (t--) {
59         solve();
60     }
61     return 0;
62 }
63 }
```

## 501: Maximum Subarray

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a_1, a_2, \dots, a_n$ , consisting of  $n$  integers. You are also given two integers  $k$  and  $x$ .

You have to perform the following operation exactly once: add  $x$  to the elements on exactly  $k$  distinct positions, and subtract  $x$  from all the others.

For example, if  $a = [2, -1, 2, 3]$ ,  $k = 1$ ,  $x = 2$ , and we have picked the first element, then after the operation the array  $a = [4, -3, 0, 1]$ .

Let  $f(a)$  be the maximum possible sum of a subarray of  $a$ . The subarray of  $a$  is a contiguous part of the array  $a$ , i. e. the array  $a_i, a_{i+1}, \dots, a_j$  for some  $1 \leq i \leq j \leq n$ . An empty subarray should also be

considered, it has sum 0.

Let the array  $a'$  be the array  $a$  after applying the aforementioned operation. Apply the operation in such a way that  $f(a')$  is the maximum possible, and print the maximum possible value of  $f(a')$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 1E18;
5 void solve() {
6     int n, k, x;
7     cin >> n >> k >> x;
8     vector<int> a(n);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11     }
12     vector dp(k + 1, array<i64, 3>{-inf, -inf, -inf});
13     dp[0] = {};
14     for (int i = 0; i < n; i++) {
15         vector g(k + 1, array<i64, 3>{-inf, -inf, -inf});
16         for (int j = 0; j <= k; j++) {
17             for (int t = 0; t < 3; t++) {
18                 g[j][t] = max(g[j][t], dp[j][t] + (t == 1 ? a[i] - x : 0));
19                 if (j < k) {
20                     g[j + 1][t] = max(g[j + 1][t], dp[j][t] + (t == 1 ? a[i] + x : 0));
21                 }
22             }
23         }
24         for (int j = 0; j <= k; j++) {
25             for (int t = 1; t < 3; t++) {
26                 g[j][t] = max(g[j][t], g[j][t - 1]);
27             }
28         }
29         dp = g;
30     }
31     auto ans = dp[k][2];
32     cout << ans << "\n";
33 }
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int t;
38     cin >> t;
39     while (t--) {
40         solve();
41     }
42     return 0;
43 }
```

## 502: Another Array Problem

- Time limit: 2 seconds

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  of  $n$  integers. You are allowed to perform the following operation on it as many times as you want (0 or more times):

Choose 2 indices  $i, j$  where  $1 \leq i < j \leq n$  and replace  $a_k$  for all  $i \leq k \leq j$  with  $|a_i - a_j|$

Print the maximum sum of all the elements of the final array that you can obtain in such a way.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    int max = *max_element(a.begin(), a.end());
12    i64 ans;
13    if (n == 2) {
14        ans = max(a[0] + a[1], 2 * abs(a[0] - a[1]));
15    } else if (n == 3) {
16        ans = 3LL * max({a[0], a[2], abs(a[0] - a[1]), abs(a[1] - a[2])});
17        ans = max(ans, 1LL * a[0] + a[1] + a[2]);
18    } else {
19        ans = 1LL * n * max;
20    }
21    cout << ans << "\n";
22 }
23 int main() {
24     ios::sync_with_stdio(false);
25     cin.tie(nullptr);
26     int t;
27     cin >> t;
28     while (t--) {
29         solve();
30     }
31     return 0;
32 }
```

### 503: The Harmonization of XOR

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array of exactly  $n$  numbers  $[1, 2, 3, \dots, n]$  along with integers  $k$  and  $x$ .

Partition the array in exactly  $k$  non-empty disjoint subsequences such that the bitwise XOR of all numbers in each subsequence is  $x$ , and each number is in exactly one subsequence. Notice that there are no constraints on the length of each subsequence.

A sequence  $a$  is a subsequence of a sequence  $b$  if  $a$  can be obtained from  $b$  by the deletion of several (possibly, zero or all) elements.

For example, for  $n = 15$ ,  $k = 6$ ,  $x = 7$ , the following scheme is valid:

$$[6, 10, 11], 6 \oplus 10 \oplus 11 = 7,$$

$$[5, 12, 14], 5 \oplus 12 \oplus 14 = 7,$$

$$[3, 9, 13], 3 \oplus 9 \oplus 13 = 7,$$

$$[1, 2, 4], 1 \oplus 2 \oplus 4 = 7,$$

$$[8, 15], 8 \oplus 15 = 7,$$

$$[7], 7 = 7,$$

The following scheme is invalid, since 8, 15 do not appear:

$$[6, 10, 11], 6 \oplus 10 \oplus 11 = 7,$$

$$[5, 12, 14], 5 \oplus 12 \oplus 14 = 7,$$

$$[3, 9, 13], 3 \oplus 9 \oplus 13 = 7,$$

$$[1, 2, 4], 1 \oplus 2 \oplus 4 = 7,$$

$$[7], 7 = 7.$$

The following scheme is invalid, since 3 appears twice, and 1, 2 do not appear:

$$[6, 10, 11], 6 \oplus 10 \oplus 11 = 7,$$

$$[5, 12, 14], 5 \oplus 12 \oplus 14 = 7,$$

$$[3, 9, 13], 3 \oplus 9 \oplus 13 = 7,$$

$$[3, 4], 3 \oplus 4 = 7,$$

$$[8, 15], 8 \oplus 15 = 7,$$

$$[7], 7 = 7.$$

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k, x;
6     cin >> n >> k >> x;
7     int xors = 0;
8     if (n % 4 == 0) xors = n;
9     else if (n % 4 == 1) xors = 1;
10    else if (n % 4 == 2) xors = n + 1;
11    else xors = 0;
12    if (xors != 0 && x != xors) {
13        cout << "NO\n";
14        return;
15    }
16    if (xors != 0 && k % 2 == 0) {
17        cout << "NO\n";
18        return;
19    }
20    if (xors == 0 && k % 2 == 1) {
21        cout << "NO\n";
22        return;
23    }
24    vector<vector<int>> a;
25    vector<bool> vis(n + 1);
26    if (n % 4 == 1) {
27        for (int i = 0; i < n; i += 2) {
28            if (i == 0) {
29                a.push_back({1});
30            } else {
31                a.push_back({i, i + 1});
32            }
33        }
34    } else if (n % 4 == 0 || n % 4 == 2) {
35        int l = __lg(n);
36        for (int i = (1 << l); i <= n; i++) {
37            if (i == x) {
38                vis[i] = true;
39                a.push_back({i});
40            } else {
41                vis[i] = vis[i ^ x] = true;
42                a.push_back({i, i ^ x});
43            }
44        }
45        for (int i = 1; i <= n; i++) {
46            if (!vis[i]) {
47                a[0].push_back(i);
48            }
49        }
50    } else {
51        for (int i = 1; i <= n; i++) {
52            if (vis[i]) continue;
53            if ((i ^ x) > n) continue;
54            if (i == x) {
55                vis[i] = true;
56                a.push_back({i});
57            } else {
58                vis[i] = vis[i ^ x] = true;
59                a.push_back({i, i ^ x});
60            }
61        }
62    }
63    if (a.empty()) {
64        cout << "NO\n";
65        return;
66    }
67}

```

```

65         }
66     for (int i = 1; i <= n; i++) {
67         if (!vis[i]) {
68             a[0].push_back(i);
69         }
70     }
71     if (k > a.size()) {
72         cout << "NO\n";
73         return;
74     }
75     cout << "YES\n";
76     for (int i = k; i < a.size(); i++) {
77         a[0].insert(a[0].end(), a[i].begin(), a[i].end());
78     }
79     a.resize(k);
80     for (auto v : a) {
81         cout << v.size();
82         for (auto x : v) {
83             cout << " " << x;
84         }
85     }
86     cout << "\n";
87 }
88 int main() {
89     ios::sync_with_stdio(false);
90     cin.tie(nullptr);
91     int t;
92     cin >> t;
93     while (t--) {
94         solve();
95     }
96     return 0;
97 }
```

## 504: Timofey and Black-White Tree

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Timofey came to a famous summer school and found a tree on  $n$  vertices. A tree is a connected undirected graph without cycles.

Every vertex of this tree, except  $c_0$ , is colored white. The vertex  $c_0$  is colored black.

Timofey wants to color all the vertices of this tree in black. To do this, he performs  $n - 1$  operations. During the  $i$ -th operation, he selects the vertex  $c_i$ , which is currently white, and paints it black.

Let's call the positivity of tree the minimum distance between all pairs of different black vertices in it. The distance between the vertices  $v$  and  $u$  is the number of edges on the path from  $v$  to  $u$ .

After each operation, Timofey wants to know the positivity of the current tree.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> c(n);
8     for (int i = 0; i < n; i++) {
9         cin >> c[i];
10        c[i]--;
11    }
12    vector<vector<int>> adj(n);
13    for (int i = 1; i < n; i++) {
14        int u, v;
15        cin >> u >> v;
16        u--, v--;
17        adj[u].push_back(v);
18        adj[v].push_back(u);
19    }
20    vector<int> dis(n, n);
21    int ans = n;
22    for (int i = 0; i < n; i++) {
23        ans = min(ans, dis[c[i]]);
24        if (i > 0) {
25            cout << ans << " \n"[i == n - 1];
26        }
27        queue<int> q;
28        q.push(c[i]);
29        dis[c[i]] = 0;
30        while (!q.empty()) {
31            int x = q.front();
32            q.pop();
33            for (auto y : adj[x]) {
34                if (dis[x] + 1 < dis[y] && dis[x] + 1 < ans) {
35                    dis[y] = dis[x] + 1;
36                    q.push(y);
37                }
38            }
39        }
40    }
41 }
42 int main() {
43     ios::sync_with_stdio(false);
44     cin.tie(nullptr);
45     int t;
46     cin >> t;
47     while (t--) {
48         solve();
49     }
50     return 0;
51 }
```

## 505: The Human Equation

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Petya and his friend, the robot Petya++, went to BFDMONCON, where the costume contest is taking place today.

While walking through the festival, they came across a scientific stand named after Professor Oak and Golfball, where they were asked to solve an interesting problem.

Given a sequence of numbers  $a_1, a_2, \dots, a_n$  you can perform several operations on this sequence.

Each operation should look as follows. You choose some subsequence<sup>†</sup>. Then you call all the numbers at odd positions in this subsequence northern, and all the numbers at even positions in this subsequence southern. In this case, only the position of the number in the subsequence is taken into account, not in the original sequence.

For example, consider the sequence 1, 4, 2, 8, 5, 7, 3, 6, 9 and its subsequence (shown in bold) 1, **4**, **2**, 8, **5**, 7, 3, **6**, 9. Then the numbers 4 and 5 are northern, and the numbers 2 and 6 are southern.

After that, you can do one of the following:

add 1 to all northern numbers and subtract 1 from all south numbers; or

add 1 to all southern numbers and subtract 1 from all northern numbers.

Thus, from the sequence 1, **4**, **2**, 8, **5**, 7, 3, **6**, 9, if you choose the subsequence shown in bold, you can get either 1, **5**, **1**, 8, **6**, 7, 3, **5**, 9 or 1, **3**, **3**, 8, **4**, 7, 3, **7**, 9.

Then the operation ends. Note also that all operations are independent, i. e. the numbers are no longer called northern or southern when one operation ends.

It is necessary to turn all the numbers of the sequence into zeros using the operations described above. Since there is very little time left before the costume contest, the friends want to know, what is the minimum number of operations required for this.

The friends were unable to solve this problem, so can you help them?

<sup>†</sup> A sequence  $c$  is a subsequence of a sequence  $d$  if  $c$  can be obtained from  $d$  by the deletion of several (possibly, zero or all) elements.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<i64> s(n + 1);
12    for (int i = 0; i < n; i++) {
13        s[i + 1] = s[i] + a[i];
14    }
15    auto ans = *max_element(s.begin(), s.end()) - *min_element(s.begin(), s.end());
16    cout << ans << "\n";
17 }
18 int main() {
19     ios::sync_with_stdio(false);
20     cin.tie(nullptr);
21     int t;
22     cin >> t;
23     while (t--) {
24         solve();
25     }
26     return 0;
27 }
```

## implementation

### 506: Is It Flower?

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vlad found a flowerbed with graphs in his yard and decided to take one for himself. Later he found out that in addition to the usual graphs,  $k$ -flowers also grew on that flowerbed. A graph is called a  $k$ -flower if it consists of a simple cycle of length  $k$ , through each vertex of which passes its own simple cycle of length  $k$  and these cycles do not intersect at the vertices. For example, 3-flower looks like this:

Note that 1-flower and 2-flower do not exist, since at least 3 vertices are needed to form a cycle.

Vlad really liked the structure of the  $k$ -flowers and now he wants to find out if he was lucky to take one of them from the flowerbed.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 void solve() {
6     int n, m;
7     cin >> n >> m;
8     vector<vector<int>> adj(n);
9     for (int i = 0; i < m; i++) {
10         int u, v;
11         cin >> u >> v;
12         u--;
13         v--;
14         adj[u].push_back(v);
15         adj[v].push_back(u);
16     }
17     int k = sqrt(n);
18     if (k * k != n || m != k * (k + 1)) {
19         cout << "NO\n";
20         return;
21     }
22     int x = -1;
23     for (int i = 0; i < n; i++) {
24         if (adj[i].size() != 2 && adj[i].size() != 4) {
25             cout << "NO\n";
26             return;
27         }
28         if (adj[i].size() == 4) {
29             x = i;
30         }
31     }
32     vector<int> e4(n), e2(n);
33     DSU d4(n), d2(n);
34     int m4 = 0;
35     for (int x = 0; x < n; x++) {
36         for (auto y : adj[x]) {
37             if (x > y) {
38                 continue;
39             }
40             if (adj[x].size() == 4 && adj[y].size() == 4) {
41                 e4[x] += 1;
42                 e4[y] += 1;
43                 m4 += 1;
44                 d4.merge(x, y);
45             } else {
46                 e2[x] += 1;
47                 e2[y] += 1;
48                 d2.merge(x, y);
49             }
50         }
51     }
52     if (m4 != k || d4.size(x) != k) {
53         cout << "NO\n";
54         return;
55     }
56     for (int i = 0; i < n; i++) {
57         if (e2[i] != 2) {
58             cout << "NO\n";
59             return;
60         }
61     }
62     for (int i = 0; i < n; i++) {
63         if (d2.size(i) != k) {
64             cout << "NO\n";
65             return;
66         }
67     }

```

```

65         }
66     }
67     set<int> s;
68     for (int i = 0; i < n; i++) {
69         if (adj[i].size() == 4) {
70             s.insert(d2.leader(i));
71         }
72     }
73     if (s.size() != k) {
74         cout << "NO\n";
75         return;
76     }
77     cout << "YES\n";
78 }
79 int main() {
80     ios::sync_with_stdio(false);
81     cin.tie(nullptr);
82     int t;
83     cin >> t;
84     while (t--) {
85         solve();
86     }
87     return 0;
88 }
```

## 507: Sweets Game

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Karlsson has visited Lillebror again. They found a box of chocolates and a big whipped cream cake at Lillebror's place. Karlsson immediately suggested to divide the sweets fairly between Lillebror and himself. Specifically, to play together a game he has just invented with the chocolates. The winner will get the cake as a reward.

The box of chocolates has the form of a hexagon. It contains 19 cells for the chocolates, some of which contain a chocolate. The players move in turns. During one move it is allowed to eat one or several chocolates that lay in the neighboring cells on one line, parallel to one of the box's sides. The picture below shows the examples of allowed moves and of an unacceptable one. The player who cannot make a move loses.

Karlsson makes the first move as he is Lillebror's guest and not vice versa. The players play optimally. Determine who will get the cake.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int L[5] = {0, 0, 0, 1, 2};
5 int R[5] = {2, 3, 4, 4, 4};
6 int id[5][5];
7 int dx[3] = {1, 0, 1};
8 int dy[3] = {0, 1, 1};
9 bool valid(int x, int y) {
10     return 0 <= x && x < 5 && L[x] <= y && y <= R[x];
11 }
12 int main() {
13     ios::sync_with_stdio(false);
14     cin.tie(nullptr);
15     memset(id, -1, sizeof(id));
16     int mask = 0;
17     int n = 0;
18     for (int x = 0; x < 5; x++) {
19         for (int y = L[x]; y <= R[x]; y++) {
20             id[x][y] = n;
21             char c;
22             cin >> c;
23             if (c == '0') {
24                 mask |= 1 << n;
25             }
26             n += 1;
27         }
28     }
29     vector<bool> win(1 << n);
30     for (int s = 0; s < (1 << n); s++) {
31         for (int x = 0; x < 5; x++) {
32             for (int y = L[x]; y <= R[x]; y++) {
33                 if (s >> id[x][y] & 1) {
34                     for (int k = 0; k < 3; k++) {
35                         int t = 0;
36                         int i = x, j = y;
37                         while (valid(i, j) && (s >> id[i][j] & 1)) {
38                             t |= 1 << id[i][j];
39                             win[s] = win[s] || !win[s ^ t];
40                             i += dx[k];
41                             j += dy[k];
42                         }
43                     }
44                 }
45             }
46         }
47     }
48     cout << (win[mask] ? "Karlsson" : "Lillebror") << "\n";
49     return 0;
50 }

```

## 508: Chris and Road

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

And while Mishka is enjoying her trip...

Chris is a little brown bear. No one knows, where and when he met Mishka, but for a long time they are together (excluding her current trip). However, best friends are important too. John is Chris' best friend.

Once walking with his friend, John gave Chris the following problem:

At the infinite horizontal road of width  $w$ , bounded by lines  $y = 0$  and  $y = w$ , there is a bus moving, presented as a convex polygon of  $n$  vertices. The bus moves continuously with a constant speed of  $v$  in a straight  $Ox$  line in direction of decreasing  $x$  coordinates, thus in time only  $x$  coordinates of its points are changing. Formally, after time  $t$  each of  $x$  coordinates of its points will be decreased by  $vt$ .

There is a pedestrian in the point  $(0, 0)$ , who can move only by a vertical pedestrian crossing, presented as a segment connecting points  $(0, 0)$  and  $(0, w)$  with any speed not exceeding  $u$ . Thus the pedestrian can move only in a straight line  $Oy$  in any direction with any speed not exceeding  $u$  and not leaving the road borders. The pedestrian can instantly change his speed, thus, for example, he can stop instantly.

Please look at the sample note picture for better understanding.

We consider the pedestrian is hit by the bus, if at any moment the point he is located in lies strictly inside the bus polygon (this means that if the point lies on the polygon vertex or on its edge, the pedestrian is not hit by the bus).

You are given the bus position at the moment 0. Please help Chris determine minimum amount of time the pedestrian needs to cross the road and reach the point  $(0, w)$  and not to be hit by the bus.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, w, v, u;
8     cin >> n >> w >> v >> u;
9     vector<int> x(n), y(n);
10    for (int i = 0; i < n; i++) {
11        cin >> x[i] >> y[i];
12    }
13    bool before = true;
14    for (int i = 0; i < n; i++) {
15        if (1LL * x[i] * u < 1LL * y[i] * v) {
16            before = false;
17        }
18    }
19    cout << fixed << setprecision(10);
20    if (before) {

```

```

1    cout << 1. * w / u << "\n";
2    return 0;
3 }
4 double ans = 1. * w / u;
5 for (int i = 0; i < n; i++) {
6     ans = max(ans, 1. * x[i] / v + 1. * (w - y[i]) / u);
7 }
8 cout << ans << "\n";
9 return 0;
10 }
```

## 509: Copy of a Copy of a Copy

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

It all started with a black-and-white picture, that can be represented as an  $n \times m$  matrix such that all its elements are either 0 or 1. The rows are numbered from 1 to  $n$ , the columns are numbered from 1 to  $m$ .

Several operations were performed on the picture (possibly, zero), each of one of the two kinds:

choose a cell such that it's not on the border (neither row 1 or  $n$ , nor column 1 or  $m$ ) and it's surrounded by four cells of the opposite color (four zeros if it's a one and vice versa) and paint it the opposite color itself;

make a copy of the current picture.

Note that the order of operations could be arbitrary, they were not necessarily alternating.

You are presented with the outcome: all  $k$  copies that were made. Additionally, you are given the initial picture. However, all  $k + 1$  pictures are shuffled.

Restore the sequence of the operations. If there are multiple answers, print any of them. The tests are constructed from the real sequence of operations, i. e. at least one answer always exists.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m, k;
```

```

8      cin >> n >> m >> k;
9      vector s(k + 1, vector<string>(n));
10     vector<int> f(k + 1);
11     for (int i = 0; i <= k; i++) {
12         for (int j = 0; j < n; j++) {
13             cin >> s[i][j];
14         }
15         for (int x = 1; x < n - 1; x++) {
16             for (int y = 1; y < m - 1; y++) {
17                 if (s[i][x][y] != s[i][x - 1][y] && s[i][x][y] != s[i][x + 1][y] && s[i][x][y] != s[i][x - 1] && s[i][x][y] != s[i][x][y + 1]) {
18                     f[i]++;
19                 }
20             }
21         }
22     }
23     vector<int> order(k + 1);
24     iota(order.begin(), order.end(), 0);
25     sort(order.begin(), order.end(), [&](int i, int j) {
26         return f[i] > f[j];
27     });
28     cout << order[0] + 1 << "\n";
29     vector<vector<int>> ans;
30     auto cur = s[order[0]];
31     for (int i = 1; i <= k; i++) {
32         int j = order[i];
33         for (int x = 0; x < n; x++) {
34             for (int y = 0; y < m; y++) {
35                 if (cur[x][y] != s[j][x][y]) {
36                     ans.push_back({1, x + 1, y + 1});
37                     cur[x][y] ^= 1;
38                 }
39             }
40         }
41         ans.push_back({2, j + 1});
42     }
43     cout << ans.size() << "\n";
44     for (auto v : ans) {
45         for (int i = 0; i < v.size(); i++) {
46             cout << v[i] << " \n"[i == v.size() - 1];
47         }
48     }
49     return 0;
50 }

```

**math****510: Blueprint for Seating**

- Time limit: 3 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

An aircraft manufacturing company wants to optimize their products for passenger airlines. The company's latest research shows that most of the delays happen because of slow boarding.

Most of the medium-sized aircraft are designed with 3-3 seat layout, meaning each row has 6 seats: 3 seats on the left side, a single aisle, and 3 seats on the right side. At each of the left and right sides there is a window seat, a middle seat, and an aisle seat. A passenger that boards an aircraft assigned to an aisle seat takes significantly less time than a passenger assigned to a window seat even when there is no one else in the aircraft.

The company decided to compute an inconvenience of a layout as the total sum of distances from each of the seats of a single row to the closest aisle. The distance from a seat to an aisle is the number of seats between them. For a 3-3 layout, a window seat has a distance of 2, a middle seat - 1, and an aisle seat - 0. The inconvenience of a 3-3 layout is  $(2 + 1 + 0) + (0 + 1 + 2) = 6$ . The inconvenience of a 3-5-3 layout is  $(2 + 1 + 0) + (0 + 1 + 2 + 1 + 0) + (0 + 1 + 2) = 10$ .

Formally, a layout is a sequence of positive integers  $a_1, a_2, \dots, a_{k+1}$  - group  $i$  having  $a_i$  seats, with  $k$  aisles between groups, the  $i$ -th aisle being between groups  $i$  and  $i + 1$ . This means that in a layout each aisle must always be between two seats, so no aisle can be next to a window, and no two aisles can be next to each other.

The company decided to design a layout with a row of  $n$  seats,  $k$  aisles and having the minimum inconvenience possible. Help them find the minimum inconvenience among all layouts of  $n$  seats and  $k$  aisles, and count the number of such layouts modulo 998 244 353.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
```

```

27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 void solve() {
36     int n, k;
37     cin >> n >> k;
38     int v = n / (2 * k);
39     i64 ans = 1LL * v * (v - 1) / 2 * (2 * k);
40     ans += 1LL * n % (2 * k) * v;
41     int t = n % (2 * k);
42     Z ways = 0;
43     if (n >= 2 * k) {
44         if (k >= 2) {
45             for (int i = 0; i <= k - 1; i++) {
46                 ways += (i % 2 ? -1 : 1) * comb.binom(k - 1, i) * comb.binom(t - 3 * i + k - 2, k - 2);
47                 ways += (i % 2 ? -1 : 1) * comb.binom(k - 1, i) * comb.binom(t - 1 - 3 * i + k - 2, k - 2);
48                 ways += (i % 2 ? -1 : 1) * comb.binom(k - 1, i) * comb.binom(t - 2 - 3 * i + k - 2, k - 2);
49             }
50         } else {
51             ways = t + 1;
52         }
53     } else {
54         ways = comb.binom(k - 1, n - k - 1);
55     }
56     cout << ans << " " << ways << "\n";
57 }
58 int main() {
59     ios::sync_with_stdio(false);
60     cin.tie(nullptr);
61     int t;
62     cin >> t;
63     while (t--) {
64         solve();
65     }
66     return 0;
67 }
```

## 511: Cyclic MEX

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

For an array  $a$ , define its cost as  $\sum_{i=1}^n \text{mex}^\dagger([a_1, a_2, \dots, a_i])$ .

You are given a permutation<sup>‡</sup>  $p$  of the set  $\{0, 1, 2, \dots, n - 1\}$ . Find the maximum cost across all cyclic shifts of  $p$ .

<sup>†</sup>mex( $[b_1, b_2, \dots, b_m]$ ) is the smallest non-negative integer  $x$  such that  $x$  does not occur among  $b_1, b_2, \dots, b_m$ .

<sup>‡</sup>A permutation of the set  $\{0, 1, 2, \dots, n - 1\}$  is an array consisting of  $n$  distinct integers from 0 to  $n - 1$  in arbitrary order. For example,  $[1, 2, 0, 4, 3]$  is a permutation, but  $[0, 1, 1]$  is not a permutation (1 appears twice in the array), and  $[0, 2, 3]$  is also not a permutation ( $n = 3$  but there is 3 in the array).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> p(n);
8     for (int i = 0; i < n; i++) {
9         cin >> p[i];
10    }
11    i64 ans = 0;
12    i64 res = n;
13    vector<int> s{-1};
14    for (int i = 0; i < 2 * n; i++) {
15        while (s.size() > 1 && p[i % n] < p[s.back() % n]) {
16            int x = s.back();
17            s.pop_back();
18            res -= 1LL * p[x % n] * (x - s.back());
19        }
20        res += 1LL * p[i % n] * (i - s.back());
21        s.push_back(i);
22        if (i >= n) {
23            ans = max(ans, res);
24        }
25    }
26    cout << ans << "\n";
27 }
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(nullptr);
31     int t;
32     cin >> t;
33     while (t--) {
34         solve();
35     }
36     return 0;
37 }
```

## 512: Geo Game

- Time limit: 2.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an interactive problem.

Theofanis and his sister are playing the following game.

They have  $n$  points in a 2D plane and a starting point  $(s_x, s_y)$ . Each player (starting from the first player) chooses one of the  $n$  points that wasn't chosen before and adds to the sum (which is initially 0) the square of the Euclidean distance from the previous point (which is either the starting point or it was chosen by the other person) to the new point (that the current player selected).

The game ends after exactly  $n$  moves (after all the points are chosen). The first player wins if the sum is even in the end. Otherwise, the second player wins.

Theofanis is a very competitive person and he hates losing. Thus, he wants to choose whether he should play first or second. Can you show him, which player to choose, and how he should play to beat his sister?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> x(n + 1), y(n + 1);
8     for (int i = 0; i < n + 1; i++) {
9         cin >> x[i] >> y[i];
10    }
11    array<set<int>, 2> set{set{}};
12    for (int i = 1; i <= n; i++) {
13        set[(x[i] + y[i]) % 2].insert(i);
14    }
15    int s = (x[0] + y[0]) % 2;
16    int t;
17    int res = n;
18    if (set[!s].size() <= set[s].size()) {
19        cout << "First" << endl;
20        t = s;
21    } else {
22        cout << "Second" << endl;
23        int x;
24        cin >> x;
25        set[0].erase(x);
26        set[1].erase(x);
27        res -= 1;
28        t = !s;
29    }
30    while (res > 0) {
31        res -= 1;
32        int x;
33        if (!set[!t].empty()) {
34            x = *set[!t].begin();
35        } else {
36            x = *set[t].begin();
37        }
38        cout << x << endl;
}

```

```

39         set[0].erase(x);
40         set[1].erase(x);
41         if (res > 0) {
42             res -= 1;
43             cin >> x;
44             set[0].erase(x);
45             set[1].erase(x);
46         }
47     }
48 }
49 int main() {
50     ios::sync_with_stdio(false);
51     cin.tie(nullptr);
52     int t;
53     cin >> t;
54     while (t--) {
55         solve();
56     }
57     return 0;
58 }
```

### 513: Monocarp and the Set

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp has  $n$  numbers  $1, 2, \dots, n$  and a set (initially empty). He adds his numbers to this set  $n$  times in some order. During each step, he adds a new number (which has not been present in the set before). In other words, the sequence of added numbers is a permutation of length  $n$ .

Every time Monocarp adds an element into the set except for the first time, he writes out a character:  
if the element Monocarp is trying to insert becomes the maximum element in the set, Monocarp writes out the character `>`;  
if the element Monocarp is trying to insert becomes the minimum element in the set, Monocarp writes out the character `<`;  
if none of the above, Monocarp writes out the character `?`.

You are given a string  $s$  of  $n - 1$  characters, which represents the characters written out by Monocarp (in the order he wrote them out). You have to process  $m$  queries to the string. Each query has the following format:

$i c$  - replace  $s_i$  with the character  $c$ .

Both before processing the queries and after each query, you have to calculate the number of different ways to order the integers  $1, 2, 3, \dots, n$  such that, if Monocarp inserts the integers into the set in that order, he gets the string  $s$ . Since the answers might be large, print them modulo 998244353.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int n, m;
39     cin >> n >> m;
40     n--;
41     Z ans = 1;
42     string s;
43     cin >> s;
44     for (int i = 1; i < n; i++) {
45         if (s[i] == '?') {
46             ans *= i;
47         }
48     }
49     auto query = [&]() {
50         if (s[0] == '?') {
51             cout << 0 << "\n";
52         } else {
53             cout << ans << "\n";
54         }
55     };
56     query();
57     while (m--) {
58         int x;
```

```

59     char y;
60     cin >> x >> y;
61     x--;
62     if (x && s[x] == '?') {
63         ans *= comb.inv(x);
64     }
65     s[x] = y;
66     if (x && y == '?') {
67         ans *= x;
68     }
69     query();
70 }
71 return 0;
72 }
```

### 514: Salyg1n and Array (simple version)

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the simple version of the problem. The only difference between the versions is the limit on the number of queries. In this version, you can make no more than 100 queries. You can make hacks only if both versions of the problem are solved.

This is an interactive problem!

salyg1n has given you a positive integer  $k$  and wants to play a game with you. He has chosen an array of  $n$  integers  $a_1, a_2, \dots, a_n$  ( $1 \leq a_i \leq 10^9$ ). You must print  $a_1 \oplus a_2 \oplus \dots \oplus a_n$ , where  $\oplus$  denotes the bitwise XOR operation. You can make queries of the following type:

?  $i$ : in response to this query, you will receive  $a_i \oplus a_{i+1} \oplus \dots \oplus a_{i+k-1}$ . Also, after this query, the subarray  $a_i, a_{i+1}, \dots, a_{i+k-1}$  will be reversed, i.e., the chosen array  $a$  will become:  $a_1, a_2, \dots, a_{i-1}, a_{i+k-1}, a_{i+k-2}, \dots, a_{i+1}, a_i, a_{i+k}, \dots, a_n$ .

You can make no more than 100 queries to answer the problem.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     int ans = 0;
8     int i = 0;
9     while (i + k <= n) {
10         cout << "? " << i + 1 << endl;
```

```

11     int res;
12     cin >> res;
13     ans ^= res;
14     i += k;
15 }
16 int t = n - i;
17 cout << "? " << n - k - t / 2 + 1 << endl;
18 int res;
19 cin >> res;
20 ans ^= res;
21 cout << "? " << n - k + 1 << endl;
22 cin >> res;
23 ans ^= res;
24 cout << "! " << ans << endl;
25 }
26 int main() {
27     ios::sync_with_stdio(false);
28     cin.tie(nullptr);
29     int t;
30     cin >> t;
31     while (t--) {
32         solve();
33     }
34     return 0;
35 }
```

## 515: Replace With Product

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Given an array  $a$  of  $n$  positive integers. You need to perform the following operation exactly once:

Choose 2 integers  $l$  and  $r$  ( $1 \leq l \leq r \leq n$ ) and replace the subarray  $a[l \dots r]$  with the single element: the product of all elements in the subarray  $(a_l \cdot \dots \cdot a_r)$ .

For example, if an operation with parameters  $l = 2, r = 4$  is applied to the array  $[5, 4, 3, 2, 1]$ , the array will turn into  $[5, 24, 1]$ .

Your task is to maximize the sum of the array after applying this operation. Find the optimal subarray to apply this operation.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
```

```

5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    int l = 0, r = n - 1;
12    while (l < r && a[l] == 1) {
13        l++;
14    }
15    while (l < r && a[r] == 1) {
16        r--;
17    }
18    i64 res = 1;
19    for (int i = l; i <= r; i++) {
20        res *= a[i];
21        if (res > 1E9) {
22            cout << l + 1 << " " << r + 1 << "\n";
23            return;
24        }
25    }
26    vector<int> p;
27    for (int i = 0; i < n; i++) {
28        if (a[i] > 1) {
29            p.push_back(i);
30        }
31    }
32    int sum = accumulate(a.begin(), a.end(), 0);
33    int L = 1, R = 1, ans = sum;
34    for (int i = 0; i < p.size(); i++) {
35        int l = p[i];
36        int res = 1;
37        int s = 0;
38        for (int j = i; j < p.size(); j++) {
39            int r = p[j];
40            res *= a[r];
41            s += a[r] - 1;
42            int v = sum - (r - l + 1) - s + res;
43            if (v > ans) {
44                ans = v;
45                L = l + 1, R = r + 1;
46            }
47        }
48    }
49    cout << L << " " << R << "\n";
50 }
51 int main() {
52     ios::sync_with_stdio(false);
53     cin.tie(nullptr);
54     int t;
55     cin >> t;
56     while (t--) {
57         solve();
58     }
59     return 0;
60 }

```

**516: Guess Game**

- Time limit: 3 seconds

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Carol has a sequence  $s$  of  $n$  non-negative integers. She wants to play the “Guess Game” with Alice and Bob.

To play the game, Carol will randomly select two integer indices  $i_a$  and  $i_b$  within the range  $[1, n]$ , and set  $a = s_{i_a}$ ,  $b = s_{i_b}$ . Please note that  $i_a$  and  $i_b$  may coincide.

Carol will tell:

the value of  $a$  to Alice;

the value of  $b$  to Bob;

the value of  $a \mid b$  to both Alice and Bob, where  $\mid$  denotes the bitwise OR operation.

Please note that Carol will not tell any information about  $s$  to either Alice or Bob.

Then the guessing starts. The two players take turns making guesses, with Alice starting first. The goal of both players is to establish which of the following is true:  $a < b$ ,  $a > b$ , or  $a = b$ .

In their turn, a player can do one of the following two things:

say “I don’t know”, and pass the turn to the other player;

say “I know”, followed by the answer “ $a < b$ ”, “ $a > b$ ”, or “ $a = b$ ”; then the game ends.

Alice and Bob hear what each other says, and can use this information to deduce the answer. Both Alice and Bob are smart enough and only say “I know” when they are completely sure.

You need to calculate the expected value of the number of turns taken by players in the game. Output the answer modulo 998 244 353.

Formally, let  $M = 998\,244\,353$ . It can be shown that the answer can be expressed as an irreducible fraction  $\frac{p}{q}$ , where  $p$  and  $q$  are integers and  $q \not\equiv 0 \pmod{M}$ . Output the integer equal to  $p \cdot q^{-1} \pmod{M}$ . In other words, output such an integer  $x$  that  $0 \leq x < M$  and  $x \cdot q \equiv p \pmod{M}$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;

```

```
10         }
11     }
12     return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 void solve() {
35     int n;
36     cin >> n;
37     Z ans = 0;
38     vector<int> a(n);
39     for (int i = 0; i < n; i++) {
40         cin >> a[i];
41     }
42     sort(a.begin(), a.end());
43     ans += 1LL * n * n;
44     for (int t = 0; t < 30; t++) {
45         for (int l = 0, r = 0; l < n; l = r) {
46             array<int, 2> cnt{};
47             while (r < n && a[l] / 2 == a[r] / 2) {
48                 cnt[a[r] % 2]++;
49                 r++;
50             }
51             ans += 1LL * cnt[1] * (cnt[1] + cnt[0]);
52         }
53         for (auto &x : a) {
54             x /= 2;
55         }
56     }
57     ans /= 1LL * n * n;
58     cout << ans << "\n";
59 }
60 int main() {
61     ios::sync_with_stdio(false);
62     cin.tie(nullptr);
63     int t;
64     cin >> t;
65     while (t--) {
66         solve();
67     }
68     return 0;
69 }
```

## 517: The Great Equalizer

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Tema bought an old device with a small screen and a worn-out inscription “The Great Equalizer” on the side.

The seller said that the device needs to be given an array  $a$  of integers as input, after which “The Great Equalizer” will work as follows:

Sort the current array in non-decreasing order and remove duplicate elements leaving only one occurrence of each element.

If the current length of the array is equal to 1, the device stops working and outputs the single number in the array - output value of the device.

Add an arithmetic progression  $\{n, n - 1, n - 2, \dots, 1\}$  to the current array, where  $n$  is the length of the current array. In other words,  $n - i$  is added to the  $i$ -th element of the array, when indexed from zero.

Go to the first step.

To test the operation of the device, Tema came up with a certain array of integers  $a$ , and then wanted to perform  $q$  operations on the array  $a$  of the following type:

Assign the value  $x$  ( $1 \leq x \leq 10^9$ ) to the element  $a_i$  ( $1 \leq i \leq n$ ).

Give the array  $a$  as input to the device and find out the result of the device’s operation, while the array  $a$  remains unchanged during the operation of the device.

Help Tema find out the output values of the device after each operation.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
```

```

11     multiset<int> s, d{0};
12     auto add = [&](int x) {
13         auto it = s.insert(x);
14         auto r = next(it);
15         if (it != s.begin()) {
16             d.insert(x - *prev(it));
17         }
18         if (r != s.end()) {
19             d.insert(*r - x);
20         }
21         if (it != s.begin() && r != s.end()) {
22             d.extract(*r - *prev(it));
23         }
24     };
25     auto del = [&](int x) {
26         auto it = s.find(x);
27         auto r = next(it);
28         if (it != s.begin()) {
29             d.extract(x - *prev(it));
30         }
31         if (r != s.end()) {
32             d.extract(*r - x);
33         }
34         if (it != s.begin() && r != s.end()) {
35             d.insert(*r - *prev(it));
36         }
37         s.erase(it);
38     };
39     for (int i = 0; i < n; i++) {
40         add(a[i]);
41     }
42     int q;
43     cin >> q;
44     for (int i = 0; i < q; i++) {
45         int x, y;
46         cin >> x >> y;
47         x--;
48         del(a[x]);
49         a[x] = y;
50         add(a[x]);
51         int ans = *s.rbegin() + *d.rbegin();
52         cout << ans << " \n"[i == q - 1];
53     }
54 }
55 int main() {
56     ios::sync_with_stdio(false);
57     cin.tie(nullptr);
58     int t;
59     cin >> t;
60     while (t--) {
61         solve();
62     }
63     return 0;
64 }
```

**518: String Mark**

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

At the Byteland State University marks are strings of the same length. Mark x is considered better than y if string y is lexicographically smaller than x.

Recently at the BSU was an important test work on which Vasya received the mark a. It is very hard for the teacher to remember the exact mark of every student, but he knows the mark b, such that every student received mark strictly smaller than b.

Vasya isn't satisfied with his mark so he decided to improve it. He can swap characters in the string corresponding to his mark as many times as he like. Now he wants to know only the number of different ways to improve his mark so that his teacher didn't notice something suspicious.

More formally: you are given two strings a, b of the same length and you need to figure out the number of different strings c such that:

- 1) c can be obtained from a by swapping some characters, in other words c is a permutation of a.
- 2) String a is lexicographically smaller than c.
- 3) String c is lexicographically smaller than b.

For two strings x and y of the same length it is true that x is lexicographically smaller than y if there exists such i, that  $x_1 = y_1, x_2 = y_2, \dots, x_{i-1} = y_{i-1}, x_i < y_i$ .

Since the answer can be very large, you need to find answer modulo  $10^9 + 7$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
```

```

22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     string a, b;
39     cin >> a >> b;
40     int n = a.size();
41     if (a >= b) {
42         cout << 0 << "\n";
43         return 0;
44     }
45     array<int, 26> cnt{};
46     for (auto c : a) {
47         cnt[c - 'a']++;
48     }
49     auto get = [&](const auto &s) {
50         Z ans = 0;
51         Z res = 1;
52         auto c = cnt;
53         for (int i = 0; i < 26; i++) {
54             res *= comb.invfac(c[i]);
55         }
56         for (int i = 0; i < n; i++) {
57             int x = s[i] - 'a';
58             for (int j = 0; j < x; j++) {
59                 if (c[j] > 0) {
60                     ans += res * c[j] * comb.fac(n - i - 1);
61                 }
62             }
63             if (c[x] == 0) {
64                 break;
65             }
66             res *= c[x];
67             c[x]--;
68         }
69         return ans;
70     };
71     auto ans = get(b) - get(a) - 1;
72     cout << ans << "\n";
73     return 0;
74 }

```

## 519: Below the Diagonal

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

You are given a square matrix consisting of  $n$  rows and  $n$  columns. We assume that the rows are numbered from 1 to  $n$  from top to bottom and the columns are numbered from 1 to  $n$  from left to right. Some cells ( $n - 1$  cells in total) of the matrix are filled with ones, the remaining cells are filled with zeros. We can apply the following operations to the matrix:

Swap  $i$ -th and  $j$ -th rows of the matrix;

Swap  $i$ -th and  $j$ -th columns of the matrix.

You are asked to transform the matrix into a special form using these operations. In that special form all the ones must be in the cells that lie below the main diagonal. Cell of the matrix, which is located on the intersection of the  $i$ -th row and of the  $j$ -th column, lies below the main diagonal if  $i > j$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> x(n - 1), y(n - 1);
10    vector<int> cnt(n);
11    for (int i = 0; i < n - 1; i++) {
12        cin >> x[i] >> y[i];
13        x[i]--, y[i]--;
14        cnt[x[i]]++;
15    }
16    vector<int> p(n);
17    iota(p.begin(), p.end(), 0);
18    sort(p.begin(), p.end(),
19         [&](int i, int j) {
20             return cnt[i] < cnt[j];
21         });
22    vector<int> invp(n);
23    for (int i = 0; i < n; i++) {
24        invp[p[i]] = i;
25    }
26    vector<int> lim(n, n);
27    for (int i = 0; i < n - 1; i++) {
28        lim[y[i]] = min(lim[y[i]], invp[x[i]]);
29    }
30    sort(p.begin(), p.end(),
31         [&](int i, int j) {
32             return lim[i] < lim[j];
33         });
34    vector<array<int, 3>> ans;
35    for (int i = 0; i < n; i++) {
36        while (i != invp[i]) {
37            ans.push_back({1, i + 1, invp[i] + 1});

```

```

38         swap(invP[i], invP[invP[i]]);
39     }
40 }
41 for (int i = 0; i < n; i++) {
42     invP[p[i]] = i;
43 }
44 for (int i = 0; i < n; i++) {
45     while (i != invP[i]) {
46         ans.push_back({2, i + 1, invP[i] + 1});
47         swap(invP[i], invP[invP[i]]);
48     }
49 }
50 cout << ans.size() << "\n";
51 for (auto [t, i, j] : ans) {
52     cout << t << " " << i << " " << j << "\n";
53 }
54 return 0;
55 }
```

## 520: The BOSS Can Count Pairs

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given two arrays  $a$  and  $b$ , both of length  $n$ .

Your task is to count the number of pairs of integers  $(i, j)$  such that  $1 \leq i < j \leq n$  and  $a_i \cdot a_j = b_i + b_j$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> cnt(n + 1);
8     vector<pair<int, int>> c(n);
9     for (int i = 0; i < n; i++) {
10         cin >> c[i].first;
11     }
12     for (int i = 0; i < n; i++) {
13         cin >> c[i].second;
14     }
15     i64 ans = 0;
16     sort(c.begin(), c.end());
17     for (int s = 1; s * s <= 2 * n; s++) {
18         cnt.assign(n + 1, 0);
19         for (auto [a, b] : c) {
20             int v = a * s - b;
21             if (1 <= v && v <= n) {
22                 ans += cnt[v];
23             }
24         }
25     }
26 }
```

```

24         if (a == s) {
25             cnt[b]++;
26         }
27     }
28     cout << ans << "\n";
29 }
30 int main() {
31     ios::sync_with_stdio(false);
32     cin.tie(nullptr);
33     int t;
34     cin >> t;
35     while (t--) {
36         solve();
37     }
38     return 0;
39 }
40 }
```

## 521: Red-Blue Operations (Easy Version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The only difference between easy and hard versions is the maximum values of  $n$  and  $q$ .

You are given an array, consisting of  $n$  integers. Initially, all elements are red.

You can apply the following operation to the array multiple times. During the  $i$ -th operation, you select an element of the array; then:

if the element is red, it increases by  $i$  and becomes blue;

if the element is blue, it decreases by  $i$  and becomes red.

The operations are numbered from 1, i. e. during the first operation some element is changed by 1 and so on.

You are asked  $q$  queries of the following form:

given an integer  $k$ , what can the largest minimum in the array be if you apply exactly  $k$  operations to it?

Note that the operations don't affect the array between queries, all queries are asked on the initial array  $a$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 1E18;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, q;
9     cin >> n >> q;
10    vector<i64> a(n);
11    for (int i = 0; i < n; i++) {
12        cin >> a[i];
13    }
14    sort(a.begin(), a.end());
15    vector<i64> f(n + 1, inf);
16    for (int i = 0; i < n; i++) {
17        f[i + 1] = min(f[i] + 1, a[i] + 1);
18    }
19    i64 sum = accumulate(a.begin(), a.end(), 0LL);
20    while (q--) {
21        i64 k;
22        cin >> k;
23        if (k < n) {
24            cout << min(f[k], a[k]) << " ";
25        } else {
26            i64 ans;
27            i64 s = sum;
28            if ((k - n) % 2 == 0) {
29                ans = f[n] + k - n;
30                s += 1LL * n * (k + k - n + 1) / 2 - (k - n) / 2;
31            } else {
32                ans = min(a[n - 1], f[n - 1] + k - (n - 1));
33                s += 1LL * (n - 1) * (k + k - n + 2) / 2 - (k - n + 1) / 2;
34            }
35            ans = min(1LL * ans, s >= 0 ? s / n : (s - n + 1) / n);
36            cout << ans << " ";
37        }
38    }
39    cout << "\n";
40    return 0;
41 }

```

## 522: Collisions

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

On a number line there are  $n$  balls. At time moment 0 for each ball the following data is known: its coordinate  $x_i$ , speed  $v_i$  (possibly, negative) and weight  $m_i$ . The radius of the balls can be ignored.

The balls collide elastically, i.e. if two balls weighing  $m_1$  and  $m_2$  and with speeds  $v_1$  and  $v_2$  collide, their new speeds will be:

Your task is to find out, where each ball will be  $t$  seconds after.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, t;
8     cin >> n >> t;
9     vector<double> x(n), v(n), m(n);
10    for (int i = 0; i < n; i++) {
11        cin >> x[i] >> v[i] >> m[i];
12    }
13    vector<int> order(n);
14    iota(order.begin(), order.end(), 0);
15    sort(order.begin(), order.end(), [&](int i, int j) {
16        return x[i] < x[j];
17    });
18    priority_queue<pair<double, int>> h;
19    double cur = 0;
20    auto check = [&](int i) {
21        int a = order[i];
22        int b = order[i + 1];
23        if (v[a] > v[b]) {
24            h.emplace(-(cur + (x[b] - x[a]) / (v[a] - v[b])), i);
25        }
26    };
27    for (int i = 1; i < n; i++) {
28        check(i - 1);
29    }
30    while (!h.empty()) {
31        auto [p, i] = h.top();
32        h.pop();
33        p = -p;
34        if (p > t) {
35            break;
36        }
37        for (int i = 0; i < n; i++) {
38            x[i] += v[i] * (p - cur);
39        }
40        cur = p;
41        int a = order[i], b = order[i + 1];
42        if (abs(x[a] - x[b]) > 1E-6) {
43            continue;
44        }
45        tie(v[a], v[b]) = pair(((m[a] - m[b]) * v[a] + 2 * m[b] * v[b]) / (m[a] + m[b]),
46                               ((m[b] - m[a]) * v[b] + 2 * m[a] * v[a]) / (m[a] + m[b]));
47        if (i) {
48            check(i - 1);
49        }
50        if (i + 2 < n) {
51            check(i + 1);
52        }
53    }
54    cout << fixed << setprecision(10);
55    for (int i = 0; i < n; i++) {
56        x[i] += v[i] * (t - cur);
57        cout << x[i] << "\n";
58    }
59    return 0;
60 }
```

## 523: Longest Subsequence

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given array  $a$  with  $n$  elements and the number  $m$ . Consider some subsequence of  $a$  and the value of least common multiple (LCM) of its elements. Denote LCM as  $l$ . Find any longest subsequence of  $a$  with the value  $l \leq m$ .

A subsequence of  $a$  is an array we can get by erasing some elements of  $a$ . It is allowed to erase zero or all elements.

The LCM of an empty array equals 1.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<int> f(m + 1);
10    vector<int> a(n);
11    for (int i = 0; i < n; i++) {
12        cin >> a[i];
13        if (a[i] <= m) {
14            f[a[i]] += 1;
15        }
16    }
17    for (int i = m; i; i--) {
18        for (int j = 2 * i; j <= m; j += i) {
19            f[j] += f[i];
20        }
21    }
22    int l = max_element(f.begin(), f.end()) - f.begin();
23    if (l == 0) {
24        cout << 1 << " " << 0 << "\n";
25        return 0;
26    }
27    int k = f[l];
28    cout << l << " " << k << "\n";
29    for (int i = 0; i < n; i++) {
30        if (l % a[i] == 0) {
31            cout << i + 1 << " ";
32        }
33    }
34    cout << "\n";
35    return 0;
}

```

```
36 }
```

## 524: Intersection

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given two set of points. The first set is determined by the equation  $A_1x + B_1y + C_1 = 0$ , and the second one is determined by the equation  $A_2x + B_2y + C_2 = 0$ .

Write the program which finds the number of points in the intersection of two given sets.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int A1, B1, C1;
8     cin >> A1 >> B1 >> C1;
9     int A2, B2, C2;
10    cin >> A2 >> B2 >> C2;
11    if (A1 == 0 && B1 == 0 && C1 != 0) {
12        cout << 0 << "\n";
13        return 0;
14    }
15    if (A2 == 0 && B2 == 0 && C2 != 0) {
16        cout << 0 << "\n";
17        return 0;
18    }
19    if (A1 == 0 && B1 == 0 && C1 == 0) {
20        cout << -1 << "\n";
21        return 0;
22    }
23    if (A2 == 0 && B2 == 0 && C2 == 0) {
24        cout << -1 << "\n";
25        return 0;
26    }
27    if (A1 * B2 - A2 * B1 != 0) {
28        cout << 1 << "\n";
29        return 0;
30    }
31    if (A1 * C2 - A2 * C1 == 0 && B1 * C2 - B2 * C1 == 0 ) {
32        cout << -1 << "\n";
33    } else {
34        cout << 0 << "\n";
35    }
36    return 0;
37 }
```

## 525: Equation

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an equation:

Your task is to find the number of distinct roots of the equation and print all of them in ascending order.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int A, B, C;
8     cin >> A >> B >> C;
9     cout << fixed << setprecision(10);
10    if (A != 0) {
11        if (A < 0) {
12            A = -A;
13            B = -B;
14            C = -C;
15        }
16        i64 t = 1LL * B * B - 4LL * A * C;
17        if (t < 0) {
18            cout << 0 << "\n";
19            return 0;
20        }
21        if (t == 0) {
22            cout << 1 << "\n";
23            cout << -1. * B / (2 * A) << "\n";
24        } else {
25            cout << 2 << "\n";
26            cout << (-B - sqrt(t)) / (2 * A) << "\n";
27            cout << (-B + sqrt(t)) / (2 * A) << "\n";
28        }
29    } else if (B != 0) {
30        cout << 1 << "\n";
31        cout << 1. * -C / B << "\n";
32    } else {
33        if (C == 0) {
34            cout << -1 << "\n";
35        } else {
36            cout << 0 << "\n";
37        }
38    }
39    return 0;
40 }
```

## 526: Vlad and the Nice Paths (easy version)

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an easy version of the problem, it differs from the hard one only by constraints on  $n$  and  $k$ .

Vlad found a row of  $n$  tiles and the integer  $k$ . The tiles are indexed from left to right and the  $i$ -th tile has the color  $c_i$ . After a little thought, he decided what to do with it.

You can start from any tile and jump to any number of tiles right, forming the path  $p$ . Let's call the path  $p$  of length  $m$  nice if:

$p$  can be divided into blocks of length exactly  $k$ , that is,  $m$  is divisible by  $k$ ;

$c_{p_1} = c_{p_2} = \dots = c_{p_k}$ ;

$c_{p_{k+1}} = c_{p_{k+2}} = \dots = c_{p_{2k}}$ ;

...

$c_{p_{m-k+1}} = c_{p_{m-k+2}} = \dots = c_{p_m}$ ;

Your task is to find the number of nice paths of maximum length. Since this number may be too large, print it modulo  $10^9 + 7$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
```

```

22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 void update(pair<int, Z> &a, pair<int, Z> b) {
36     if (b.first > a.first) {
37         a = b;
38     } else if (a.first == b.first) {
39         a.second += b.second;
40     }
41 }
42 void solve() {
43     int n, k;
44     cin >> n >> k;
45     vector<int> c(n);
46     for (int i = 0; i < n; i++) {
47         cin >> c[i];
48     }
49     vector<pair<int, Z>> dp(n + 1);
50     dp[0] = {0, 1};
51     pair<int, Z> ans{0, 1};
52     for (int i = 1; i <= n; i++) {
53         int cnt = 0;
54         for (int j = i - 1; j >= 0; j--) {
55             cnt += (c[j] == c[i - 1]);
56             if (cnt >= k) {
57                 update(dp[i], {dp[j].first + 1, dp[j].second * comb.binom(cnt - 1, k - 1)
58                         });
59             }
60             update(ans, dp[i]);
61         }
62         cout << ans.second << "\n";
63     }
64     int main() {
65         ios::sync_with_stdio(false);
66         cin.tie(nullptr);
67         int t;
68         cin >> t;
69         while (t--) {
70             solve();
71         }
72         return 0;
73     }

```

## 527: Another Wine Tasting Event

- Time limit: 0.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

After the first successful edition, Gabriella has been asked to organize a second wine tasting event. There will be  $2n - 1$  bottles of wine arranged in a row, each of which is either red wine or white wine.

This time, Gabriella has already chosen the type and order of all the bottles. The types of the wines are represented by a string  $s$  of length  $2n - 1$ . For each  $1 \leq i \leq 2n - 1$ , it holds that  $s_i = R$  if the  $i$ -th bottle is red wine, and  $s_i = W$  if the  $i$ -th bottle is white wine.

Exactly  $n$  critics have been invited to attend. The critics are numbered from 1 to  $n$ . Just like last year, each critic  $j$  wants to taste an interval of wines, that is, the bottles at positions  $a_j, a_j + 1, \dots, b_j$  for some  $1 \leq a_j \leq b_j \leq 2n - 1$ . Moreover, they have the following additional requirements:

each of them wants to taste at least  $n$  wines, that is, it must hold that  $b_j - a_j + 1 \geq n$ ;

no two critics must taste exactly the same wines, that is, if  $j \neq k$  it must hold that  $a_j \neq a_k$  or  $b_j \neq b_k$ .

Gabriella knows that, since the event is held in a coastal region of Italy, critics are especially interested in the white wines, and don't care much about the red ones. (Indeed, white wine is perfect to accompany seafood.) Thus, to ensure fairness, she would like that all critics taste the same number of white wines.

Help Gabriella find an integer  $x$  (with  $0 \leq x \leq 2n - 1$ ) such that there exists a valid assignment of intervals to critics where each critic tastes exactly  $x$  white wines. It can be proved that at least one such  $x$  always exists.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     string s;
10    cin >> s;
11    vector<int> pre(2 * n);
12    int ans = 0;
13    for (int i = 0; i < 2 * n - 1; i++) {
14        pre[i + 1] = pre[i] + (s[i] == 'W');
15    }
16    for (int i = 0; i < n; i++) {
17        ans = max(ans, pre[i + n] - pre[i]);
18    }
19    cout << ans << "\n";
20    return 0;
21 }
```

## 528: Moving Dots

- Time limit: 1.5 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

We play a game with  $n$  dots on a number line.

The initial coordinate of the  $i$ -th dot is  $x_i$ . These coordinates are distinct. Every dot starts moving simultaneously with the same constant speed.

Each dot moves in the direction of the closest dot (different from itself) until it meets another dot. In the case of a tie, it goes to the left. Two dots meet if they are in the same coordinate, after that, they stop moving.

After enough time, every dot stops moving. The result of a game is the number of distinct coordinates where the dots stop.

Because this game is too easy, calculate the sum of results when we play the game for every subset of the given  $n$  dots that has at least two dots. As the result can be very large, print the sum modulo  $10^9 + 7$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 template<int P>
15 # struct MInt;
16 constexpr int P = 1000000007;
17 using Z = MInt<P>;
18 int main() {
19     ios::sync_with_stdio(false);
20     cin.tie(nullptr);
21     int n;
22     cin >> n;
23     vector<int> x(n);
24     for (int i = 0; i < n; i++) {
25         cin >> x[i];
26     }
27     vector<Z> pw(n + 1);

```

```

28     pw[0] = 1;
29     for (int i = 1; i <= n; i++) {
30         pw[i] = pw[i - 1] * 2;
31     }
32     Z ans = 0;
33     for (int i = 0; i < n; i++) {
34         int l = i, r = i;
35         for (int j = i + 1; j < n; j++) {
36             while (l >= 0 && x[i] - x[l] <= x[j] - x[i]) {
37                 l--;
38             }
39             while (r < n && x[r] - x[j] < x[j] - x[i]) {
40                 r++;
41             }
42             ans += pw[l + 1] * pw[n - r];
43         }
44     }
45     cout << ans << "\n";
46     return 0;
47 }
```

**misc****530: Happy Sets**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Define a set  $A$  as a child of set  $B$  if and only if for each element of value  $x$  that is in  $A$ , there exists an element of value  $x + 1$  that is in  $B$ .

Given integers  $N$  and  $K$ . In order to make Chaneka happy, you must find the number of different arrays containing  $N$  sets  $[S_1, S_2, S_3, \dots, S_N]$  such that:

- Each set can only contain zero or more different integers from 1 to  $K$ .
- There exists a way to rearrange the order of the array of sets into  $[S'_1, S'_2, S'_3, \dots, S'_N]$  such that  $S'_i$  is a child of  $S'_{i+1}$  for each  $1 \leq i \leq N - 1$ .

Print the answer modulo 998 244 353.

Two sets are considered different if and only if there is at least one value that only occurs in one of the two sets.

Two arrays of sets are considered different if and only if there is at least one index  $i$  such that the sets at index  $i$  in the two arrays are different.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int N, K;
39     cin >> N >> K;
40     Z ans = 1;
41     for (int i = 0; i < N; i++) {
42         if (1 + (N - 1 - i) <= K) {
43             Z res = 1;
44             Z ways = 0;
45             ways += comb.fac(N - i) * power(Z(N - i + 1), K - (N - i) + 1);
46             ways -= comb.fac(N - i - 1) * power(Z(N - i), K - (N - i - 1) + 1);
47             ans += ways * comb.binom(N, N - i) * comb.fac(N - i);
48         }
49     }
50     cout << ans << "\n";
51     return 0;
52 }

```

### 531: Restoration of string

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

A substring of some string is called the most frequent, if the number of its occurrences is not less than number of occurrences of any other substring.

You are given a set of strings. A string (not necessarily from this set) is called good if all elements of the set are the most frequent substrings of this string. Restore the non-empty good string with minimum length. If several such strings exist, restore lexicographically minimum string. If there are no good strings, print “NO” (without quotes).

A substring of a string is a contiguous subsequence of letters in the string. For example, “ab”, “c”, “abc” are substrings of string “abc”, while “ac” is not a substring of that string.

The number of occurrences of a substring in a string is the number of starting positions in the string where the substring occurs. These occurrences could overlap.

String a is lexicographically smaller than string b, if a is a prefix of b, or a has a smaller letter at the first position where a and b differ.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<string> s(n);
10    for (int i = 0; i < n; i++) {
11        cin >> s[i];
12    }
13    vector<int> l(26, -1), r(26, -1);
14    vector<bool> vis(26);
15    for (int i = 0; i < n; i++) {
16        for (int j = 1; j < s[i].size(); j++) {
17            int x = s[i][j - 1] - 'a';
18            int y = s[i][j] - 'a';
19            if (r[x] != -1 && r[x] != y) {
20                cout << "NO\n";
21                return 0;
22            }
23            r[x] = y;
24            if (l[y] != -1 && l[y] != x) {
25                cout << "NO\n";
26                return 0;
27            }
28            l[y] = x;
29        }
30        for (auto c : s[i]) {
31            vis[c - 'a'] = true;
32        }
33    }
34    string ans;

```

```

35     for (int i = 0; i < 26; i++) {
36         if (l[i] == -1 && vis[i]) {
37             for (int j = i; j != -1; j = r[j]) {
38                 vis[j] = false;
39                 ans += 'a' + j;
40             }
41         }
42     }
43     if (find(vis.begin(), vis.end(), true) != vis.end()) {
44         cout << "NO\n";
45         return 0;
46     }
47     cout << ans << "\n";
48     return 0;
49 }
```

## 532: More Wrong

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an interactive problem.

The jury has hidden a permutation<sup>†</sup>  $p$  of length  $n$ .

In one query, you can pick two integers  $l$  and  $r$  ( $1 \leq l < r \leq n$ ) by paying  $(r - l)^2$  coins. In return, you will be given the number of inversions<sup>‡</sup> in the subarray  $[p_l, p_{l+1}, \dots, p_r]$ .

Find the index of the maximum element in  $p$  by spending at most  $5 \cdot n^2$  coins.

Note: the grader is not adaptive: the permutation is fixed before any queries are made.

<sup>†</sup> A permutation of length  $n$  is an array consisting of  $n$  distinct integers from 1 to  $n$  in arbitrary order. For example,  $[2, 3, 1, 5, 4]$  is a permutation, but  $[1, 2, 2]$  is not a permutation (2 appears twice in the array), and  $[1, 3, 4]$  is also not a permutation ( $n = 3$  but there is 4 in the array).

<sup>‡</sup> The number of inversions in an array is the number of pairs of indices  $(i, j)$  such that  $i < j$  and  $a_i > a_j$ . For example, the array  $[10, 2, 6, 3]$  contains 4 inversions. The inversions are  $(1, 2)$ ,  $(1, 3)$ ,  $(1, 4)$ , and  $(3, 4)$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int query(int l, int r) {
5     if (l == r) {
6         return 0;
```

```

7      }
8      cout << "? " << l << " " << r << endl;
9      int ans;
10     cin >> ans;
11     return ans;
12 }
13 int solve(int l, int r) {
14     if (l == r) {
15         return l;
16     }
17     int m = (l + r) / 2;
18     int x = solve(l, m);
19     int y = solve(m + 1, r);
20     if (query(l, y - 1) == query(l, y)) {
21         return y;
22     } else {
23         return x;
24     }
25 }
26 void solve() {
27     int n;
28     cin >> n;
29     int ans = solve(1, n);
30     cout << "! " << ans << endl;
31 }
32 int main() {
33     ios::sync_with_stdio(false);
34     cin.tie(nullptr);
35     int t;
36     cin >> t;
37     while (t--) {
38         solve();
39     }
40     return 0;
41 }

```

### 533: Bertown roads

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Bertown has  $n$  junctions and  $m$  bidirectional roads. We know that one can get from any junction to any other one by the existing roads.

As there were more and more cars in the city, traffic jams started to pose real problems. To deal with them the government decided to make the traffic one-directional on all the roads, thus easing down the traffic. Your task is to determine whether there is a way to make the traffic one-directional so that there still is the possibility to get from any junction to any other one. If the answer is positive, you should also find one of the possible ways to orient the roads.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 set<pair<int, int>> E;
5 # struct EBCC;
6 int main() {
7     ios::sync_with_stdio(false);
8     cin.tie(nullptr);
9     int n, m;
10    cin >> n >> m;
11    EBCC g(n);
12    for (int i = 0; i < m; i++) {
13        int u, v;
14        cin >> u >> v;
15        u--, v--;
16        g.addEdge(u, v);
17    }
18    auto res = g.work();
19    if (count(res.begin(), res.end(), 0) < n) {
20        cout << 0 << "\n";
21        return 0;
22    }
23    for (auto [x, y] : E) {
24        cout << x + 1 << " " << y + 1 << "\n";
25    }
26    return 0;
27 }
```

### 534: Rectangle Puzzle

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given two rectangles on a plane. The centers of both rectangles are located in the origin of coordinates (meaning the center of the rectangle's symmetry). The first rectangle's sides are parallel to the coordinate axes: the length of the side that is parallel to the Ox axis, equals w, the length of the side that is parallel to the Oy axis, equals h. The second rectangle can be obtained by rotating the first rectangle relative to the origin of coordinates by angle .

Your task is to find the area of the region which belongs to both given rectangles. This region is shaded in the picture.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr double Pi = numbers::pi;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int w, h, a;
9     cin >> w >> h >> a;
10    if (w < h) {
11        swap(w, h);
12    }
13    if (a > 90) {
14        a = 180 - a;
15    }
16    double ang = Pi * a / 180;
17    cout << fixed << setprecision(10);
18    double ans;
19    double t = atan(1. * h / w);
20    double e = sqrt(1. * h * h + 1. * w * w);
21    if (ang < 2 * t) {
22        ans = 1. * w * h;
23        if (ang > 0) {
24            double r = e / 2 * sin(ang + t) - h / 2.;
25            ans -= r * r / sin(ang) / cos(ang);
26            r = e / 2 * sin(Pi / 2 - ang + t) - w / 2.;
27            ans -= r * r / sin(ang) / cos(ang);
28        }
29    } else {
30        ans = 1. * h * h / sin(ang);
31    }
32    cout << ans << "\n";
33    return 0;
34 }
```

### 535: Area of Two Circles' Intersection

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given two circles. Find the area of their intersection.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr long double Pi = numbers::pi;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     long double x1, y1, r1;
9     cin >> x1 >> y1 >> r1;
```

```

10     long double x2, y2, r2;
11     cin >> x2 >> y2 >> r2;
12     long double d = sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
13     cout << fixed << setprecision(10);
14     if (d >= r1+r2) {
15         cout << 0. << "\n";
16         return 0;
17     }
18     if (r1+d <= r2) {
19         cout << Pi * r1 * r1 << "\n";
20         return 0;
21     }
22     if (r2+d <= r1) {
23         cout << Pi * r2 * r2 << "\n";
24         return 0;
25     }
26     long double ans = 0;
27     long double ang1 = acos((r1*r1 + d*d - r2*r2) / (2*r1*d));
28     ans += ang1 * r1 * r1;
29     ans -= r1*sin(ang1) * r1*cos(ang1);
30     long double ang2 = acos((r2*r2 + d*d - r1*r1) / (2*r2*d));
31     ans += ang2 * r2 * r2;
32     ans -= r2*sin(ang2) * r2*cos(ang2);
33     cout << ans << "\n";
34 }
35 }
```

### 536: Smart Boy

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Once Petya and Vasya invented a new game and called it “Smart Boy”. They located a certain set of words - the dictionary - for the game. It is admissible for the dictionary to contain similar words.

The rules of the game are as follows: first the first player chooses any letter (a word as long as 1) from any word from the dictionary and writes it down on a piece of paper. The second player adds some other letter to this one’s initial or final position, thus making a word as long as 2, then it’s the first player’s turn again, he adds a letter in the beginning or in the end thus making a word as long as 3 and so on. But the player mustn’t break one condition: the newly created word must be a substring of a word from a dictionary. The player who can’t add a letter to the current word without breaking the condition loses.

Also if by the end of a turn a certain string  $s$  is written on paper, then the player, whose turn it just has been, gets a number of points according to the formula:

where

$c$  is a sequence number of symbol  $c$  in Latin alphabet, numbered starting from 1. For example, , and .

is the number of words from the dictionary where the line  $s$  occurs as a substring at least once.

Your task is to learn who will win the game and what the final score will be. Every player plays optimally and most of all tries to win, then - to maximize the number of his points, then - to minimize the number of the points of the opponent.

Problem: [link](#)

Solution: [link](#)

```

1 n = int(input())
2 num = dict()
3 for _ in range(n) :
4     s = input()
5     sub = set()
6     for i in range(len(s)) :
7         for j in range(i, len(s)) :
8             sub.add(s[i:j+1])
9     for s in sub :
10        if s in num :
11            num[s] += 1
12        else :
13            num[s] = 1
14 fs = dict()
15 def f(s) :
16     global fs
17     if s in fs :
18         return fs[s]
19     ans = (False, 0, 0)
20     for x in range(26) :
21         c = chr(x + ord('a'))
22         for t in [c + s, s + c] :
23             if t in num :
24                 sm = 0
25                 mx = 0
26                 for c in t :
27                     sm += ord(c) - ord('a') + 1
28                     mx = max(mx, ord(c) - ord('a') + 1)
29                 (win, a, b) = f(t)
30                 ans = max(ans, (not win, -b + sm * mx + num[t], -a));
31     fs[s] = ans
32     return ans
33 (win, a, b) = f('')
34 if win :
35     print('First')
36 else :
37     print('Second')
38 print(a, -b)
```

### 537: Moon Craters

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are lots of theories concerning the origin of moon craters. Most scientists stick to the meteorite theory, which says that the craters were formed as a result of celestial bodies colliding with the Moon. The other version is that the craters were parts of volcanoes.

An extraterrestrial intelligence research specialist professor Okulov (the namesake of the Okulov, the author of famous textbooks on programming) put forward an alternate hypothesis. Guess what kind of a hypothesis it was – sure, the one including extraterrestrial mind involvement. Now the professor is looking for proofs of his hypothesis.

Professor has data from the moon robot that moves linearly in one direction along the Moon surface. The moon craters are circular in form with integer-valued radii. The moon robot records only the craters whose centers lay on his path and sends to the Earth the information on the distance from the centers of the craters to the initial point of its path and on the radii of the craters.

According to the theory of professor Okulov two craters made by an extraterrestrial intelligence for the aims yet unknown either are fully enclosed one in the other or do not intersect at all. Internal or external tangency is acceptable. However the experimental data from the moon robot do not confirm this theory! Nevertheless, professor Okulov is hopeful. He perfectly understands that to create any logical theory one has to ignore some data that are wrong due to faulty measuring (or skillful disguise by the extraterrestrial intelligence that will be sooner or later found by professor Okulov!) Thats why Okulov wants to choose among the available crater descriptions the largest set that would satisfy his theory.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> c(n), r(n);
10    for (int i = 0; i < n; i++) {
11        cin >> c[i] >> r[i];
12    }
13    vector<int> ord(n);
14    iota(ord.begin(), ord.end(), 0);
15    sort(ord.begin(), ord.end(), [&](int i, int j) {
16        return r[i] < r[j];
17    });
18    vector<int> dp(n);
19    vector<vector<int>> sub(n);
20    vector<int> g(n), lst(n);
21    for (auto i : ord) {
22        vector<tuple<int, int, int>> e;
23        for (auto j : ord) {

```

```
24         if (c[j] - r[j] >= c[i] - r[i] && c[j] + r[j] <= c[i] + r[i]) {
25             e.emplace_back(c[j] - r[j], 1, j);
26             e.emplace_back(c[j] + r[j], -1, j);
27         }
28     }
29     sort(e.begin(), e.end());
30     int v = 0;
31     int u = -1;
32     for (auto [x, t, j] : e) {
33         if (t == 1) {
34             g[j] = v + dp[j];
35             lst[j] = u;
36         } else if (g[j] > v) {
37             v = g[j];
38             u = j;
39         }
40     }
41     dp[i] = v + 1;
42     while (u != -1) {
43         sub[i].push_back(u);
44         u = lst[u];
45     }
46 }
47 vector<tuple<int, int, int>> e;
48 for (auto j : ord) {
49     e.emplace_back(c[j] - r[j], 1, j);
50     e.emplace_back(c[j] + r[j], -1, j);
51 }
52 sort(e.begin(), e.end());
53 int v = 0;
54 int u = -1;
55 for (auto [x, t, j] : e) {
56     if (t == 1) {
57         g[j] = v + dp[j];
58         lst[j] = u;
59     } else if (g[j] > v) {
60         v = g[j];
61         u = j;
62     }
63 }
64 int ans = v;
65 cout << ans << "\n";
66 queue<int> q;
67 while (u != -1) {
68     q.push(u);
69     u = lst[u];
70 }
71 while (!q.empty()) {
72     int x = q.front();
73     q.pop();
74     cout << x + 1 << " ";
75     for (auto y : sub[x]) {
76         q.push(y);
77     }
78 }
79 cout << "\n";
80 return 0;
81 }
```

### 538: What Has Dirichlet Got to Do with That?

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

You all know the Dirichlet principle, the point of which is that if  $n$  boxes have no less than  $n + 1$  items, that leads to the existence of a box in which there are at least two items.

Having heard of that principle, but having not mastered the technique of logical thinking, 8 year olds Stas and Masha invented a game. There are  $a$  different boxes and  $b$  different items, and each turn a player can either add a new box or a new item. The player, after whose turn the number of ways of putting  $b$  items into  $a$  boxes becomes no less than a certain given number  $n$ , loses. All the boxes and items are considered to be different. Boxes may remain empty.

Who loses if both players play optimally and Stas's turn is first?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <iostream>
2 #include <map>
3 #include <cmath>
4 using namespace std;
5 int a, b, n;
6 map<pair<int,int>, int> f;
7 int win(int a, int b) {
8     if (a > 1 && (b >= 30 || pow(a, b) >= n)) {
9         return 1;
10    }
11    if (a == 1 && (b >= 30 || pow(2, b) >= n)) {
12        return 0;
13    }
14    if (b == 1 && pow(a, 2) >= n) {
15        if ((n - a) % 2 == 0) {
16            return 1;
17        } else {
18            return -1;
19        }
20    }
21    auto key = make_pair(a, b);
22    if (f.find(key) != f.end()) {
23        return f[key];
24    }
25    f[key] = max(-win(a + 1, b), -win(a, b + 1));
26    return f[key];
27 }
28 int main() {
29     cin >> a >> b >> n;
30     int ans = win(a, b);
31     if (ans == 1) {

```

```

32     cout << "Masha" << endl;
33 } else if (ans == -1) {
34     cout << "Stas" << endl;
35 } else {
36     cout << "Missing" << endl;
37 }
38 return 0;
39 }
```

**539: Ant on the Tree**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Connected undirected graph without cycles is called a tree. Trees is a class of graphs which is interesting not only for people, but for ants too.

An ant stands at the root of some tree. He sees that there are  $n$  vertexes in the tree, and they are connected by  $n - 1$  edges so that there is a path between any pair of vertexes. A leaf is a distinct from root vertex, which is connected with exactly one other vertex.

The ant wants to visit every vertex in the tree and return to the root, passing every edge twice. In addition, he wants to visit the leaves in a specific order. You are to find some possible route of the ant.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<vector<int>> adj(n);
10    for (int i = 1; i < n; i++) {
11        int u, v;
12        cin >> u >> v;
13        u--, v--;
14        adj[u].push_back(v);
15        adj[v].push_back(u);
16    }
17    vector<int> leaf(n);
18    int cnt = 0;
19    function<void(int, int)> dfs = [&](int x, int p) {
20        leaf[x] = x;
21        for (auto y : adj[x]) {
22            if (y == p) {
```

```

23         continue;
24     }
25     dfs(y, x);
26     leaf[x] = leaf[y];
27 }
28 if (leaf[x] == x) {
29     cnt++;
30 }
31 };
32 dfs(0, -1);
33 vector<int> a(cnt), pos(n);
34 for (int i = 0; i < cnt; i++) {
35     cin >> a[i];
36     a[i]--;
37     pos[a[i]] = i;
38 }
39 vector<int> ans, b;
40 dfs = [&](int x, int p) {
41     ans.push_back(x);
42     sort(adj[x].begin(), adj[x].end(), [&](int x, int y) {
43         return pos[leaf[x]] < pos[leaf[y]];
44     });
45     for (auto y : adj[x]) {
46         if (y == p) {
47             continue;
48         }
49         dfs(y, x);
50         ans.push_back(x);
51     }
52     if (leaf[x] == x) {
53         b.push_back(x);
54     }
55 };
56 dfs(0, -1);
57 if (a != b) {
58     cout << -1 << "\n";
59     return 0;
60 }
61 for (auto x : ans) {
62     cout << x + 1 << " ";
63 }
64 cout << "\n";
65 return 0;
66 }

```

## 540: Thanos Nim

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Alice and Bob are playing a game with  $n$  piles of stones. It is guaranteed that  $n$  is an even number. The  $i$ -th pile has  $a_i$  stones.

Alice and Bob will play a game alternating turns with Alice going first.

On a player's turn, they must choose exactly  $\frac{n}{2}$  nonempty piles and independently remove a positive number of stones from each of the chosen piles. They can remove a different number of stones from the piles in a single turn. The first player unable to make a move loses (when there are less than  $\frac{n}{2}$  nonempty piles).

Given the starting configuration, determine who will win the game.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    sort(a.begin(), a.end());
14    if (count(a.begin(), a.begin() + n / 2 + 1, a[0]) == n / 2 + 1) {
15        cout << "Bob\n";
16    } else {
17        cout << "Alice\n";
18    }
19    return 0;
20 }
```

## 541: Stripe 2

- Time limit: 1 second
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

Once Bob took a paper stripe of  $n$  squares (the height of the stripe is 1 square). In each square he wrote an integer number, possibly negative. He became interested in how many ways exist to cut this stripe into three pieces so that the sum of numbers from each piece is equal to the sum of numbers from any other piece, and each piece contains positive integer amount of squares. Would you help Bob solve this problem?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n), s(n + 1);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12        s[i + 1] = s[i] + a[i];
13    }
14    if (s[n] % 3 != 0) {
15        cout << 0 << "\n";
16        return 0;
17    }
18    i64 ans = 0;
19    int cnt = 0;
20    for (int i = 1; i < n; i++) {
21        if (s[i] == s[n] / 3 * 2) {
22            ans += cnt;
23        }
24        if (s[i] == s[n] / 3) {
25            cnt += 1;
26        }
27    }
28    cout << ans << "\n";
29    return 0;
30 }

```

## 542: Flag 2

- Time limit: 2 seconds
- Memory limit: 128 megabytes
- Input file: standard input
- Output file: standard output

According to a new ISO standard, a flag of every country should have, strangely enough, a chequered field  $n \times m$ , each square should be wholly painted one of 26 colours. The following restrictions are set:

In each row at most two different colours can be used.

No two adjacent squares can be painted the same colour.

Pay attention, please, that in one column more than two different colours can be used.

Berland's government took a decision to introduce changes into their country's flag in accordance with the new standard, at the same time they want these changes to be minimal. By the given description of Berland's flag you should find out the minimum amount of squares that need to be painted different colour to make the flag meet the new ISO standard. You are as well to build one of the possible variants of the new Berland's flag.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int inf = 1E9;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, m;
9     cin >> n >> m;
10    vector<string> s(n);
11    for (int i = 0; i < n; i++) {
12        cin >> s[i];
13    }
14    vector dp(n + 1, vector(26, vector(26, inf)));
15    vector pre(n + 1, vector(26, vector(26, pair{-1, -1})));
16    dp[0].assign(26, vector(26, 0));
17    for (int i = 0; i < n; i++) {
18        for (int x = 0; x < 26; x++) {
19            for (int y = 0; y < 26; y++) {
20                if (x == y) {
21                    continue;
22                }
23                int cost = 0;
24                for (int j = 0; j < m; j++) {
25                    cost += s[i][j] - 'a' != (j % 2 ? y : x);
26                }
27                for (int a = 0; a < 26; a++) {
28                    for (int b = 0; b < 26; b++) {
29                        if (a != x && b != y && dp[i][a][b] + cost < dp[i + 1][x][y]) {
30                            dp[i + 1][x][y] = dp[i][a][b] + cost;
31                            pre[i + 1][x][y] = {a, b};
32                        }
33                    }
34                }
35            }
36        }
37    }
38    int ans = inf;
39    int x = -1, y = -1;
40    for (int a = 0; a < 26; a++) {
41        for (int b = 0; b < 26; b++) {
42            if (dp[n][a][b] < ans) {
43                ans = dp[n][a][b];
44                x = a;
45                y = b;
46            }
47        }
48    }
49    cout << ans << "\n";
50    for (int i = n - 1; i >= 0; i--) {
51        for (int j = 0; j < m; j++) {
52            s[i][j] = 'a' + (j % 2 ? y : x);
53        }
54        tie(x, y) = pre[i + 1][x][y];
55    }
56    for (int i = 0; i < n; i++) {
57        cout << s[i] << "\n";
58    }
}

```

```

59     return 0;
60 }
```

**543: Start of the season**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Before the start of the football season in Berland a strange magic ritual is held. The most experienced magicians have to find a magic matrix of the size  $n \times n$  ( $n$  is even number). Gods will never allow to start the championship without it. Matrix should contain integers from 0 to  $n - 1$ , main diagonal should contain only zeroes and matrix should be symmetric. Moreover, all numbers in each row should be different. Magicians are very tired of the thinking process, so they ask you to write a program to find such matrix.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector a(n, vector<int>(n));
10    for (int x = 0; x < n - 1; x++) {
11        for (int j = 0; j < n - 1; j++) {
12            int k = (x - j + n - 1) % (n - 1);
13            if (j == k) {
14                a[j][n - 1] = a[n - 1][j] = x + 1;
15            } else {
16                a[j][k] = a[k][j] = x + 1;
17            }
18        }
19    }
20    for (int i = 0; i < n; i++) {
21        for (int j = 0; j < n; j++) {
22            cout << a[i][j] << " \n"[j == n - 1];
23        }
24    }
25    return 0;
26 }
```

**544: The hat**

- Time limit: 1 second

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an interactive problem.

Imur Ishakov decided to organize a club for people who love to play the famous game The hat. The club was visited by  $n$  students, where  $n$  is even. Imur arranged them all in a circle and held a draw to break the students in pairs, but something went wrong. The participants are numbered so that participant  $i$  and participant  $i + 1$  ( $1 \leq i \leq n - 1$ ) are adjacent, as well as participant  $n$  and participant 1. Each student was given a piece of paper with a number in such a way, that for every two adjacent students, these numbers differ exactly by one. The plan was to form students with the same numbers in a pair, but it turned out that not all numbers appeared exactly twice.

As you know, the most convenient is to explain the words to the partner when he is sitting exactly across you. Students with numbers  $i$  and sit across each other. Imur is wondering if there are two people sitting across each other with the same numbers given. Help him to find such pair of people if it exists.

You can ask questions of form which number was received by student  $i$ ?, and the goal is to determine whether the desired pair exists in no more than 60 questions.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int n;
5 int query(int x) {
6     cout << "? " << x % n + 1 << endl;
7     int ans;
8     cin >> ans;
9     return ans;
10 }
11 int main() {
12     ios::sync_with_stdio(false);
13     cin.tie(nullptr);
14     cin >> n;
15     int lo = 0, hi = n / 2;
16     int flo = query(0) - query(n / 2);
17     int fhi = -flo;
18     if (flo % 2 != 0) {
19         cout << "! " << -1 << endl;
20         return 0;
21     }
22     if (flo == 0) {
23         cout << "! " << 1 << endl;
24         return 0;
25 }
```

```

26     while (true) {
27         int x = (lo + hi) / 2;
28         int fx = query(x) - query(x + n / 2);
29         if (fx == 0) {
30             cout << "!" << x + 1 << endl;
31             return 0;
32         }
33         if ((fx > 0) == (flo > 0)) {
34             lo = x;
35             flo = fx;
36         } else {
37             hi = x;
38             fhi = fx;
39         }
40     }
41     return 0;
42 }
```

## 545: Lucky Sorting

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Petya loves lucky numbers. We all know that lucky numbers are the positive integers whose decimal representations contain only the lucky digits 4 and 7. For example, numbers 47, 744, 4 are lucky and 5, 17, 467 are not.

Petya got an array consisting of  $n$  numbers, it is the gift for his birthday. Now he wants to sort it in the non-decreasing order. However, a usual sorting is boring to perform, that's why Petya invented the following limitation: one can swap any two numbers but only if at least one of them is lucky. Your task is to sort the array according to the specified limitation. Find any possible sequence of the swaps (the number of operations in the sequence should not exceed  $2n$ ).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n;
9     cin >> n;
10    vector<int> a(n);
11    for (int i = 0; i < n; i++) {
```

```

12         cin >> a[i];
13     }
14     if (is_sorted(a.begin(), a.end())) {
15         cout << 0 << "\n";
16         return 0;
17     }
18     vector<int> q(n);
19     iota(q.begin(), q.end(), 0);
20     sort(q.begin(), q.end(), [&](int i, int j) {
21         return a[i] < a[j];
22     });
23     vector<int> p(n), invp(n);
24     for (int i = 0; i < n; i++) {
25         p[q[i]] = i;
26         invp[i] = q[i];
27     }
28     int x = -1;
29     for (int i = 0; i < n; i++) {
30         bool ok = true;
31         string s = to_string(a[i]);
32         for (auto c : s) {
33             if (c != '4' && c != '7') {
34                 ok = false;
35             }
36         }
37         if (ok) {
38             x = i;
39             break;
40         }
41     }
42     if (x == -1) {
43         cout << -1 << "\n";
44         return 0;
45     }
46     iota(q.begin(), q.end(), 0);
47     iota(a.begin(), a.end(), 0);
48     DSU dsu(n);
49     for (int i = 0; i < n; i++) {
50         dsu.merge(i, p[i]);
51     }
52     vector<pair<int, int>> ans;
53     auto swap = [&](int x, int y) {
54         ans.emplace_back(q[x] + 1, q[y] + 1);
55         swap(q[x], q[y]);
56         swap(a[q[x]], a[q[y]]);
57         swap(p[q[x]], p[q[y]]);
58         swap(invp[p[q[x]]], invp[p[q[y]]]);
59     };
60     for (int i = 0; i < n; i++) {
61         if (!dsu.same(x, i)) {
62             swap(x, i);
63             dsu.merge(x, i);
64         }
65     }
66     while (p[q[x]] != q[x]) {
67         swap(x, a[invp[q[x]]]);
68     }
69     cout << ans.size() << "\n";
70     for (auto [x, y] : ans) {
71         cout << x << " " << y << "\n";
72     }
73     return 0;
74 }
```

## 546: Work Group

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

One Big Software Company has  $n$  employees numbered from 1 to  $n$ . The director is assigned number 1. Every employee of the company except the director has exactly one immediate superior. The director, of course, doesn't have a superior.

We will call person  $a$  a subordinates of another person  $b$ , if either  $b$  is an immediate supervisor of  $a$ , or the immediate supervisor of  $a$  is a subordinate to person  $b$ . In particular, subordinates of the head are all other employees of the company.

To solve achieve an Important Goal we need to form a workgroup. Every person has some efficiency, expressed by a positive integer  $a_i$ , where  $i$  is the person's number. The efficiency of the workgroup is defined as the total efficiency of all the people included in it.

The employees of the big software company are obsessed with modern ways of work process organization. Today pair programming is at the peak of popularity, so the workgroup should be formed with the following condition. Each person entering the workgroup should be able to sort all of his subordinates who are also in the workgroup into pairs. In other words, for each of the members of the workgroup the number of his subordinates within the workgroup should be even.

Your task is to determine the maximum possible efficiency of the workgroup formed at observing the given condition. Any person including the director of company can enter the workgroup.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> p(n), a(n);
10    vector<vector<int>> adj(n);
11    for (int i = 0; i < n; i++) {
12        cin >> p[i] >> a[i];
13        if (i > 0) {
14            adj[p[i] - 1].push_back(i);
15        }
16    }
17    vector<array<i64, 2>> dp(n);
18    function<void(int)> dfs = [&](int x) {
19        array<i64, 2> g{0, -i64(1E18)};

```

```

20         for (auto y : adj[x]) {
21             dfs(y);
22             array<i64, 2> ng{-i64(1E18), -i64(1E18)};
23             for (int i = 0; i < 2; i++) {
24                 ng[i] = max(ng[i], g[i] + dp[y][0]);
25                 ng[i ^ 1] = max(ng[i ^ 1], g[i] + dp[y][1]);
26                 ng[i ^ 1] = max(ng[i ^ 1], g[i] + dp[y][0] + a[y]);
27             }
28             g = ng;
29         }
30         dp[x] = g;
31     };
32     dfs(0);
33     i64 ans = max(dp[0][0] + a[0], dp[0][1]);
34     cout << ans << "\n";
35     return 0;
36 }
```

## 547: Graph Cost

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an initially empty undirected graph with  $n$  nodes, numbered from 1 to  $n$  (i. e.  $n$  nodes and 0 edges). You want to add  $m$  edges to the graph, so the graph won't contain any self-loop or multiple edges.

If an edge connecting two nodes  $u$  and  $v$  is added, its weight must be equal to the greatest common divisor of  $u$  and  $v$ , i. e.  $\gcd(u, v)$ .

In order to add edges to the graph, you can repeat the following process any number of times (possibly zero):

choose an integer  $k \geq 1$ ;

add exactly  $k$  edges to the graph, each having a weight equal to  $k + 1$ . Adding these  $k$  edges costs  $k + 1$  in total.

For example, if you can add 5 more edges to the graph of weight 6, you may add them, and it will cost 6 for the whole pack of 5 edges. But if you can only add 4 edges of weight 6 to the graph, you can't perform this operation for  $k = 5$ .

Given two integers  $n$  and  $m$ , find the minimum total cost to form a graph of  $n$  vertices and exactly  $m$  edges using the operation above. If such a graph can't be constructed, output  $-1$ .

Note that the final graph may consist of several connected components.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     i64 m;
7     cin >> n >> m;
8     i64 ans = m;
9     vector<i64> f(n + 1);
10    for (int i = 1; i <= n; i++) {
11        f[i] = 1LL * (n / i) * (n / i);
12    }
13    for (int i = n; i; i--) {
14        for (int j = 2 * i; j <= n; j += i) {
15            f[i] -= f[j];
16        }
17    }
18    for (int i = 1; i <= n; i++) {
19        f[i]--;
20        f[i] /= 2;
21    }
22    for (int i = n; i >= 2; i--) {
23        i64 t = min(f[i], m) / (i - 1);
24        m -= t * (i - 1);
25        ans += t;
26    }
27    if (m) ans = -1;
28    cout << ans << "\n";
29 }
30 int main() {
31     ios::sync_with_stdio(false);
32     cin.tie(nullptr);
33     int t;
34     cin >> t;
35     while (t--) {
36         solve();
37     }
38     return 0;
39 }
```

## 548: GCD Queries

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an interactive problem.

There is a secret permutation  $p$  of  $[0, 1, 2, \dots, n - 1]$ . Your task is to find 2 indices  $x$  and  $y$  ( $1 \leq x, y \leq n$ , possibly  $x = y$ ) such that  $p_x = 0$  or  $p_y = 0$ . In order to find it, you are allowed to ask at most  $2n$  queries.

In one query, you give two integers  $i$  and  $j$  ( $1 \leq i, j \leq n, i \neq j$ ) and receive the value of  $\gcd(p_i, p_j)$ <sup>†</sup>.

Note that the permutation  $p$  is fixed before any queries are made and does not depend on the queries.

$\dagger$   $\gcd(x, y)$  denotes the greatest common divisor (GCD) of integers  $x$  and  $y$ . Note that  $\gcd(x, 0) = \gcd(0, x) = x$  for all positive integers  $x$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int query(int x, int y) {
5     cout << "? " << x + 1 << " " << y + 1 << endl;
6     int res;
7     cin >> res;
8     return res;
9 }
10 void solve() {
11     int n;
12     cin >> n;
13     int x = 0, y = 1;
14     int g = query(x, y);
15     for (int i = 2; i < n; i++) {
16         int gx = query(i, x);
17         int gy = query(i, y);
18         if (gx >= max(gy, g)) {
19             y = i;
20             g = gx;
21         } else if (gy >= max(gx, g)) {
22             x = i;
23             g = gy;
24         }
25     }
26     cout << "! " << x + 1 << " " << y + 1 << endl;
27     int res;
28     cin >> res;
29 }
30 int main() {
31     ios::sync_with_stdio(false);
32     cin.tie(nullptr);
33     int t;
34     cin >> t;
35     while (t--) {
36         solve();
37     }
38     return 0;
39 }
```

## 549: Carry Bit

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Let  $f(x, y)$  be the number of carries of  $x + y$  in binary (i. e.  $f(x, y) = g(x) + g(y) - g(x + y)$ , where  $g(x)$  is the number of ones in the binary representation of  $x$ ).

Given two integers  $n$  and  $k$ , find the number of ordered pairs  $(a, b)$  such that  $0 \leq a, b < 2^n$ , and  $f(a, b)$  equals  $k$ . Note that for  $a \neq b$ ,  $(a, b)$  and  $(b, a)$  are considered as two different pairs.

As this number may be large, output it modulo  $10^9 + 7$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 1000000007;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
8     if (x < 0) {
9         x += P;
10    }
11    if (x >= P) {
12        x -= P;
13    }
14    return x;
15}
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25}
26 # struct Z;
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(nullptr);
30     int n, k;
31     cin >> n >> k;
32     vector<Z> fac(n + 1), invfac(n + 1);
33     fac[0] = 1;
34     for (int i = 1; i <= n; i++) {
35         fac[i] = fac[i - 1] * i;
36     }
37     invfac[n] = fac[n].inv();
38     for (int i = n; i; i--) {
39         invfac[i - 1] = invfac[i] * i;
40     }
41     if (k == 0) {
42         cout << power(Z(3), n) << "\n";
43         return 0;
44     }
45     auto binom = [&](int n, int m) -> Z {
46         if (n < m || m < 0) {
47             return 0;
48         }
49         return fac[n] * invfac[m] * invfac[n - m];
50     };

```

```

51     Z ans = 0;
52     vector<Z> p(n + 1);
53     p[0] = 1;
54     for (int i = 1; i <= n; i++) {
55         p[i] = p[i - 1] * 3;
56     }
57     for (int i = 1; i <= k; i++) {
58         if (2 * i <= n + 1) {
59             ans += binom(k - 1, i - 1) * binom(n - k, i - 1) * p[n - i - i + 1];
60         }
61         if (2 * i <= n) {
62             ans += binom(k - 1, i - 1) * binom(n - k, i) * p[n - i - i];
63         }
64     }
65     cout << ans << "\n";
66 }
67 }
```

## 551: Wish I Knew How to Sort

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a binary array  $a$  (all elements of the array are 0 or 1) of length  $n$ . You wish to sort this array, but unfortunately, your algorithms teacher forgot to teach you sorting algorithms. You perform the following operations until  $a$  is sorted:

Choose two random indices  $i$  and  $j$  such that  $i < j$ . Indices are chosen equally probable among all pairs of indices  $(i, j)$  such that  $1 \leq i < j \leq n$ .

If  $a_i > a_j$ , then swap elements  $a_i$  and  $a_j$ .

What is the expected number of such operations you will perform before the array becomes sorted?

It can be shown that the answer can be expressed as an irreducible fraction  $\frac{p}{q}$ , where  $p$  and  $q$  are integers and  $q \not\equiv 0 \pmod{998\,244\,353}$ . Output the integer equal to  $p \cdot q^{-1} \pmod{998\,244\,353}$ . In other words, output such an integer  $x$  that  $0 \leq x < 998\,244\,353$  and  $x \cdot q \equiv p \pmod{998\,244\,353}$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 998244353;
5 using i64 = long long;
6 // assume -P <= x < 2P
```

```

7  int norm(int x) {
8      if (x < 0) {
9          x += P;
10     }
11     if (x >= P) {
12         x -= P;
13     }
14     return x;
15 }
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25 }
26 # struct Z;
27 void solve() {
28     int n;
29     cin >> n;
30     vector<int> a(n);
31     for (int i = 0; i < n; i++) {
32         cin >> a[i];
33     }
34     int cnt = count(a.begin(), a.end(), 0);
35     int bad = count(a.begin(), a.begin() + cnt, 1);
36     Z ans = 0;
37     for (int i = 1; i <= bad; i++) {
38         ans += Z(1) / i / i;
39     }
40     ans *= Z(n) * (n - 1) / 2;
41     cout << ans << "\n";
42 }
43 int main() {
44     ios::sync_with_stdio(false);
45     cin.tie(nullptr);
46     int t;
47     cin >> t;
48     while (t--) {
49         solve();
50     }
51     return 0;
52 }
```

## 552: MEX vs MED

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a permutation  $p_1, p_2, \dots, p_n$  of length  $n$  of numbers  $0, \dots, n - 1$ . Count the number of subsegments  $1 \leq l \leq r \leq n$  of this permutation such that  $\text{mex}(p_l, p_{l+1}, \dots, p_r) > \text{med}(p_l, p_{l+1}, \dots, p_r)$ .

*mex* of  $S$  is the smallest non-negative integer that does not occur in  $S$ . For example:

$$\text{mex}(0, 1, 2, 3) = 4$$

$$\text{mex}(0, 4, 1, 3) = 2$$

$$\text{mex}(5, 4, 0, 1, 2) = 3$$

*med* of the set  $S$  is the median of the set, i.e. the element that, after sorting the elements in non-decreasing order, will be at position number  $\left\lfloor \frac{|S|+1}{2} \right\rfloor$  (array elements are numbered starting from 1 and here  $\lfloor v \rfloor$  denotes rounding  $v$  down.). For example:

$$\text{med}(0, 1, 2, 3) = 1$$

$$\text{med}(0, 4, 1, 3) = 1$$

$$\text{med}(5, 4, 0, 1, 2) = 2$$

A sequence of  $n$  numbers is called a permutation if it contains all the numbers from 0 to  $n - 1$  exactly once.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> p(n);
8     for (int i = 0; i < n; i++) {
9         cin >> p[i];
10    }
11    vector<int> pre(n + 1), suf(n + 1);
12    pre[0] = suf[n] = n;
13    for (int i = 0; i < n; i++) {
14        pre[i + 1] = min(pre[i], p[i]);
15    }
16    for (int i = n - 1; i >= 0; i--) {
17        suf[i] = min(suf[i + 1], p[i]);
18    }
19    i64 ans = 1;
20    int l = 0, r = n;
21    while (r - l > 1) {
22        if (pre[l + 1] > suf[r - 1]) {
23            l++;
24            ans += max(0, min(n - l, pre[l] * 2) - (r - l) + 1);
25        } else {
26            r--;
27            ans += max(0, min(r, suf[r] * 2) - (r - l) + 1);
28        }
29    }
30    cout << ans << "\n";
31}
32 int main() {
```

```

33     ios::sync_with_stdio(false);
34     cin.tie(nullptr);
35     int t;
36     cin >> t;
37     while (t--) {
38         solve();
39     }
40     return 0;
41 }
```

## orange

### brute force

#### 553: Lights

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In the end of the day, Anna needs to turn off the lights in the office. There are  $n$  lights and  $n$  light switches, but their operation scheme is really strange. The switch  $i$  changes the state of light  $i$ , but it also changes the state of some other light  $a_i$  (change the state means that if the light was on, it goes off and vice versa).

Help Anna to turn all the lights off using minimal number of switches, or say it is impossible.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     string s;
8     cin >> s;
9     vector<int> a(n);
10    vector<int> deg(n);
11    for (int i = 0; i < n; i++) {
12        cin >> a[i];
13        a[i] -= 1;
14        deg[a[i]] += 1;
15    }
16    queue<int> q;
17    for (int i = 0; i < n; i++) {
```

```
18         if (deg[i] == 0) {
19             q.push(i);
20         }
21     }
22     vector<int> ans;
23     while (!q.empty()) {
24         int x = q.front();
25         q.pop();
26         if (s[x] == '1') {
27             s[x] = '0';
28             s[a[x]] ^= 1;
29             ans.push_back(x);
30         }
31         if (--deg[a[x]] == 0) {
32             q.push(a[x]);
33         }
34     }
35     for (int i = 0; i < n; i++) {
36         if (deg[i]) {
37             int j = i;
38             int t = 0;
39             int len = 0;
40             int res = 0;
41             while (deg[j]) {
42                 if (s[j] == '1') {
43                     t ^= 1;
44                 }
45                 res += t;
46                 deg[j] = 0;
47                 j = a[j];
48                 len += 1;
49             }
50             if (t == 1) {
51                 cout << -1 << "\n";
52                 return;
53             }
54             for (int k = 0; k < len; k++) {
55                 if (s[j] == '1') {
56                     t ^= 1;
57                 }
58                 if (t == (res < len - res)) {
59                     ans.push_back(j);
60                 }
61                 j = a[j];
62             }
63         }
64     }
65     cout << ans.size() << "\n";
66     for (auto x : ans) {
67         cout << x + 1 << " \n"[x == ans.back()];
68     }
69 }
70 int main() {
71     ios::sync_with_stdio(false);
72     cin.tie(nullptr);
73     int t;
74     cin >> t;
75     while (t--) {
76         solve();
77     }
78     return 0;
79 }
```

## 554: Swapping Characters

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

We had a string  $s$  consisting of  $n$  lowercase Latin letters. We made  $k$  copies of this string, thus obtaining  $k$  identical strings  $s_1, s_2, \dots, s_k$ . After that, in each of these strings we swapped exactly two characters (the characters we swapped could be identical, but they had different indices in the string).

You are given  $k$  strings  $s_1, s_2, \dots, s_k$ , and you have to restore any string  $s$  so that it is possible to obtain these strings by performing aforementioned operations. Note that the total length of the strings you are given doesn't exceed 5000 (that is,  $k \cdot n \leq 5000$ ).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int k, n;
8     cin >> k >> n;
9     vector<string> s(k);
10    for (int i = 0; i < k; i++) {
11        cin >> s[i];
12    }
13    vector<array<int, 26>> cnt(k);
14    for (int i = 0; i < k; i++) {
15        for (auto c : s[i]) {
16            cnt[i][c - 'a']++;
17        }
18    }
19    for (int i = 0; i < k; i++) {
20        if (cnt[i] != cnt[0]) {
21            cout << -1 << "\n";
22            return 0;
23        }
24    }
25    const int dup = *max_element(cnt[0].begin(), cnt[0].end()) > 1;
26    for (int i = 0; i < k; i++) {
27        if (s[i] != s[0]) {
28            vector<int> p;
29            for (int j = 0; j < n; j++) {
30                if (s[0][j] != s[i][j]) {
31                    p.push_back(j);
32                }
33            }
34            if (p.size() > 4) {
35                cout << -1 << "\n";

```

```

36         return 0;
37     }
38     for (auto x : p) {
39         for (int y = 0; y < n; y++) {
40             if (x != y) {
41                 auto t = s[0];
42                 swap(t[x], t[y]);
43                 bool ok = true;
44                 for (int i = 0; i < k; i++) {
45                     vector<int> diff;
46                     for (int j = 0; j < n; j++) {
47                         if (t[j] != s[i][j]) {
48                             diff.push_back(j);
49                         }
50                     }
51                     if (diff.size() > 2) {
52                         ok = false;
53                         break;
54                     }
55                     if (diff.size() == 0 && !dup) {
56                         ok = false;
57                         break;
58                     }
59                 }
60                 if (ok) {
61                     cout << t << "\n";
62                     return 0;
63                 }
64             }
65         }
66     }
67     cout << -1 << "\n";
68     return 0;
69 }
70 swap(s[0][0], s[0][1]);
71 cout << s[0] << "\n";
72 return 0;
73 }
74 }
```

## 555: Solitaire

- Time limit: 1.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vasya has a pack of 54 cards (52 standard cards and 2 distinct jokers). That is all he has at the moment. Not to die from boredom, Vasya plays Solitaire with them.

Vasya lays out  $nm$  cards as a rectangle  $n \times m$ . If there are jokers among them, then Vasya should change them with some of the rest of 54 -  $nm$  cards (which are not layed out) so that there were no jokers left. Vasya can pick the cards to replace the jokers arbitrarily. Remember, that each card presents in pack exactly once (i. e. in a single copy). Vasya tries to perform the replacements so that the solitaire was solved.

Vasya thinks that the solitaire is solved if after the jokers are replaced, there exist two non-overlapping squares 3 3, inside each of which all the cards either have the same suit, or pairwise different ranks.

Determine by the initial position whether the solitaire can be solved or not. If it can be solved, show the way in which it is possible.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int suitToId(char x) {
5     if (x == 'C') {
6         return 0;
7     } else if (x == 'D') {
8         return 1;
9     } else if (x == 'H') {
10        return 2;
11    } else {
12        return 3;
13    }
14 }
15 int rankToId(char x) {
16     if (x == 'T') {
17         return 8;
18     } else if (x == 'J') {
19         return 9;
20     } else if (x == 'Q') {
21         return 10;
22     } else if (x == 'K') {
23         return 11;
24     } else if (x == 'A') {
25         return 12;
26     } else {
27         return x - '2';
28     }
29 }
30 int cardToId(string s) {
31     if (s[0] == 'J' && isdigit(s[1])) {
32         return 52 + s[1] - '1';
33     } else {
34         return suitToId(s[1]) * 13 + rankToId(s[0]);
35     }
36 }
37 char idToSuit(int x) {
38     return "CDHS"[x];
39 }
40 char idToRank(int x) {
41     return "23456789TJQKA"[x];
42 }
43 string idToCard(int x) {
44     string s;
45     if (x >= 52) {
46         s += 'J';
47         s += '1' + x - 52;
48     } else {
49         s += idToRank(x % 13);
50         s += idToSuit(x / 13);

```

```

51     }
52     return s;
53 }
54 int main() {
55     ios::sync_with_stdio(false);
56     cin.tie(nullptr);
57     int n, m;
58     cin >> n >> m;
59     vector a(n, vector<int>(m));
60     vector vis(54, -1);
61     for (int i = 0; i < n; i++) {
62         for (int j = 0; j < m; j++) {
63             string s;
64             cin >> s;
65             a[i][j] = cardToId(s);
66             vis[a[i][j]] = i * m + j;
67         }
68     }
69     auto check = [&](int j1, int j2) {
70         for (int x1 = 0; x1+3 <= n; x1++) {
71             for (int y1 = 0; y1+3 <= m; y1++) {
72                 for (int x2 = 0; x2+3 <= n; x2++) {
73                     for (int y2 = 0; y2+3 <= m; y2++) {
74                         if (abs(x1 - x2) < 3 && abs(y1 - y2) < 3) {
75                             continue;
76                         }
77                         set<int> s1, r1, s2, r2;
78                         for (int i = 0; i < 3; i++) {
79                             for (int j = 0; j < 3; j++) {
80                                 s1.insert(a[x1+i][y1+j] / 13);
81                                 r1.insert(a[x1+i][y1+j] % 13);
82                                 s2.insert(a[x2+i][y2+j] / 13);
83                                 r2.insert(a[x2+i][y2+j] % 13);
84                             }
85                         }
86                         if ((s1.size() == 1 || r1.size() == 9) && (s2.size() == 1 || r2.size() == 9)) {
87                             cout << "Solution exists.\n";
88                             if (j1 == -1 && j2 == -1) {
89                                 cout << "There are no jokers.\n";
90                             } else if (j1 != -1 && j2 != -1) {
91                                 cout << "Replace J1 with " << idToCard(j1) << " and J2
92                                 with " << idToCard(j2) << ".\n";
93                             } else if (j1 != -1) {
94                                 cout << "Replace J1 with " << idToCard(j1) << ".\n";
95                             } else if (j2 != -1) {
96                                 cout << "Replace J2 with " << idToCard(j2) << ".\n";
97                             }
98                             cout << "Put the first square to (" << x1+1 << ", " << y1+1 << ").\n";
99                             cout << "Put the second square to (" << x2+1 << ", " << y2+1 << ").\n";
100                            exit(0);
101                         }
102                     }
103                 }
104             }
105         };
106         if (vis[52] == -1 && vis[53] == -1) {
107             check(-1, -1);
108         } else if (vis[52] != -1 && vis[53] != -1) {
109             for (int i = 0; i < 52; i++) {
110                 for (int j = 0; j < 52; j++) {

```

```

111         if (vis[i] == -1 && vis[j] == -1 && i != j) {
112             a[vis[52] / m][vis[52] % m] = i;
113             a[vis[53] / m][vis[53] % m] = j;
114             check(i, j);
115         }
116     }
117 }
118 } else if (vis[52] != -1) {
119     for (int i = 0; i < 52; i++) {
120         if (vis[i] == -1) {
121             a[vis[52] / m][vis[52] % m] = i;
122             check(i, -1);
123         }
124     }
125 } else {
126     for (int i = 0; i < 52; i++) {
127         if (vis[i] == -1) {
128             a[vis[53] / m][vis[53] % m] = i;
129             check(-1, i);
130         }
131     }
132 }
133 cout << "No solution.\n";
134 return 0;
135 }
```

## 556: Safe

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vasya tries to break in a safe. He knows that a code consists of n numbers, and every number is a 0 or a 1. Vasya has made m attempts to enter the code. After each attempt the system told him in how many position stand the right numbers. It is not said in which positions the wrong numbers stand. Vasya has been so unlucky that he hasn't entered the code where there would be more than 5 correct numbers. Now Vasya is completely bewildered: he thinks there's a mistake in the system and it is self-contradictory. Help Vasya - calculate how many possible code variants are left that do not contradict the previous system responses.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
```

```

7     int n, m;
8     cin >> n >> m;
9     i64 ans = 0;
10    vector<string> s(m);
11    vector<int> k(m);
12    for (int i = 0; i < m; i++) {
13        cin >> s[i] >> k[i];
14    }
15    function<void(int, vector<int>)> dfs = [&](int x, vector<int> a) {
16        if (x == n) {
17            ans += a == k;
18            return;
19        }
20        for (int i = 0; i < m; i++) {
21            if (a[i] > k[i]) {
22                return;
23            }
24        }
25        for (int c = 0; c < 2; c++) {
26            auto b = a;
27            for (int i = 0; i < m; i++) {
28                b[i] += s[i][x] == '0' + c;
29            }
30            dfs(x+1, b);
31        }
32    };
33    dfs(0, vector(m, 0));
34    cout << ans << "\n";
35    return 0;
36 }

```

## 557: Race

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Today's kilometer long auto race takes place in Berland. The track is represented by a straight line as long as  $s$  kilometers. There are  $n$  cars taking part in the race, all of them start simultaneously at the very beginning of the track. For every car is known its behavior - the system of segments on each of which the speed of the car is constant. The  $j$ -th segment of the  $i$ -th car is pair  $(v_i, j, t_i, j)$ , where  $v_i, j$  is the car's speed on the whole segment in kilometers per hour and  $t_i, j$  is for how many hours the car had been driving at that speed. The segments are given in the order in which they are "being driven on" by the cars.

Your task is to find out how many times during the race some car managed to have a lead over another car. A lead is considered a situation when one car appears in front of another car. It is known, that all the leads happen instantly, i. e. there are no such time segment of positive length, during which some two cars drive "together". At one moment of time on one and the same point several leads may appear.

In this case all of them should be taken individually. Meetings of cars at the start and finish are not considered to be counted as leads.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, s;
8     cin >> n >> s;
9     vector<vector<pair<int, int>>> a(n);
10    for (int i = 0; i < n; i++) {
11        int k;
12        cin >> k;
13        a[i].resize(k);
14        for (int j = 0; j < k; j++) {
15            int x, y;
16            cin >> x >> y;
17            a[i][j] = {x, y};
18        }
19    }
20    int ans = 0;
21    for (int x = 0; x < n; x++) {
22        for (int y = x+1; y < n; y++) {
23            auto A = a[x], B = a[y];
24            int i = 0, j = 0;
25            int lead = A[0] > B[0];
26            int d = 0;
27            while (i < A.size() && j < B.size()) {
28                int t = min(A[i].second, B[j].second);
29                d += (A[i].first - B[j].first) * t;
30                A[i].second -= t;
31                B[j].second -= t;
32                if (!A[i].second) {
33                    i++;
34                }
35                if (!B[j].second) {
36                    j++;
37                }
38                if (d && (d > 0) != lead) {
39                    ans += 1;
40                    lead ^= 1;
41                }
42            }
43        }
44    }
45    cout << ans << "\n";
46    return 0;
47 }
```

## 558: Safe cracking

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Right now you are to solve a very, very simple problem - to crack the safe. Four positive integers stand one by one on a circle protecting the safe. You know that to unlock this striking safe you have to make all four numbers equal to one. Operations are as follows: you may choose two adjacent numbers and increase both by one; you may choose two adjacent even numbers and divide both by two. Nothing else. Crack the safe!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     array<int, 4> a;
8     for (int i = 0; i < 4; i++) {
9         cin >> a[i];
10    }
11    vector<pair<char, int>> ans;
12    while (*max_element(a.begin(), a.end()) > 3) {
13        int x = max_element(a.begin(), a.end()) - a.begin();
14        if (a[x] % 2) {
15            ans.emplace_back('+', x+1);
16            a[x] += 1;
17            a[(x+1) % 4] += 1;
18        }
19        if (a[(x+1) % 4] % 2) {
20            ans.emplace_back('+', (x+1) % 4 + 1);
21            a[(x+1) % 4] += 1;
22            a[(x+2) % 4] += 1;
23        }
24        ans.emplace_back('/', x+1);
25        a[x] /= 2;
26        a[(x+1) % 4] /= 2;
27    }
28    map<array<int, 4>, pair<array<int, 4>, pair<char, int>>> S;
29    S[a] = {};
30    queue<array<int, 4>> q;
31    q.push(a);
32    while (!q.empty()) {
33        auto a = q.front();
34        q.pop();
35        if (a == array{1, 1, 1, 1}) {
36            break;
37        }
38        for (int i = 0; i < 4; i++) {

```

```

39         auto b = a;
40         b[i] += 1;
41         b[(i+1) % 4] += 1;
42         if (!S.count(b)) {
43             S[b] = {a, {'+', i+1}};
44             q.push(b);
45         }
46         b = a;
47         if (b[i] % 2 || b[(i+1) % 4] % 2) {
48             continue;
49         }
50         b[i] /= 2;
51         b[(i+1) % 4] /= 2;
52         if (!S.count(b)) {
53             S[b] = {a, {'/', i+1}};
54             q.push(b);
55         }
56     }
57 }
58 vector<pair<char, int>> res;
59 for (array<int, 4> t = {1, 1, 1, 1}; t != a; t = S[t].first) {
60     res.emplace_back(S[t].second);
61 }
62 ans.insert(ans.end(), res.rbegin(), res.rend());
63 for (auto [x, y] : ans) {
64     cout << x << y << "\n";
65 }
66 return 0;
67 }

```

## 559: Toys

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Little Masha loves arranging her toys into piles on the floor. And she also hates it when somebody touches her toys. One day Masha arranged all her  $n$  toys into several piles and then her elder brother Sasha came and gathered all the piles into one. Having seen it, Masha got very upset and started crying. Sasha still can't calm Masha down and mom is going to come home soon and punish Sasha for having made Masha crying. That's why he decides to restore the piles' arrangement. However, he doesn't remember at all the way the toys used to lie. Of course, Masha remembers it, but she can't talk yet and can only help Sasha by shouting happily when he arranges the toys in the way they used to lie. That means that Sasha will have to arrange the toys in every possible way until Masha recognizes the needed arrangement. The relative position of the piles and toys in every pile is irrelevant, that's why the two ways of arranging the toys are considered different if can be found two such toys that when arranged in the first way lie in one and the same pile and do not if arranged in the second way. Sasha is looking for the fastest way of trying all the ways because mom will come soon. With every action Sasha can take a toy from any pile and move it to any other pile (as a result a new pile may appear or

the old one may disappear). Sasha wants to find the sequence of actions as a result of which all the pile arrangement variants will be tried exactly one time each. Help Sasha. As we remember, initially all the toys are located in one pile.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<vector<vector<int>>> ans;
10    ans.push_back({}); 
11    for (int i = 1; i <= n; i++) {
12        vector<vector<vector<int>>> res;
13        int t = 1;
14        for (auto p : ans) {
15            vector<vector<vector<int>>> a;
16            p.push_back({i});
17            a.push_back(p);
18            p.pop_back();
19            for (int j = p.size()-1; j >= 0; j--) {
20                p[j].push_back(i);
21                a.push_back(p);
22                p[j].pop_back();
23            }
24            if (t) {
25                reverse(a.begin(), a.end());
26            }
27            res.insert(res.end(), a.begin(), a.end());
28            t ^= 1;
29        }
30        ans = res;
31    }
32    cout << ans.size() << "\n";
33    for (auto p : ans) {
34        for (auto s : p) {
35            cout << "{";
36            for (auto x : s) {
37                cout << x << ",}"[x == s.back()];
38            }
39            cout << ",\n"[s == p.back()];
40        }
41    }
42    return 0;
43 }
```

## 560: Multitest Generator

- Time limit: 2 seconds
- Memory limit: 256 megabytes

- Input file: standard input
- Output file: standard output

Let's call an array  $b_1, b_2, \dots, b_m$  a test if  $b_1 = m - 1$ .

Let's call an array  $b_1, b_2, \dots, b_m$  a multitest if the array  $b_2, b_3, \dots, b_m$  can be split into  $b_1$  non-empty subarrays so that each of these subarrays is a test. Note that each element of the array must be included in exactly one subarray, and the subarrays must consist of consecutive elements.

Let's define the function  $f$  from the array  $b_1, b_2, \dots, b_m$  as the minimum number of operations of the form "Replace any  $b_i$  with any non-negative integer  $x$ ", which needs to be done so that the array  $b_1, b_2, \dots, b_m$  becomes a multitest.

You are given an array of positive integers  $a_1, a_2, \dots, a_n$ . For each  $i$  from 1 to  $n - 1$ , find  $f([a_i, a_{i+1}, \dots, a_n])$ .

Below are some examples of tests and multitests.

Tests: [1, 5], [2, 2, 2], [3, 4, 1, 1], [5, 0, 0, 0, 0, 0], [7, 1, 2, 3, 4, 5, 6, 7], [0]. These arrays are tests since their first element (underlined) is equal to the length of the array minus one.

Multitests: [1, 1, 1], [2, 3, 0, 0, 1, 1, 12], [3, 2, 2, 7, 1, 1, 3, 4, 4, 4], [4, 0, 3, 1, 7, 9, 4, 2, 0, 0, 9, 1, 777]. Underlined are the subarrays after the split, and double underlined are the first elements of each subarray.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<int> step(n + 1), suf(n + 1), f(n + 1);
12    vector<int> ans(n - 1);
13    for (int i = n - 1; i >= 1; i--) {
14        step[i] = (i + a[i] + 1 > n || step[i + a[i] + 1] == -1) ? -1 : step[i + a[i] + 1]
15            + 1;
16        suf[i] = max(suf[i + 1], step[i + 1]);
17        f[i] = max(i + a[i] + 1 > n ? -1 : f[i + a[i] + 1] + 1, suf[i + 1]);
18        if (step[i] == a[i - 1]) {
19            ans[i - 1] = 0;
20        } else if (step[i] != -1 || f[i] >= a[i - 1]) {
21            ans[i - 1] = 1;
22        } else {
23            ans[i - 1] = 2;
24        }
25    }
26}
```

```

23         }
24     }
25     for (int i = 0; i < n - 1; i++) {
26         cout << ans[i] << " \n"[i == n - 2];
27     }
28 }
29 int main() {
30     ios::sync_with_stdio(false);
31     cin.tie(nullptr);
32     int t;
33     cin >> t;
34     while (t--) {
35         solve();
36     }
37     return 0;
38 }
```

**561: Tree Master**

- Time limit: 3 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

You are given a tree with  $n$  weighted vertices labeled from 1 to  $n$  rooted at vertex 1. The parent of vertex  $i$  is  $p_i$  and the weight of vertex  $i$  is  $a_i$ . For convenience, define  $p_1 = 0$ .

For two vertices  $x$  and  $y$  of the same depth<sup>†</sup>, define  $f(x, y)$  as follows:

Initialize  $\text{ans} = 0$ .

While both  $x$  and  $y$  are not 0:  $\text{ans} \leftarrow \text{ans} + a_x \cdot a_y; x \leftarrow p_x; y \leftarrow p_y$ .

$\text{ans} \leftarrow \text{ans} + a_x \cdot a_y;$

$x \leftarrow p_x;$

$y \leftarrow p_y$ .

$f(x, y)$  is the value of  $\text{ans}$ .

You will process  $q$  queries. In the  $i$ -th query, you are given two integers  $x_i$  and  $y_i$  and you need to calculate  $f(x_i, y_i)$ .

<sup>†</sup> The depth of vertex  $v$  is the number of edges on the unique simple path from the root of the tree to vertex  $v$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, q;
8     cin >> n >> q;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    vector<int> p(n, -1), dep(n);
14    for (int i = 1; i < n; i++) {
15        cin >> p[i];
16        p[i]--;
17        dep[i] = dep[p[i]] + 1;
18    }
19    vector<vector<int>> v(n);
20    vector<int> id(n);
21    for (int i = 0; i < n; i++) {
22        id[i] = v[dep[i]].size();
23        v[dep[i]].push_back(i);
24    }
25    vector<vector<i64>> g(n);
26    for (int i = 0; i < n; i++) {
27        if (v[dep[i]].size() <= 300) {
28            g[i].resize(v[dep[i]].size(), -1);
29        }
30    }
31    function<i64(int, int)> f = [&](int x, int y) {
32        if (x == -1) {
33            return 0LL;
34        }
35        if (v[dep[x]].size() <= 300 && g[x][id[y]] != -1) {
36            return g[x][id[y]];
37        }
38        i64 res = f(p[x], p[y]) + 1LL * a[x] * a[y];
39        if (v[dep[x]].size() <= 300) {
40            g[x][id[y]] = res;
41        }
42        return res;
43    };
44    while (q--) {
45        int x, y;
46        cin >> x >> y;
47        x--, y--;
48        cout << f(x, y) << "\n";
49    }
50    return 0;
51 }

```

## 562: Serval and Shift-Shift-Shift

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Serval has two  $n$ -bit binary integer numbers  $a$  and  $b$ . He wants to share those numbers with Toxel.

Since Toxel likes the number  $b$  more, Serval decides to change  $a$  into  $b$  by some (possibly zero) operations. In an operation, Serval can choose any positive integer  $k$  between 1 and  $n$ , and change  $a$  into one of the following number:

$$a \oplus (a \ll k)$$

$$a \oplus (a \gg k)$$

In other words, the operation moves every bit of  $a$  left or right by  $k$  positions, where the overflowed bits are removed, and the missing bits are padded with 0. The bitwise XOR of the shift result and the original  $a$  is assigned back to  $a$ .

Serval does not have much time. He wants to perform no more than  $n$  operations to change  $a$  into  $b$ . Please help him to find out an operation sequence, or determine that it is impossible to change  $a$  into  $b$  in at most  $n$  operations. You do not need to minimize the number of operations.

In this problem,  $x \oplus y$  denotes the bitwise XOR operation of  $x$  and  $y$ .  $a \ll k$  and  $a \gg k$  denote the logical left shift and logical right shift.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     string a, b;
8     cin >> a >> b;
9     vector<int> ans;
10    if (a == b) {
11        cout << 0 << "\n";
12        return;
13    }
14    vector<int> A(n), B(n);
15    for (int i = 0; i < n; i++) {
16        A[i] = a[i] - '0';
17        B[i] = b[i] - '0';
18    }
19    if (count(A.begin(), A.end(), 1) == 0) {
20        cout << -1 << "\n";
21        return;
22    }
23    if (count(B.begin(), B.end(), 1) == 0) {
24        cout << -1 << "\n";
25        return;
26    }
27    int x = find(A.begin(), A.end(), 1) - A.begin();
28    int y = find(B.begin(), B.end(), 1) - B.begin();
29    if (x > y) {
30        int d = x - y;
31        for (int i = 0; i + d < n; i++) {
32            A[i] ^= A[i + d];

```

```

33         }
34         ans.push_back(d);
35         x = y;
36     }
37     for (int i = x + 1; i < n; i++) {
38         if (A[i] != B[i]) {
39             int d = i - x;
40             for (int j = n - 1; j >= d; j--) {
41                 A[j] ^= A[j - d];
42             }
43             ans.push_back(-d);
44         }
45     }
46     int z = n - 1;
47     while (A[z] == 0) {
48         z--;
49     }
50     for (int i = x; i >= 0; i--) {
51         if (A[i] != B[i]) {
52             int d = z - i;
53             for (int j = d; j < n; j++) {
54                 A[j - d] ^= A[j];
55             }
56             ans.push_back(d);
57         }
58     }
59     cout << ans.size() << "\n";
60     for (int i = 0; i < ans.size(); i++) {
61         cout << ans[i] << " \n"[i == ans.size() - 1];
62     }
63 }
64 int main() {
65     ios::sync_with_stdio(false);
66     cin.tie(nullptr);
67     int t;
68     cin >> t;
69     while (t--) {
70         solve();
71     }
72     return 0;
73 }
```

### 563: Three Chairs

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

One day Kira found  $n$  friends from Morioh and decided to gather them around a table to have a peaceful conversation. The height of friend  $i$  is equal to  $a_i$ . It so happened that the height of each of the friends is unique.

Unfortunately, there were only 3 chairs in Kira's house, and obviously, it will not be possible to seat all friends! So, Kira has to invite only 3 of his friends.

But everything is not so simple! If the heights of the lowest and the tallest of the invited friends are not coprime, then the friends will play tricks on each other, which will greatly anger Kira.

Kira became interested, how many ways are there to choose 3 of his friends so that they don't play tricks? Two ways are considered different if there is a friend invited in one way, but not in the other.

Formally, if Kira invites friends  $i, j$ , and  $k$ , then the following should be true:  $\gcd(\min(a_i, a_j, a_k), \max(a_i, a_j, a_k)) = 1$ , where  $\gcd(x, y)$  denotes the greatest common divisor (GCD) of the numbers  $x$  and  $y$ .

Kira is not very strong in computer science, so he asks you to count the number of ways to invite friends.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    sort(a.begin(), a.end());
14    const int m = a[n - 1];
15    vector<int> mu(m + 1);
16    mu[1] = 1;
17    for (int i = 1; i <= m; i++) {
18        for (int j = 2 * i; j <= m; j += i) {
19            mu[j] -= mu[i];
20        }
21    }
22    vector<vector<int>> divs(m + 1);
23    for (int i = 1; i <= m; i++) {
24        for (int j = i; j <= m; j += i) {
25            divs[j].push_back(i);
26        }
27    }
28    i64 ans = 0, cur = 0;
29    vector<int> cl(m + 1), cr(m + 1);
30    for (int i = 0; i < n; i++) {
31        for (auto d : divs[a[i]]) {
32            cr[d]++;
33        }
34    }
35    for (int i = 0; i < n; i++) {
36        for (auto d : divs[a[i]]) {
37            cur -= mu[d] * cl[d];
38            cr[d]--;
39        }
40        ans += cur;
41        for (auto d : divs[a[i]]) {
42            cur += mu[d] * cr[d];
43            cl[d]++;
44        }
}
```

```

45     }
46     cout << ans << "\n";
47     return 0;
48 }
```

## 564: Rectangle Shrinking

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You have a rectangular grid of height 2 and width  $10^9$  consisting of unit cells. There are  $n$  rectangles placed on this grid, and the borders of these rectangles pass along cell borders. The  $i$ -th rectangle covers all cells in rows from  $u_i$  to  $d_i$  inclusive and columns from  $l_i$  to  $r_i$  inclusive ( $1 \leq u_i \leq d_i \leq 2$ ;  $1 \leq l_i \leq r_i \leq 10^9$ ). The initial rectangles can intersect, be nested, and coincide arbitrarily.

You should either remove each rectangle, or replace it with any of its non-empty subrectangles. In the latter case, the new subrectangle must lie inside the initial rectangle, and its borders must still pass along cell borders. In particular, it is allowed for the subrectangle to be equal to the initial rectangle.

After that replacement, no two (non-removed) rectangles are allowed to have common cells, and the total area covered with the new rectangles must be as large as possible.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> u(n), d(n), l(n), r(n);
8     for (int i = 0; i < n; i++) {
9         cin >> u[i] >> l[i] >> d[i] >> r[i];
10    }
11    map<int, int> f[2], g[2];
12    vector<int> p(n);
13    iota(p.begin(), p.end(), 0);
14    sort(p.begin(), p.end(), [&](int i, int j) {
15        return l[i] < l[j];
16    });
17    for (auto i : p) {
18        if (d[i] - u[i] == 1) continue;
19        for (int x = u[i] - 1; x < d[i]; x++) {
20            if (!f[x].empty() && f[x].rbegin()->second >= l[i] - 1) {
21                f[x].rbegin()->second = max(f[x].rbegin()->second, r[i]);
22            } else {
```

```

23             f[x][l[i]] = r[i];
24         }
25     }
26 }
27 vector<int> q;
28 for (auto i : p) {
29     if (d[i] - u[i] != 1) continue;
30     int cv[2] {};
31     for (int x = 0; x < 2; x++) {
32         auto it = f[x].upper_bound(l[i]);
33         if (it != f[x].begin() && prev(it)->second >= r[i]) {
34             cv[x] = 1;
35         }
36     }
37     if (cv[0] && cv[1]) {
38         u[i] = d[i] = l[i] = r[i] = 0;
39         continue;
40     }
41     if (!cv[0] && !cv[1]) {
42         if (!g[0].empty() && g[0].rbegin()->second >= r[i]) {
43             u[i] = d[i] = l[i] = r[i] = 0;
44         } else if (!g[0].empty() && l[i] <= g[0].rbegin()->second + 1) {
45             l[i] = g[0].rbegin()->second + 1;
46             g[0].rbegin()->second = r[i];
47         } else {
48             g[0][l[i]] = r[i];
49         }
50         continue;
51     }
52     if (cv[0]) u[i] = 2;
53     else d[i] = 1;
54 }
55 g[1] = g[0];
56 for (auto i : p) {
57     if (!u[i] || d[i] - u[i] == 1) continue;
58     int x = u[i] - 1;
59     auto it = g[x].upper_bound(l[i]);
60     if (it != g[x].end()) {
61         r[i] = min(r[i], it->first - 1);
62     }
63     if (it != g[x].begin() && prev(it)->second >= r[i]) {
64         u[i] = d[i] = l[i] = r[i] = 0;
65         continue;
66     }
67     if (it != g[x].begin() && prev(it)->second >= l[i]) {
68         l[i] = prev(it)->second + 1;
69         prev(it)->second = r[i];
70     } else {
71         g[x][l[i]] = r[i];
72     }
73 }
74 int ans = 0;
75 for (int x = 0; x < 2; x++) {
76     for (auto [l, r] : g[x]) {
77         ans += r - l + 1;
78     }
79 }
80 cout << ans << "\n";
81 for (int i = 0; i < n; i++) {
82     cout << u[i] << " " << l[i] << " " << d[i] << " " << r[i] << "\n";
83 }
84 }
85 int main() {
86     ios::sync_with_stdio(false);

```

```

87     cin.tie(nullptr);
88     int t;
89     cin >> t;
90     while (t--) {
91         solve();
92     }
93     return 0;
94 }
```

## 565: Decomposition

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

For a sequence of integers  $[x_1, x_2, \dots, x_k]$ , let's define its decomposition as follows:

Process the sequence from the first element to the last one, maintaining the list of its subsequences. When you process the element  $x_i$ , append it to the end of the first subsequence in the list such that the bitwise AND of its last element and  $x_i$  is greater than 0. If there is no such subsequence in the list, create a new subsequence with only one element  $x_i$  and append it to the end of the list of subsequences.

For example, let's analyze the decomposition of the sequence  $[1, 3, 2, 0, 1, 3, 2, 1]$ :

processing element 1, the list of subsequences is empty. There is no subsequence to append 1 to, so we create a new subsequence  $[1]$ ;

processing element 3, the list of subsequences is  $[[1]]$ . Since the bitwise AND of 3 and 1 is 1, the element is appended to the first subsequence;

processing element 2, the list of subsequences is  $[[1, 3]]$ . Since the bitwise AND of 2 and 3 is 2, the element is appended to the first subsequence;

processing element 0, the list of subsequences is  $[[1, 3, 2]]$ . There is no subsequence to append 0 to, so we create a new subsequence  $[0]$ ;

processing element 1, the list of subsequences is  $[[1, 3, 2], [0]]$ . There is no subsequence to append 1 to, so we create a new subsequence  $[1]$ ;

processing element 3, the list of subsequences is  $[[1, 3, 2], [0], [1]]$ . Since the bitwise AND of 3 and 2 is 2, the element is appended to the first subsequence;

processing element 2, the list of subsequences is  $[[1, 3, 2, 3], [0], [1]]$ . Since the bitwise AND of 2 and 3 is 2, the element is appended to the first subsequence;

processing element 1, the list of subsequences is  $[[1, 3, 2, 3, 2], [0], [1]]$ . The element 1 cannot be appended to any of the first two subsequences, but can be appended to the third one.

The resulting list of subsequences is  $[[1, 3, 2, 3, 2], [0], [1, 1]]$ .

Let  $f([x_1, x_2, \dots, x_k])$  be the number of subsequences the sequence  $[x_1, x_2, \dots, x_k]$  is decomposed into.

Now, for the problem itself.

You are given a sequence  $[a_1, a_2, \dots, a_n]$ , where each element is an integer from 0 to 3. Let  $a[i..j]$  be the sequence  $[a_i, a_{i+1}, \dots, a_j]$ . You have to calculate  $\sum_{i=1}^n \sum_{j=i}^n f(a[i..j])$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     i64 ans = 0;
10    vector<int> a(n), prv(n, -1);
11    vector<array<int, 4>> p(n);
12    int lst = -1;
13    array<int, 4> l{-1, -1, -1, -1};
14    for (int i = 0; i < n; i++) {
15        cin >> a[i];
16        if (a[i] == 0) ans += 1LL * (i + 1) * (n - i);
17        else {
18            prv[i] = lst;
19            lst = i;
20        }
21        p[i] = l;
22        l[a[i]] = i;
23    }
24    for (int i = 0, j = 0, k = 0, l[2] {}; i < n; i++) {
25        if (j < i) j = i;
26        while (j < n && a[j] == 0) j++;
27        ans += n - j;
28        if (k < i) k = i;
29        while (k < n && (prv[k] < i || (a[k] & a[prv[k]]))) k++;
30        ans += n - k;
31        for (int t = 0; t < 2; t++) {
32            if (l[t] < k) l[t] = k;
33            while (l[t] < n && (a[l[t]] != ((t + 1) ^ 3) || a[prv[l[t]]] != (t + 1) || p[prv[l[t]]][3] < max(k, p[prv[l[t]]][(t + 1) ^ 3]))) l[t]++;
34        }
35        if (k < n) ans += n - l[a[k] - 1];
36    }
37    cout << ans << "\n";
38    return 0;
39 }
```

## 566: Kirill and Company

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Kirill lives on a connected undirected graph of  $n$  vertices and  $m$  edges at vertex 1. One fine evening he gathered  $f$  friends, the  $i$ -th friend lives at the vertex  $h_i$ . So all friends are now in the vertex 1, the  $i$ -th friend must get to his home to the vertex  $h_i$ .

The evening is about to end and it is time to leave. It turned out that  $k$  ( $k \leq 6$ ) of his friends have no cars, and they would have to walk if no one gives them a ride. One friend with a car can give a ride to any number of friends without cars, but only if he can give them a ride by driving along one of the shortest paths to his house.

For example, in the graph below, a friend from vertex  $h_i = 5$  can give a ride to friends from the following sets of vertices: [2, 3], [2, 4], [2], [3], [4], but can't give a ride to friend from vertex 6 or a set [3, 4].

Kirill wants as few friends as possible to have to walk. Help him find the minimum possible number.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int inf = 1E9;
5 void solve() {
6     int n, m;
7     cin >> n >> m;
8     vector<vector<int>> adj(n);
9     for (int i = 0; i < m; i++) {
10         int u, v;
11         cin >> u >> v;
12         u--, v--;
13         adj[u].push_back(v);
14         adj[v].push_back(u);
15     }
16     int f;
17     cin >> f;
18     vector<int> h(f);
19     for (int i = 0; i < f; i++) {
20         cin >> h[i];
21         h[i]--;
22     }
23     int k;
24     cin >> k;
25     for (int i = 0; i < k; i++) {
26         int p;
27         cin >> p;

```

```

28         p--;
29         swap(h[i], h[p]);
30     }
31     auto bfs = [&](int s) {
32         vector<int> d(n, -1);
33         queue<array<int, 2>> q;
34         q.push({s, 0});
35         while (!q.empty()) {
36             auto [x, v] = q.front();
37             q.pop();
38             if (d[x] != -1) {
39                 continue;
40             }
41             d[x] = v;
42             for (auto y : adj[x]) {
43                 q.push({y, v + 1});
44             }
45         }
46         return d;
47     };
48     auto d0 = bfs(0);
49     sort(h.begin(), h.begin() + k, [&](int i, int j) {
50         return d0[i] < d0[j];
51     });
52     vector<vector<int>> d(k);
53     for (int i = 0; i < k; i++) {
54         d[i] = bfs(h[i]);
55     }
56     vector<int> dp(1 << k);
57     dp[0] = 1;
58     for (int i = k; i < f; i++) {
59         vector<int> good(1 << k);
60         for (int s = 1; s < (1 << k); s++) {
61             int lst = -1;
62             int dist = 0;
63             for (int i = 0; i < k; i++) {
64                 if (s >> i & 1) {
65                     if (lst == -1) {
66                         dist += d0[h[i]];
67                     } else {
68                         dist += d[lst][h[i]];
69                     }
70                     lst = i;
71                 }
72             }
73             dist += d[lst][h[i]];
74             if (dist == d0[h[i]]) {
75                 good[s] = 1;
76             }
77         }
78         for (int s = (1 << k) - 1; s; s--) {
79             for (int t = s; t; t = (t - 1) & s) {
80                 dp[s] |= dp[s ^ t] & good[t];
81             }
82         }
83     }
84     int ans = k;
85     for (int i = 0; i < (1 << k); i++) {
86         if (dp[i]) {
87             ans = min(ans, k - __builtin_popcount(i));
88         }
89     }
90     cout << ans << "\n";
91 }

```

```

92 int main() {
93     ios::sync_with_stdio(false);
94     cin.tie(nullptr);
95     int t;
96     cin >> t;
97     while (t--) {
98         solve();
99     }
100    return 0;
101 }
```

## data structures

### 567: Happy Life in University

- Time limit: 1 second
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Egor and his friend Arseniy are finishing school this year and will soon enter university. And since they are very responsible guys, they have started preparing for admission already.

First of all, they decided to take care of where they will live for the long four years of study, and after visiting the university's website, they found out that the university dormitory can be represented as a root tree with  $n$  vertices with the root at vertex 1. In the tree, each vertex represents a recreation with some type of activity  $a_i$ . The friends need to choose 2 recreations (not necessarily different) in which they will settle. The guys are convinced that the more the value of the following function  $f(u, v) = \text{diff}(u, \text{lca}(u, v)) \cdot \text{diff}(v, \text{lca}(u, v))$ , the more fun their life will be. Help Egor and Arseniy and find the maximum value of  $f(u, v)$  among all pairs of recreations!

$\dagger \text{diff}(u, v)$  - the number of different activities listed on the simple path from vertex  $u$  to vertex  $v$ .

$\dagger \text{lca}(u, v)$  - a vertex  $p$  such that it is at the maximum distance from the root and is a parent of both vertex  $u$  and vertex  $v$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct HLD;
5 template<class Info, class Tag>
6 # struct LazySegmentTree;
7 # struct Tag;
```

```

8  # struct Info;
9  Info operator+(const Info &a, const Info &b) {
10    return {max(a.max, b.max)};
11 }
12 void solve() {
13   int n;
14   cin >> n;
15   vector<int> p(n);
16   HLD t(n);
17   for (int i = 1; i < n; i++) {
18     cin >> p[i];
19     p[i]--;
20     t.addEdge(i, p[i]);
21   }
22   t.work();
23   vector<int> a(n);
24   for (int i = 0; i < n; i++) {
25     cin >> a[i];
26     a[i]--;
27   }
28   vector<set<int>> s(n);
29   LazySegmentTree<Info, Tag> seg(n);
30   i64 ans = 1;
31   auto dfs = [&](auto self, int x) -> void {
32     for (auto y : t.adj[x]) {
33       self(self, y);
34     }
35     for (auto it = s[a[x]].lower_bound(t.in[x]); it != s[a[x]].end(); ) {
36       if (*it >= t.out[x]) {
37         continue;
38       }
39       int y = t.seq[*it];
40       it = s[a[x]].erase(it);
41       seg.rangeApply(t.in[y], t.out[y], {-1});
42     }
43     seg.rangeApply(t.in[x], t.out[x], {1});
44     s[a[x]].insert(t.in[x]);
45     int res = 1;
46     for (auto y : t.adj[x]) {
47       int v = seg.rangeQuery(t.in[y], t.out[y]).max;
48       ans = max(ans, 1LL * res * v);
49       res = max(res, v);
50     }
51   };
52   dfs(dfs, 0);
53   cout << ans << "\n";
54 }
55 int main() {
56   ios::sync_with_stdio(false);
57   cin.tie(nullptr);
58   int t;
59   cin >> t;
60   while (t--) {
61     solve();
62   }
63   return 0;
64 }

```

**568: Yet Another Inversions Problem**

- Time limit: 2 seconds

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a permutation  $p_0, p_1, \dots, p_{n-1}$  of odd integers from 1 to  $2n - 1$  and a permutation  $q_0, q_1, \dots, q_{k-1}$  of integers from 0 to  $k - 1$ .

An array  $a_0, a_1, \dots, a_{nk-1}$  of length  $nk$  is defined as follows:

For example, if  $p = [3, 5, 1]$  and  $q = [0, 1]$ , then  $a = [3, 6, 5, 10, 1, 2]$ .

Note that all arrays in the statement are zero-indexed. Note that each element of the array  $a$  is uniquely determined.

Find the number of inversions in the array  $a$ . Since this number can be very large, you should find only its remainder modulo 998 244 353.

An inversion in array  $a$  is a pair  $(i, j)$  ( $0 \leq i < j < nk$ ) such that  $a_i > a_j$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 template <typename T>
35 # struct Fenwick;
```

```

36 void solve() {
37     int n, k;
38     cin >> n >> k;
39     vector<int> p(n), q(k);
40     for (int i = 0; i < n; i++) {
41         cin >> p[i];
42     }
43     for (int i = 0; i < k; i++) {
44         cin >> q[i];
45     }
46     Fenwick<int> fen(k);
47     Z ans = 0;
48     for (int i = k - 1; i >= 0; i--) {
49         ans += fen.sum(q[i]);
50         fen.add(q[i], 1);
51     }
52     ans *= n;
53     fen.init(2 * n);
54     Z all = 1LL * k * (k + 1) / 2;
55     for (int d = 0; d < k && d <= 18; d++) {
56         all -= k - d;
57     }
58     for (int i = n - 1; i >= 0; i--) {
59         for (int d = -18; d <= 18; d++) {
60             if (abs(d) >= k) {
61                 continue;
62             }
63             i64 v = d > 0 ? i64(p[i]) << d : (p[i] >> -d) + 1;
64             if (v > 2 * n) {
65                 v = 2 * n;
66             }
67             ans += Z(fen.sum(v)) * (k - abs(d));
68         }
69         ans += Z(fen.sum(2 * n)) * all;
70         fen.add(p[i], 1);
71     }
72     cout << ans << "\n";
73 }
74 int main() {
75     ios::sync_with_stdio(false);
76     cin.tie(nullptr);
77     int t;
78     cin >> t;
79     while (t--) {
80         solve();
81     }
82     return 0;
83 }

```

### 569: Light Bulbs (Hard Version)

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

The easy and hard versions of this problem differ only in the constraints on  $n$ . In the hard version, the sum of values of  $n$  over all test cases does not exceed  $2 \cdot 10^5$ . Furthermore, there are no additional

constraints on the value of  $n$  in a single test case.

There are  $2n$  light bulbs arranged in a row. Each light bulb has a color from 1 to  $n$  (exactly two light bulbs for each color).

Initially, all light bulbs are turned off. You choose a set of light bulbs  $S$  that you initially turn on. After that, you can perform the following operations in any order any number of times:

choose two light bulbs  $i$  and  $j$  of the same color, exactly one of which is on, and turn on the second one;

choose three light bulbs  $i, j, k$ , such that both light bulbs  $i$  and  $k$  are on and have the same color, and the light bulb  $j$  is between them ( $i < j < k$ ), and turn on the light bulb  $j$ .

You want to choose a set of light bulbs  $S$  that you initially turn on in such a way that by performing the described operations, you can ensure that all light bulbs are turned on.

Calculate two numbers:

the minimum size of the set  $S$  that you initially turn on;

the number of sets  $S$  of minimum size (taken modulo 998244353).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>

```

```

29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
35 void solve() {
36     int n;
37     cin >> n;
38     vector<i64> w(n);
39     vector<int> c(2 * n);
40     for (int i = 0; i < n; i++) {
41         w[i] = rng();
42     }
43     vector<i64> f(2 * n + 1);
44     for (int i = 0; i < 2 * n; i++) {
45         cin >> c[i];
46         c[i]--;
47         f[i + 1] = f[i] ^ w[c[i]];
48     }
49     int ans = count(f.begin(), f.end(), 0) - 1;
50     Z ways = 1;
51     map<i64, int> last;
52     for (int i = 0; i <= 2 * n; i++) {
53         last[f[i]] = i;
54     }
55     for (int i = 0; i < 2 * n; i++) {
56         if (f[i] == 0) {
57             int cnt = 1;
58             int j = i + 1;
59             while (f[j] != 0) {
60                 cnt += 1;
61                 j = last[f[j]];
62                 j += 1;
63             }
64             ways *= cnt;
65         }
66     }
67     cout << ans << " " << ways << "\n";
68 }
69 int main() {
70     ios::sync_with_stdio(false);
71     cin.tie(nullptr);
72     int t;
73     cin >> t;
74     while (t--) {
75         solve();
76     }
77     return 0;
78 }
```

## 570: Sofia and Strings

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Sofia has a string  $s$  of length  $n$ , consisting only of lowercase English letters. She can perform operations of the following types with this string.

Select an index  $1 \leq i \leq |s|$  and remove the character  $s_i$  from the string.

Select a pair of indices  $(l, r)$  ( $1 \leq l \leq r \leq |s|$ ) and sort the substring  $s_l s_{l+1} \dots s_r$  in alphabetical order.

Sofia wants to obtain the string  $t$  of length  $m$  after performing zero or more operations on string  $s$  as described above. Please determine whether it is possible or not.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 1E18;
5 void solve() {
6     int n, m;
7     cin >> n >> m;
8     string s, t;
9     cin >> s >> t;
10    array<vector<int>, 26> vec;
11    for (int i = n - 1; i >= 0; i--) {
12        vec[s[i] - 'a'].push_back(i);
13    }
14    for (int i = 0; i < m; i++) {
15        int x = t[i] - 'a';
16        if (vec[x].empty()) {
17            cout << "NO\n";
18            return;
19        }
20        int p = vec[x].back();
21        vec[x].pop_back();
22        for (int j = 0; j < x; j++) {
23            while (!vec[j].empty() && vec[j].back() < p) {
24                vec[j].pop_back();
25            }
26        }
27    }
28    cout << "YES\n";
29 }
30 int main() {
31     ios::sync_with_stdio(false);
32     cin.tie(nullptr);
33     int t;
34     cin >> t;
35     while (t--) {
36         solve();
37     }
38     return 0;
39 }
```

## 571: Infinite Card Game

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Monocarp and Bicarp are playing a card game. Each card has two parameters: an attack value and a defence value. A card  $s$  beats another card  $t$  if the attack of  $s$  is strictly greater than the defence of  $t$ .

Monocarp has  $n$  cards, the  $i$ -th of them has an attack value of  $ax_i$  and a defence value of  $ay_i$ . Bicarp has  $m$  cards, the  $j$ -th of them has an attack value of  $bx_j$  and a defence value of  $by_j$ .

On the first move, Monocarp chooses one of his cards and plays it. Bicarp has to respond with his own card that beats that card. After that, Monocarp has to respond with a card that beats Bicarp's card. After that, it's Bicarp's turn, and so forth.

After a card is beaten, it returns to the hand of the player who played it. It implies that each player always has the same set of cards to play as at the start of the game. The game ends when the current player has no cards that beat the card which their opponent just played, and the current player loses.

If the game lasts for  $100^{500}$  moves, it's declared a draw.

Both Monocarp and Bicarp play optimally. That is, if a player has a winning strategy regardless of his opponent's moves, he plays for a win. Otherwise, if he has a drawing strategy, he plays for a draw.

You are asked to calculate three values:

the number of Monocarp's starting moves that result in a win for Monocarp;

the number of Monocarp's starting moves that result in a draw;

the number of Monocarp's starting moves that result in a win for Bicarp.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> ax(n), ay(n);
8     for (int i = 0; i < n; i++) {
9         cin >> ax[i];
10    }
11    for (int i = 0; i < n; i++) {

```

```

12         cin >> ay[i];
13     }
14     int m;
15     cin >> m;
16     vector<int> bx(m), by(m);
17     for (int i = 0; i < m; i++) {
18         cin >> bx[i];
19     }
20     for (int i = 0; i < m; i++) {
21         cin >> by[i];
22     }
23     vector<int> pa(n), pb(m);
24     iota(pa.begin(), pa.end(), 0);
25     iota(pb.begin(), pb.end(), 0);
26     sort(pa.begin(), pa.end(),
27           [&](int i, int j) {
28               return ax[i] < ax[j];
29           });
30     sort(pb.begin(), pb.end(),
31           [&](int i, int j) {
32               return bx[i] < bx[j];
33           });
34     auto fa = pa, fb = pb;
35     for (int i = n - 2; i >= 0; i--) {
36         fa[i] = ay[pa[i]] > ay[fa[i + 1]] ? pa[i] : fa[i + 1];
37     }
38     for (int i = m - 2; i >= 0; i--) {
39         fb[i] = by[pb[i]] > by[fb[i + 1]] ? pb[i] : fb[i + 1];
40     }
41     vector<vector<int>> adj(n + m);
42     vector<int> deg(n + m);
43     vector<int> dp(n + m, -1);
44     auto addEdge = [&](int u, int v) {
45         deg[u] += 1;
46         adj[v].push_back(u);
47     };
48     for (int i = 0; i < n; i++) {
49         auto it = partition_point(pb.begin(), pb.end(),
50             [&](int j) {
51                 return bx[j] <= ay[i];
52             });
53         if (it != pb.end()) {
54             addEdge(i, n + fb[it - pb.begin()]);
55         }
56     }
57     for (int i = 0; i < m; i++) {
58         auto it = partition_point(pa.begin(), pa.end(),
59             [&](int j) {
60                 return ax[j] <= by[i];
61             });
62         if (it != pa.end()) {
63             addEdge(n + i, fa[it - pa.begin()]);
64         }
65     }
66     queue<int> q;
67     for (int i = 0; i < n + m; i++) {
68         if (deg[i] == 0) {
69             dp[i] = 1;
70             q.push(i);
71         }
72     }
73     while (!q.empty()) {
74         int x = q.front();
75         q.pop();

```

```

76         for (auto y : adj[x]) {
77             dp[y] = !dp[x];
78             q.push(y);
79         }
80     }
81     int ans1 = count(dp.begin(), dp.begin() + n, 1);
82     int ans2 = count(dp.begin(), dp.begin() + n, -1);
83     int ans3 = count(dp.begin(), dp.begin() + n, 0);
84     cout << ans1 << " " << ans2 << " " << ans3 << "\n";
85 }
86 int main() {
87     ios::sync_with_stdio(false);
88     cin.tie(nullptr);
89     int t;
90     cin >> t;
91     while (t--) {
92         solve();
93     }
94     return 0;
95 }
```

## 572: wxhtzdy ORO Tree

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

After (finally) qualifying for the IOI 2023, wxhtzdy was very happy, so he decided to do what most competitive programmers do: trying to guess the problems that will be on IOI. During this process, he accidentally made a problem, which he thought was really cool.

You are given a tree (a connected acyclic graph) with  $n$  vertices and  $n - 1$  edges. Vertex  $i$  ( $1 \leq i \leq n$ ) has a value  $a_i$ .

Lets' define  $g(u, v)$  as the bitwise or of the values of all vertices on the shortest path from  $u$  to  $v$ . For example, let's say that we want to calculate  $g(3, 4)$ , on the tree from the first test case in the example. On the path from 3 to 4 are vertices 3, 1, 4. Then,  $g(3, 4) = a_3 | a_1 | a_4$  (here,  $|$  represents the bitwise OR operation).

Also, you are given  $q$  queries, and each query looks like this:

You are given  $x$  and  $y$ . Let's consider all vertices  $z$  such that  $z$  is on the shortest path from  $x$  to  $y$  (inclusive).

Lets define the niceness of a vertex  $z$  as the sum of the number of non-zero bits in  $g(x, z)$  and the number of non-zero bits in  $g(y, z)$ . You need to find the maximum niceness among all vertices  $z$  on the shortest path from  $x$  to  $y$ .

Since his brain is really tired after solving an output only problem on SIO (he had to do it to qualify for the IOI), he wants your help with this problem.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<vector<int>> adj(n);
12    for (int i = 1; i < n; i++) {
13        int u, v;
14        cin >> u >> v;
15        u--, v--;
16        adj[u].push_back(v);
17        adj[v].push_back(u);
18    }
19    vector<int> dep(n), siz(n), in(n), s(n), p(n), top(n), pre(n);
20    vector nxt(n + 1, vector<int>(30, n));
21    vector lst(n + 1, vector<int>(30, 0));
22    int cur = 0;
23    auto dfs1 = [&](auto self, int x) -> void {
24        if (x) {
25            adj[x].erase(find(adj[x].begin(), adj[x].end(), p[x]));
26        }
27        siz[x] = 1;
28        for (auto &y : adj[x]) {
29            p[y] = x;
30            dep[y] = dep[x] + 1;
31            self(self, y);
32            siz[x] += siz[y];
33            if (siz[y] > siz[adj[x][0]]) {
34                swap(y, adj[x][0]);
35            }
36        }
37    };
38    dfs1(dfs1, 0);
39    auto dfs2 = [&](auto self, int x) -> void {
40        in[x] = cur++;
41        s[in[x]] = x;
42        pre[x] = a[x];
43        if (top[x] != x) {
44            pre[x] |= pre[p[x]];
45        }
46        for (auto y : adj[x]) {
47            top[y] = y == adj[x][0] ? top[x] : y;
48            self(self, y);
49        }
50    };
51    dfs2(dfs2, 0);
52    for (int i = n - 1; i >= 0; i--) {
53        nxt[i] = nxt[i + 1];
54        for (int j = 0; j < 30; j++) {
55            if (a[s[i]] >> j & 1) {
56                nxt[i][j] = i;
57            }
58        }
59    }
60}
```

```

57         }
58     }
59 }
60 for (int i = 1; i <= n; i++) {
61     lst[i] = lst[i - 1];
62     for (int j = 0; j < 30; j++) {
63         if (a[s[i - 1]] >> j & 1) {
64             lst[i][j] = i;
65         }
66     }
67 }
68 int q;
69 cin >> q;
70 while (q--) {
71     int x, y;
72     cin >> x >> y;
73     x--, y--;
74     vector<array<int, 3>> seg, segr;
75     while (top[x] != top[y]) {
76         if (dep[top[x]] > dep[top[y]]) {
77             seg.push_back({in[x] + 1, in[top[x]], pre[x]});
78             x = p[top[x]];
79         } else {
80             segr.push_back({in[top[y]], in[y] + 1, pre[y]});
81             y = p[top[y]];
82         }
83     }
84     if (dep[x] < dep[y]) {
85         int val = 0;
86         for (int i = 0; i < 30; i++) {
87             if (nxt[in[x]][i] <= in[y]) {
88                 val |= 1 << i;
89             }
90         }
91         seg.push_back({in[x], in[y] + 1, val});
92     } else {
93         int val = 0;
94         for (int i = 0; i < 30; i++) {
95             if (nxt[in[y]][i] <= in[x]) {
96                 val |= 1 << i;
97             }
98         }
99         seg.push_back({in[x] + 1, in[y], val});
100    }
101    int d = 0;
102    seg.insert(seg.end(), segr.rbegin(), segr.rend());
103    for (auto [l, r, v] : seg) {
104        d += abs(l - r);
105    }
106    auto get = [&]() {
107        vector<pair<int, int>> a;
108        int o = 0;
109        int cnt = 0;
110        for (auto [l, r, x] : seg) {
111            int no = o | x;
112            int diff = no ^ o;
113            o = no;
114            while (diff) {
115                int i = __builtin_ctz(diff);
116                diff ^= 1 << i;
117                if (l < r) {
118                    a.emplace_back(nxt[l][i] - l + cnt, 1);
119                } else {
120                    a.emplace_back(l - lst[l][i] + cnt, 1);
121                }
122            }
123        }
124        return a;
125    };
126    cout << get();
127 }

```

```

121             }
122         }
123         cnt += abs(l - r);
124     }
125     return a;
126 };
127 auto a = get();
128 reverse(seg.begin(), seg.end());
129 for (auto &[l, r, v] : seg) {
130     swap(l, r);
131 }
132 auto b = get();
133 for (auto &[x, y] : b) {
134     x = d - x;
135     y = -1;
136 }
137 a.insert(a.end(), b.begin(), b.end());
138 sort(a.begin(), a.end());
139 int res = b.size();
140 int ans = 0;
141 int lst = 0;
142 for (auto [u, v] : a) {
143     if (u > lst) {
144         ans = max(ans, res);
145     }
146     lst = u;
147     res += v;
148 }
149 if (d > lst) {
150     ans = max(ans, res);
151 }
152 cout << ans << " \n"[q == 0];
153 }
154 }
155 int main() {
156     ios::sync_with_stdio(false);
157     cin.tie(nullptr);
158     int t;
159     cin >> t;
160     while (t--) {
161         solve();
162     }
163     return 0;
164 }
```

### 573: Optimal Insertion

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given two arrays of integers  $a_1, a_2, \dots, a_n$  and  $b_1, b_2, \dots, b_m$ .

You need to insert all elements of  $b$  into  $a$  in an arbitrary way. As a result you will get an array  $c_1, c_2, \dots, c_{n+m}$  of size  $n + m$ .

Note that you are not allowed to change the order of elements in  $a$ , while you can insert elements of  $b$  at arbitrary positions. They can be inserted at the beginning, between any elements of  $a$ , or at the end. Moreover, elements of  $b$  can appear in the resulting array in any order.

What is the minimum possible number of inversions in the resulting array  $c$ ? Recall that an inversion is a pair of indices  $(i, j)$  such that  $i < j$  and  $c_i > c_j$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class Info, class Tag>
5 # struct LazySegmentTree;
6 # struct Tag;
7 # struct Info;
8 Info operator+(Info a, Info b) {
9     return {min(a.min, b.min)};
10 }template <typename T>
11 # struct Fenwick;
12 void solve() {
13     int n, m;
14     cin >> n >> m;
15     vector<int> a(n);
16     for (int i = 0; i < n; i++) {
17         cin >> a[i];
18     }
19     vector<int> b(m);
20     for (int i = 0; i < m; i++) {
21         cin >> b[i];
22     }
23     sort(b.begin(), b.end());
24     vector<int> p(n);
25     iota(p.begin(), p.end(), 0);
26     sort(p.begin(), p.end(),
27           [&](int i, int j) {
28               return a[i] < a[j];
29           });
30     auto v = a;
31     sort(v.begin(), v.end());
32     i64 ans = 0;
33     Fenwick<int> fen(n);
34     for (int i = 0; i < n; i++) {
35         int x = lower_bound(v.begin(), v.end(), a[i]) - v.begin();
36         ans += fen.rangeSum(x + 1, n);
37         fen.add(x, 1);
38     }
39     LazySegmentTree<Info, Tag> seg(n + 1);
40     for (int i = 0; i <= n; i++) {
41         seg.modify(i, {i});
42     }
43     for (int i = 0, j = 0; auto x : b) {
44         while (i < n && a[p[i]] <= x) {
45             seg.rangeApply(p[i] + 1, n + 1, {-1});
46             i++;
47         }
48         while (j < n && a[p[j]] < x) {
49             seg.rangeApply(0, p[j] + 1, {1});
50             j++;
51     }
52 }
```

```

51         }
52         ans += seg.rangeQuery(0, n + 1).min;
53     }
54     cout << ans << "\n";
55 }
56 int main() {
57     ios::sync_with_stdio(false);
58     cin.tie(nullptr);
59     int t;
60     cin >> t;
61     while (t--) {
62         solve();
63     }
64     return 0;
65 }
```

## 574: Arithmetic Operations

- Time limit: 5 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

You are given an array of integers  $a_1, a_2, \dots, a_n$ .

You can do the following operation any number of times (possibly zero):

Choose any index  $i$  and set  $a_i$  to any integer (positive, negative or 0).

What is the minimum number of operations needed to turn  $a$  into an arithmetic progression? The array  $a$  is an arithmetic progression if  $a_{i+1} - a_i = a_i - a_{i-1}$  for any  $2 \leq i \leq n - 1$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int B = 400;
5 constexpr int N = 1E5;
6 int cnt[(B + 10) * N];
7 int main() {
8     ios::sync_with_stdio(false);
9     cin.tie(nullptr);
10    int n;
11    cin >> n;
12    vector<int> a(n);
13    for (int i = 0; i < n; i++) {
14        cin >> a[i];
15    }
16    int ans = 0;
17    for (int d = -B; d <= B; d++) {
18        auto b = a;
19        for (int i = 0; i < n; i++) {
```

```

20         if (d > 0) {
21             b[i] += i * d;
22         } else {
23             b[i] += (n - i) * -d;
24         }
25         ans = max(ans, ++cnt[b[i]]);
26     }
27     for (int i = 0; i < n; i++) {
28         cnt[b[i]] = 0;
29     }
30 }
31 for (int i = 0; i < n; i++) {
32     for (int j = max(0, i - B); j < i; j++) {
33         if ((a[i] - a[j]) % (i - j) == 0) {
34             int v = (a[i] - a[j]) / (i - j) + N;
35             ans = max(ans, ++cnt[v] + 1);
36         }
37     }
38     for (int j = max(0, i - B); j < i; j++) {
39         if ((a[i] - a[j]) % (i - j) == 0) {
40             int v = (a[i] - a[j]) / (i - j) + N;
41             cnt[v] = 0;
42         }
43     }
44 }
45 ans = n - ans;
46 cout << ans << "\n";
47 return 0;
48 }
```

## 575: Exotic Queries

- Time limit: 4 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

AquaMoon gives RiverHamster a sequence of integers  $a_1, a_2, \dots, a_n$ , and RiverHamster gives you  $q$  queries. Each query is expressed by two integers  $l$  and  $r$ .

For each query independently, you can take any continuous segment of the sequence and subtract an identical non-negative value from all the numbers of this segment. You can do so multiple (possibly, zero) times. However, you may not choose two intersecting segments which are not included in one another. Your goal is to convert to 0 all numbers whose initial value was within the range  $[l, r]$ . You must do so in the minimum number of operations.

Please note that the queries are independent, the numbers in the array are restored to their initial values between the queries.

Formally, for each query, you are to find the smallest  $m$  such that there exists a sequence  $\{(x_j, y_j, z_j)\}_{j=1}^m$  satisfying the following conditions:

for any  $1 \leq j \leq m$ ,  $z_j \geq 0$  and  $1 \leq x_j \leq y_j \leq n$  (here  $[x_j, y_j]$  correspond to the segment of the sequence);

for any  $1 \leq j < k \leq m$ , it is true that  $[x_j, y_j] \subseteq [x_k, y_k]$ , or  $[x_k, y_k] \subseteq [x_j, y_j]$ , or  $[x_j, y_j] \cap [x_k, y_k] = \emptyset$ ;

for any  $1 \leq i \leq n$ , such that  $l \leq a_i \leq r$ , it is true that

$$a_i = \sum_{\substack{1 \leq j \leq m \\ x_j \leq i \leq y_j}} z_j.$$

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct Node;
5 Node *add(Node *t, int l, int r, int x) {
6     if (t) {
7         t = new Node(*t);
8     } else {
9         t = new Node;
10    }
11    t->cnt += 1;
12    if (r - l == 1) {
13        return t;
14    }
15    int m = (l + r) / 2;
16    if (x < m) {
17        t->l = add(t->l, l, m, x);
18    } else {
19        t->r = add(t->r, m, r, x);
20    }
21    return t;
22}
23 int query(Node *t1, Node *t2, int l, int r, int x) {
24     int cnt = (t2 ? t2->cnt : 0) - (t1 ? t1->cnt : 0);
25     if (cnt == 0 || l >= x) {
26         return -1;
27     }
28     if (r - l == 1) {
29         return l;
30     }
31     int m = (l + r) / 2;
32     int res = query(t1 ? t1->r : t1, t2 ? t2->r : t2, m, r, x);
33     if (res == -1) {
34         res = query(t1 ? t1->l : t1, t2 ? t2->l : t2, l, m, x);
35     }
36     return res;
37}
38 template <typename T>
39 # struct Fenwick;
40 int main() {
41     ios::sync_with_stdio(false);
42     cin.tie(nullptr);
43     int n, q;
44     cin >> n >> q;

```

```

45     vector<int> a(n);
46     for (int i = 0; i < n; i++) {
47         cin >> a[i];
48         a[i]--;
49     }
50     vector<vector<int>> pos(n);
51     for (int i = 0; i < n; i++) {
52         pos[a[i]].push_back(i);
53     }
54     vector<int> precnt(n + 1);
55     for (int i = 0; i < n; i++) {
56         precnt[i + 1] = precnt[i] + pos[i].size();
57     }
58     vector<Node *> tree(n + 1);
59     for (int i = 0; i < n; i++) {
60         tree[i + 1] = add(tree[i], 0, n, a[i]);
61     }
62     vector<array<int, 2>> point;
63     for (int v = 0; v < n; v++) {
64         for (int i = 1; i < pos[v].size(); i++) {
65             int x = query(tree[pos[v][i - 1] + 1], tree[pos[v][i]], 0, n, v);
66             point.push_back({x, v});
67         }
68     }
69     sort(point.begin(), point.end());
70     vector<int> ans(q);
71     vector<array<int, 3>> qry(q);
72     for (int i = 0; i < q; i++) {
73         int l, r;
74         cin >> l >> r;
75         l--;
76         ans[i] += precnt[r] - precnt[l];
77         qry[i] = {l, r, i};
78     }
79     Fenwick<int> fen(n);
80     sort(qry.begin(), qry.end());
81     for (int i = 0; auto [l, r, j] : qry) {
82         while (i < point.size() && point[i][0] < l) {
83             fen.add(point[i][1], 1);
84             i++;
85         }
86         ans[j] -= fen.rangeSum(l, r);
87     }
88     for (int i = 0; i < q; i++) {
89         cout << ans[i] << "\n";
90     }
91     return 0;
92 }
```

## 576: Almost Difference

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Let's denote a function

You are given an array  $a$  consisting of  $n$  integers. You have to calculate the sum of  $d(a_i, a_j)$  over all pairs  $(i, j)$  such that  $1 \leq i \leq j \leq n$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 using i128 = __int128;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n;
9     cin >> n;
10    i64 sum = 0;
11    i128 ans = 0;
12    map<int, int> cnt;
13    for (int i = 0; i < n; i++) {
14        int a;
15        cin >> a;
16        ans += 1LL * a * i - sum;
17        sum += a;
18        ans += cnt[a + 1];
19        ans -= cnt[a - 1];
20        cnt[a]++;
21    }
22    if (ans == 0) {
23        cout << 0;
24    }
25    if (ans < 0) {
26        ans = -ans;
27        cout << '-';
28    }
29    string s;
30    while (ans > 0) {
31        s += '0' + ans % 10;
32        ans /= 10;
33    }
34    reverse(s.begin(), s.end());
35    cout << s << "\n";
36    return 0;
37 }
```

## 577: Eyes Closed

- Time limit: 2.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vasya and Petya were tired of studying so they decided to play a game. Before the game begins Vasya looks at array  $a$  consisting of  $n$  integers. As soon as he remembers all elements of  $a$  the game begins.

Vasya closes his eyes and Petya does q actions of one of two types:

- 1) Petya says 4 integers  $l_1, r_1, l_2, r_2$  - boundaries of two non-intersecting segments. After that he swaps one random element from the  $[l_1, r_1]$  segment with another random element from the  $[l_2, r_2]$  segment.
- 2) Petya asks Vasya the sum of the elements of a in the  $[l, r]$  segment.

Vasya is a mathematician so he answers Petya the mathematical expectation of the sum of the elements in the segment.

Your task is to write a program which will answer the second type questions as Vasya would do it. In other words your program should print the mathematical expectation of the sum of the elements of a in the  $[l, r]$  segment for every second type query.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class Info, class Tag>
5 # struct LazySegmentTree;
6 # struct Tag;
7 # struct Info;
8 Info operator+(Info a, Info b) {
9     Info c;
10    c.sum = a.sum + b.sum;
11    c.cnt = a.cnt + b.cnt;
12    return c;
13 }
14 int main() {
15     ios::sync_with_stdio(false);
16     cin.tie(nullptr);
17     int n, q;
18     cin >> n >> q;
19     vector<int> a(n);
20     for (int i = 0; i < n; i++) {
21         cin >> a[i];
22     }
23     LazySegmentTree<Info, Tag> seg(n);
24     for (int i = 0; i < n; i++) {
25         seg.modify(i, {a[i], 1});
26     }
27     cout << fixed << setprecision(10);
28     for (int i = 0; i < q; i++) {
29         int t;
30         cin >> t;
31         if (t == 1) {
32             int l1, r1, l2, r2;
33             cin >> l1 >> r1 >> l2 >> r2;
34             l1--, l2--;
35             double s1 = seg.rangeQuery(l1, r1).sum;
36             double s2 = seg.rangeQuery(l2, r2).sum;
37             seg.rangeApply(l1, r1, {1.0 - 1.0 / (r1 - l1), s2 / (r2 - l2) / (r1 - l1)});
```

```

38         seg.rangeApply(l2, r2, {1.0 - 1.0 / (r2 - l2), s1 / (r2 - l2) / (r1 - l1)});
39     } else {
40         int l, r;
41         cin >> l >> r;
42         l--;
43         double ans = seg.rangeQuery(l, r).sum;
44         cout << ans << "\n";
45     }
46 }
47 return 0;
48 }
```

## 578: Ralph And His Tour in Binary Country

- Time limit: 2.5 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Ralph is in the Binary Country. The Binary Country consists of  $n$  cities and  $(n - 1)$  bidirectional roads connecting the cities. The roads are numbered from 1 to  $(n - 1)$ , the  $i$ -th road connects the city labeled (here  $x$  denotes the  $x$  rounded down to the nearest integer) and the city labeled  $(i + 1)$ , and the length of the  $i$ -th road is  $L_i$ .

Now Ralph gives you  $m$  queries. In each query he tells you some city  $A_i$  and an integer  $H_i$ . He wants to make some tours starting from this city. He can choose any city in the Binary Country (including  $A_i$ ) as the terminal city for a tour. He gains happiness  $(H_i - L)$  during a tour, where  $L$  is the distance between the city  $A_i$  and the terminal city.

Ralph is interested in tours from  $A_i$  in which he can gain positive happiness. For each query, compute the sum of happiness gains for all such tours.

Ralph will never take the same tour twice or more (in one query), he will never pass the same city twice or more in one tour.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<vector<int>> d(n);
10    vector<int> L(n);
11    for (int i = 1; i < n; i++) {
```

```

12     cin >> L[i];
13 }
14 for (int i = 0; i < n; i++) {
15     d[i].push_back(0);
16 }
17 for (int i = n - 1; i > 0; i--) {
18     auto &a = d[(i - 1) / 2];
19     int k = a.size();
20     a.insert(a.end(), d[i].begin(), d[i].end());
21     for (int j = k; j < a.size(); j++) {
22         a[j] += L[i];
23     }
24     inplace_merge(a.begin(), a.begin() + k, a.end());
25 }
26 vector<vector<i64>> s(n);
27 for (int i = 0; i < n; i++) {
28     s[i].resize(d[i].size());
29     for (int j = 0; j < d[i].size(); j++) {
30         if (j) {
31             s[i][j] = s[i][j - 1];
32         }
33         s[i][j] += d[i][j];
34     }
35 }
36 for (int i = 0; i < m; i++) {
37     int A, H;
38     cin >> A >> H;
39     A--;
40     i64 ans = 0;
41     int j = lower_bound(d[A].begin(), d[A].end(), H) - d[A].begin();
42     if (j > 0) {
43         ans += 1LL * H * j - s[A][j - 1];
44     }
45     while (A > 0) {
46         int B = (A - 1) / 2;
47         H -= L[B];
48         int C = 4 * B + 3 - A;
49         if (H > 0) {
50             ans += H;
51         }
52         if (C < n) {
53             int j = lower_bound(d[C].begin(), d[C].end(), H - L[C]) - d[C].begin();
54             if (j > 0) {
55                 ans += 1LL * (H - L[C]) * j - s[C][j - 1];
56             }
57         }
58         A = B;
59     }
60     cout << ans << "\n";
61 }
62 return 0;
63 }

```

**579: Envy**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

For a connected undirected weighted graph  $G$ , MST (minimum spanning tree) is a subgraph of  $G$  that contains all of  $G$ 's vertices, is a tree, and sum of its edges is minimum possible.

You are given a graph  $G$ . If you run a MST algorithm on graph it would give you only one MST and it causes other edges to become jealous. You are given some queries, each query contains a set of edges of graph  $G$ , and you should determine whether there is a MST containing all these edges or not.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, m;
9     cin >> n >> m;
10    vector<int> u(m), v(m), w(m);
11    for (int i = 0; i < m; i++) {
12        cin >> u[i] >> v[i] >> w[i];
13        u[i]--, v[i]--;
14    }
15    const int V = *max_element(w.begin(), w.end()) + 1;
16    vector<vector<int>> e(V);
17    for (int i = 0; i < m; i++) {
18        e[w[i]].push_back(i);
19    }
20    int q;
21    cin >> q;
22    vector<int> ans(q, 1);
23    vector<vector<pair<int, vector<int>>> qry(V);
24    DSU dsu(n);
25    for (int i = 0; i < q; i++) {
26        int k;
27        cin >> k;
28        vector<int> a(k);
29        for (int j = 0; j < k; j++) {
30            cin >> a[j];
31            a[j]--;
32        }
33        sort(a.begin(), a.end(),
34              [&](int i, int j) {
35                  return w[i] < w[j];
36              });
37        for (int l = 0, r = 0; l < k; l = r) {
38            while (r < k && w[a[l]] == w[a[r]]) {
39                r++;
40            }
41            qry[w[a[l]]].emplace_back(i, vector(a.begin() + l, a.begin() + r));
42        }
43    }
44    for (int c = 0; c < V; c++) {
45        for (auto [i, a] : qry[c]) {
46            int k = a.size();
47            vector<array<int, 2>> e;
48            e.reserve(k);
49            for (auto j : a) {
50                e.push_back({dsu.find(u[j]), dsu.find(v[j])});
51            }
52        }
53    }
54}
```

```

51         }
52     for (auto [u, v] : e) {
53         if (!dsu.merge(u, v)) {
54             ans[i] = 0;
55         }
56     }
57     for (auto [u, v] : e) {
58         dsu.f[u] = u;
59         dsu.f[v] = v;
60     }
61     for (auto i : e[c]) {
62         dsu.merge(u[i], v[i]);
63     }
64 }
65 for (int i = 0; i < q; i++) {
66     cout << (ans[i] ? "YES" : "NO") << "\n";
67 }
68 return 0;
69 }
70 }
```

## 580: Trees and Segments

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The teachers of the Summer Informatics School decided to plant  $n$  trees in a row, and it was decided to plant only oaks and firs. To do this, they made a plan, which can be represented as a binary string  $s$  of length  $n$ . If  $s_i = 0$ , then the  $i$ -th tree in the row should be an oak, and if  $s_i = 1$ , then the  $i$ -th tree in the row should be a fir.

The day of tree planting is tomorrow, and the day after tomorrow an inspector will come to the School. The inspector loves nature very much, and he will evaluate the beauty of the row as follows:

First, he will calculate  $l_0$  as the maximum number of consecutive oaks in the row (the maximum substring consisting of zeros in the plan  $s$ ). If there are no oaks in the row, then  $l_0 = 0$ .

Then, he will calculate  $l_1$  as the maximum number of consecutive firs in the row (the maximum substring consisting of ones in the plan  $s$ ). If there are no firs in the row, then  $l_1 = 0$ .

Finally, he will calculate the beauty of the row as  $a \cdot l_0 + l_1$  for some  $a$  - the inspector's favourite number.

The teachers know the value of the parameter  $a$ , but for security reasons they cannot tell it to you. They only told you that  $a$  is an integer from 1 to  $n$ .

Since the trees have not yet been planted, the teachers decided to change the type of no more than  $k$  trees to the opposite (i.e., change  $s_i$  from 0 to 1 or from 1 to 0 in the plan) in order to maximize the

beauty of the row of trees according to the inspector.

For each integer  $j$  from 1 to  $n$  answer the following question independently:

What is the maximum beauty of the row of trees that the teachers can achieve by changing the type of no more than  $k$  trees if the inspector's favourite number  $a$  is equal to  $j$ ?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     string s;
8     cin >> s;
9     vector<int> ans(n + 1);
10    auto work = [&]() {
11        vector f(n + 1, vector<int>(k + 1));
12        vector<int> g(n + 1, -1);
13        for (int i = n - 1; i >= 0; i--) {
14            f[i] = f[i + 1];
15            int cost = 0;
16            for (int j = i; j < n; j++) {
17                cost += (s[j] == '0');
18                if (cost <= k) {
19                    f[i][cost] = max(f[i][cost], j - i + 1);
20                }
21            }
22            for (int j = 1; j <= k; j++) {
23                f[i][j] = max(f[i][j], f[i][j - 1]);
24            }
25        }
26        vector<int> h(n + 1, k + 1);
27        h[0] = 0;
28        for (int i = 0; i <= n; i++) {
29            if (i > 0) {
30                int cost = 0;
31                for (int j = i - 1; j >= 0; j--) {
32                    cost += (s[j] == '1');
33                    h[i - j] = min(h[i - j], cost);
34                }
35            }
36            for (int j = 0; j <= n; j++) {
37                if (h[j] <= k) {
38                    g[j] = max(g[j], f[i][k - h[j]]);
39                }
40            }
41        }
42        for (int a = 1; a <= n; a++) {
43            for (int i = 0; i <= n; i++) {
44                if (g[i] != -1) {
45                    ans[a] = max(ans[a], i * a + g[i]);
46                }
47            }
48        }
49    };
50    work();
51    reverse(s.begin(), s.end());

```

```

52     work();
53     for (int i = 1; i <= n; i++) {
54         cout << ans[i] << " \n"[i == n];
55     }
56 }
57 int main() {
58     ios::sync_with_stdio(false);
59     cin.tie(nullptr);
60     int t;
61     cin >> t;
62     while (t--) {
63         solve();
64     }
65     return 0;
66 }
```

### 581: Max to the Right of Min

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a permutation  $p$  of length  $n$  - an array, consisting of integers from 1 to  $n$ , all distinct.

Let  $p_{l,r}$  denote a subarray - an array formed by writing down elements from index  $l$  to index  $r$ , inclusive.

Let  $\text{maxpos}_{l,r}$  denote the index of the maximum element on  $p_{l,r}$ . Similarly, let  $\text{minpos}_{l,r}$  denote the index of the minimum element on it.

Calculate the number of subarrays  $p_{l,r}$  such that  $\text{maxpos}_{l,r} > \text{minpos}_{l,r}$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> p(n);
10    for (int i = 0; i < n; i++) {
11        cin >> p[i];
12    }
13    vector<int> lmin(n, -1), rmin(n, n), lmax(n, -1), rmax(n, n);
14    vector<int> smin, smax;
15    for (int i = 0; i < n; i++) {
```

```

16     while (!smin.empty() && p[i] < p[smin.back()]) {
17         rmin[smin.back()] = i;
18         smin.pop_back();
19     }
20     if (!smin.empty()) {
21         lmin[i] = smin.back();
22     }
23     smin.push_back(i);
24     while (!smax.empty() && p[i] > p[smax.back()]) {
25         rmax[smax.back()] = i;
26         smax.pop_back();
27     }
28     if (!smax.empty()) {
29         lmax[i] = smax.back();
30     }
31     smax.push_back(i);
32 }
33 i64 ans = 0;
34 vector<int> s{-1};
35 vector<i64> sum{0LL};
36 for (int i = 0; i < n; i++) {
37     while (s.size() > 1 && p[i] < p[s.back()]) {
38         rmin[s.back()] = i;
39         s.pop_back();
40         sum.pop_back();
41     }
42     int l = upper_bound(s.begin(), s.end(), lmax[i]) - s.begin();
43     if (l < s.size()) {
44         ans += 1LL * (s[l] - max(s[l - 1], lmax[i]))
45         * (min(rmax[i], rmin[s[l]]) - i);
46         l++;
47         int m = partition_point(s.begin() + l, s.end(), [&](int x) {
48             return rmin[x] > rmax[i];
49         }) - s.begin();
50         ans += 1LL * (s[m - 1] - s[l - 1]) * rmax[i];
51         ans -= 1LL * (s.back() - s[l - 1]) * i;
52         ans += sum.back() - sum[m - 1];
53     }
54     sum.push_back(sum.back() + 1LL * (i - s.back()) * rmin[i]);
55     s.push_back(i);
56 }
57 cout << ans << "\n";
58 return 0;
59 }
```

## 582: Competition

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The secondary diagonal of a square matrix is a diagonal going from the top right to the bottom left corner. Let's define an  $n$ -degree staircase as a square matrix  $n \times n$  containing no squares above the secondary diagonal (the picture below shows a 5-degree staircase).

The squares of the  $n$ -degree staircase contain  $m$  sportsmen.

A sportsman needs one second to move to a side-neighboring square of the staircase. Before the beginning of the competition each sportsman must choose one of the shortest ways to the secondary diagonal.

After the starting whistle the competition begins and all sportsmen start moving along the chosen paths. When a sportsman reaches a cell of the secondary diagonal, he stops and moves no more. The competition ends when all sportsmen reach the secondary diagonal. The competition is considered successful if during it no two sportsmen were present in the same square simultaneously. Any square belonging to the secondary diagonal also cannot contain more than one sportsman. If a sportsman at the given moment of time leaves a square and another sportsman comes to it, then they are not considered to occupy the same square simultaneously. Note that other extreme cases (for example, two sportsmen moving towards each other) are impossible as the chosen ways are the shortest ones.

You are given positions of  $m$  sportsmen on the staircase. Your task is to choose among them the maximum number of sportsmen for who the competition can be successful, that is, so that there existed such choice of shortest ways for the sportsmen at which no two sportsmen find themselves in the same square simultaneously. All other sportsmen that are not chosen will be removed from the staircase before the competition starts.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<int> l(m), r(m);
10    vector<vector<int>> f(n);
11    for (int i = 0; i < m; i++) {
12        cin >> l[i] >> r[i];
13        l[i] = n - l[i];
14        f[l[i]].push_back(i);
15    }
16    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<> q;
17    vector<int> ans;
18    for (int i = 0; i < n; i++) {
19        for (auto j : f[i]) {
20            q.emplace(r[j], j);
21        }
22    }
23    while (!q.empty()) {
24        auto [r, j] = q.top();
25        q.pop();
26        if (r <= i) {
27            continue;
28        }
29        ans.push_back(j);}
```

```

29         break;
30     }
31 }
32 cout << ans.size() << "\n";
33 for (int i = 0; i < ans.size(); i++) {
34     cout << ans[i] + 1 << " \n"[i == ans.size() - 1];
35 }
36 return 0;
37 }
```

### 583: Tenzing and Triangle

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  pairwise-distinct points and a line  $x + y = k$  on a two-dimensional plane. The  $i$ -th point is at  $(x_i, y_i)$ . All points have non-negative coordinates and are strictly below the line. Alternatively,  $0 \leq x_i, y_i, x_i + y_i < k$ .

Tenzing wants to erase all the points. He can perform the following two operations:

**Draw triangle:** Tenzing will choose two non-negative integers  $a, b$  that satisfy  $a + b < k$ , then all points inside the triangle formed by lines  $x = a, y = b$  and  $x + y = k$  will be erased. It can be shown that this triangle is an isosceles right triangle. Let the side lengths of the triangle be  $l, l$  and  $\sqrt{2}l$  respectively. Then, the cost of this operation is  $l \cdot A$ . The blue area of the following picture describes the triangle with  $a = 1, b = 1$  with cost =  $1 \cdot A$ .

The blue area of the following picture describes the triangle with  $a = 1, b = 1$  with cost =  $1 \cdot A$ .

**Erase a specific point:** Tenzing will choose an integer  $i$  that satisfies  $1 \leq i \leq n$  and erase the point  $i$ . The cost of this operation is  $c_i$ .

Help Tenzing find the minimum cost to erase all of the points.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class Info, class Tag>
5 # struct LazySegmentTree;
6 # struct Tag;
7 # struct Info;
8 Info operator+(Info a, Info b) {
9     return {min(a.mn, b.mn)};
10 }
```

```

11 int main() {
12     ios::sync_with_stdio(false);
13     cin.tie(nullptr);
14     int n, k, A;
15     cin >> n >> k >> A;
16     vector<int> l(n), r(n), w(n);
17     for (int i = 0; i < n; i++) {
18         cin >> l[i] >> r[i] >> w[i];
19         r[i] = k - r[i];
20     }
21     vector<vector<int>> f(k + 1);
22     vector<i64> dp(k + 1);
23     for (int i = 0; i < n; i++) {
24         f[r[i]].push_back(i);
25     }
26     LazySegmentTree<Info, Tag> seg(k + 1);
27     seg.modify(0, {0});
28     for (int i = 1; i <= k; i++) {
29         for (auto j : f[i]) {
30             seg.rangeApply(0, l[j] + 1, {-w[j]});
31         }
32         dp[i] = min(dp[i - 1], seg.rangeQuery(0, i).mn + 1LL * i * A);
33         seg.modify(i, {dp[i] - 1LL * i * A});
34     }
35     auto ans = dp[k];
36     for (int i = 0; i < n; i++) {
37         ans += w[i];
38     }
39     cout << ans << "\n";
40     return 0;
41 }
```

### 584: Omsk Metro (hard version)

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. The only difference between the simple and hard versions is that in this version  $u$  can take any possible value.

As is known, Omsk is the capital of Berland. Like any capital, Omsk has a well-developed metro system. The Omsk metro consists of a certain number of stations connected by tunnels, and between any two stations there is exactly one path that passes through each of the tunnels no more than once. In other words, the metro is a tree.

To develop the metro and attract residents, the following system is used in Omsk. Each station has its own weight  $x \in \{-1, 1\}$ . If the station has a weight of  $-1$ , then when the station is visited by an Omsk resident, a fee of 1 burle is charged. If the weight of the station is  $1$ , then the Omsk resident is rewarded with 1 burle.

Omsk Metro currently has only one station with number 1 and weight  $x = 1$ . Every day, one of the following events occurs:

A new station with weight  $x$  is added to the station with number  $v_i$ , and it is assigned a number that is one greater than the number of existing stations.

Alex, who lives in Omsk, wonders: is there a subsegment† (possibly empty) of the path between vertices  $u$  and  $v$  such that, by traveling along it, exactly  $k$  burles can be earned (if  $k < 0$ , this means that  $k$  burles will have to be spent on travel). In other words, Alex is interested in whether there is such a subsegment of the path that the sum of the weights of the vertices in it is equal to  $k$ . Note that the subsegment can be empty, and then the sum is equal to 0.

You are a friend of Alex, so your task is to answer Alex's questions.

†Subsegment - continuous sequence of elements.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct Info;
5 Info operator+(Info a, Info b) {
6     Info c;
7     c.mxpre = max(a.mxpre, a.sum + b.mxpre);
8     c.mnpre = min(a.mnpre, a.sum + b.mnpre);
9     c.mxsuf = max(b.mxsuf, b.sum + a.mxsuf);
10    c.mnsuf = min(b.mnsuf, b.sum + a.mnsuf);
11    c.mx = max({a.mx, b.mx, a.mxsuf + b.mxpre});
12    c.mn = min({a.mn, b.mn, a.mnsuf + b.mnpre});
13    c.sum = a.sum + b.sum;
14    return c;
15 }
16 Info rev(Info a) {
17     swap(a.mxpre, a.mxsuf);
18     swap(a.mnpre, a.mnsuf);
19     return a;
20 }
21 void solve() {
22     int n;
23     cin >> n;
24     vector<int> p{-1}, x{1};
25     vector<array<int, 3>> qry;
26     for (int i = 0; i < n; i++) {
27         char o;
28         cin >> o;
29         if (o == '+') {
30             int v, xi;
31             cin >> v >> xi;
32             v--;
33             p.push_back(v);
34             x.push_back(xi);
35         } else {
36             int u, v, k;
37             cin >> u >> v >> k;
38             u--, v--;

```

```

39         qry.push_back({u, v, k});
40     }
41 }
42 n = p.size();
43 const int logn = __lg(n);
44 vector pp(logn + 1, vector<int>(n, -1));
45 vector f(logn + 1, vector<Info>(n));
46 vector<int> dep(n);
47 for (int i = 0; i < n; i++) {
48     if (i > 0) {
49         dep[i] = dep[p[i]] + 1;
50     }
51     pp[0][i] = p[i];
52     if (x[i] == 1) {
53         f[0][i].mx = f[0][i].mxpre = f[0][i].mxsuf = f[0][i].sum = 1;
54     } else {
55         f[0][i].mn = f[0][i].mnpree = f[0][i].mnsuf = f[0][i].sum = -1;
56     }
57     for (int j = 0; (2 << j) <= dep[i] + 1; j++) {
58         pp[j + 1][i] = pp[j][pp[j][i]];
59         f[j + 1][i] = f[j][i] + f[j][pp[j][i]];
60     }
61 }
62 auto query = [&](int x, int y) {
63     if (dep[x] < dep[y]) {
64         swap(x, y);
65     }
66     Info l, r;
67     for (int i = logn; i >= 0; i--) {
68         if (dep[x] - (1 << i) >= dep[y]) {
69             l = l + f[i][x];
70             x = pp[i][x];
71         }
72     }
73     if (x == y) {
74         return l + f[0][x];
75     }
76     for (int i = logn; i >= 0; i--) {
77         if (pp[i][x] != pp[i][y]) {
78             l = l + f[i][x];
79             r = r + f[i][y];
80             x = pp[i][x];
81             y = pp[i][y];
82         }
83     }
84     return l + f[1][x] + f[0][y] + rev(r);
85 };
86 for (auto [u, v, k] : qry) {
87     auto info = query(u, v);
88     if (info.mn <= k && k <= info.mx) {
89         cout << "YES\n";
90     } else {
91         cout << "NO\n";
92     }
93 }
94 int main() {
95     ios::sync_with_stdio(false);
96     cin.tie(nullptr);
97     int t;
98     cin >> t;
99     while (t--) {
100         solve();
101     }
102 }
```

```
103     return 0;
104 }
```

## 585: MEX of LCM

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  of length  $n$ . A positive integer  $x$  is called good if it is impossible to find a subsegment<sup>†</sup> of the array such that the least common multiple of all its elements is equal to  $x$ .

You need to find the smallest good integer.

A subsegment<sup>†</sup> of the array  $a$  is a set of elements  $a_l, a_{l+1}, \dots, a_r$  for some  $1 \leq l \leq r \leq n$ . We will denote such subsegment as  $[l, r]$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    int lim = 20 * n;
12    vector<bool> e(lim);
13    vector<int> l;
14    for (int i = 0; i < n; i++) {
15        vector<int> nl;
16        if (a[i] < lim) {
17            nl.push_back(a[i]);
18        }
19        for (auto x : l) {
20            i64 v = lcm(1LL * x, 1LL * a[i]);
21            if (v < lim) {
22                nl.push_back(v);
23            }
24        }
25        sort(nl.begin(), nl.end());
26        nl.erase(unique(nl.begin(), nl.end()), nl.end());
27        l = nl;
28        for (auto x : l) {
29            e[x] = true;
30        }
31    }
```

```

32     int ans = 1;
33     while (e[ans]) {
34         ans++;
35     }
36     cout << ans << "\n";
37 }
38 int main() {
39     ios::sync_with_stdio(false);
40     cin.tie(nullptr);
41     int t;
42     cin >> t;
43     while (t--) {
44         solve();
45     }
46     return 0;
47 }
```

## 586: Fill the Matrix

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There is a square matrix, consisting of  $n$  rows and  $n$  columns of cells, both numbered from 1 to  $n$ . The cells are colored white or black. Cells from 1 to  $a_i$  are black, and cells from  $a_i + 1$  to  $n$  are white, in the  $i$ -th column.

You want to place  $m$  integers in the matrix, from 1 to  $m$ . There are two rules:

each cell should contain at most one integer;

black cells should not contain integers.

The beauty of the matrix is the number of such  $j$  that  $j + 1$  is written in the same row, in the next column as  $j$  (in the neighbouring cell to the right).

What's the maximum possible beauty of the matrix?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
```

```

9         cin >> a[i];
10        a[i] = n - a[i];
11    }
12    i64 m;
13    cin >> m;
14    vector<int> l(n, -1), r(n, n);
15    vector<int> s;
16    for (int i = 0; i < n; i++) {
17        while (!s.empty() && a[i] < a[s.back()]) {
18            r[s.back()] = i;
19            s.pop_back();
20        }
21        if (!s.empty()) {
22            l[i] = s.back();
23        }
24        s.push_back(i);
25    }
26    vector<i64> cnt(n + 1);
27    for (int i = 0; i < n; i++) {
28        int len = r[i] - l[i] - 1;
29        int v = 0;
30        if (l[i] != -1) {
31            v = max(v, a[l[i]]);
32        }
33        if (r[i] != n) {
34            v = max(v, a[r[i]]);
35        }
36        cnt[len] += a[i] - v;
37    }
38    i64 ans = m;
39    i64 res = m;
40    for (int i = n; i >= 1; i--) {
41        if (i * cnt[i] < res) {
42            res -= i * cnt[i];
43            ans -= cnt[i];
44        } else {
45            ans -= (res + i - 1) / i;
46            break;
47        }
48    }
49    cout << ans << "\n";
50 }
51 int main() {
52     ios::sync_with_stdio(false);
53     cin.tie(nullptr);
54     int t;
55     cin >> t;
56     while (t--) {
57         solve();
58     }
59     return 0;
60 }

```

## dfs and similar

### 587: Lomsat gelral

- Time limit: 2 seconds
- Memory limit: 256 megabytes

- Input file: standard input
- Output file: standard output

You are given a rooted tree with root in vertex 1. Each vertex is coloured in some colour.

Let's call colour c dominating in the subtree of vertex v if there are no other colours that appear in the subtree of vertex v more times than colour c. So it's possible that two or more colours will be dominating in the subtree of some vertex.

The subtree of vertex v is the vertex v and all other vertices that contains vertex v in each path to the root.

For each vertex v find the sum of all dominating colours in the subtree of vertex v.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> c(n);
10    for (auto &x : c) {
11        cin >> x;
12        x--;
13    }
14    vector<int> siz(n);
15    vector<vector<int>> adj(n);
16    for (int i = 1; i < n; i++) {
17        int x, y;
18        cin >> x >> y;
19        x--, y--;
20        adj[x].push_back(y);
21        adj[y].push_back(x);
22    }
23    function<void(int, int)> dfs = [&](int x, int p) {
24        if (p != -1) {
25            adj[x].erase(find(adj[x].begin(), adj[x].end(), p));
26        }
27        siz[x] = 1;
28        for (auto &y : adj[x]) {
29            dfs(y, x);
30            siz[x] += siz[y];
31            if (siz[y] > siz[adj[x][0]]) {
32                swap(y, adj[x][0]);
33            }
34        }
35    };
36    dfs(0, -1);
37    vector<int> cnt(n);
38    vector<i64> sum(n+1);
39    vector<i64> ans(n);
40    int mx = 0;
41    sum[0] = 1LL * n * (n+1) / 2;

```

```

42     auto addv = [&](int x, int t) {
43         sum[cnt[x]] -= x+1;
44         cnt[x] += t;
45         mx = max(mx, cnt[x]);
46         sum[cnt[x]] += x+1;
47         while (!sum[mx]) {
48             mx--;
49         }
50     };
51     function<void(int, int)> add = [&](int x, int t) {
52         addv(c[x], t);
53         for (auto y : adj[x]) {
54             add(y, t);
55         }
56     };
57     function<void(int)> calc = [&](int x) {
58         for (auto y : adj[x]) {
59             if (y != adj[x][0]) {
60                 calc(y);
61                 add(y, -1);
62             }
63         }
64         if (!adj[x].empty()) {
65             calc(adj[x][0]);
66             for (auto y : adj[x]) {
67                 if (y != adj[x][0]) {
68                     add(y, 1);
69                 }
70             }
71         }
72         addv(c[x], 1);
73         ans[x] = sum[mx];
74     };
75     calc(0);
76     add(0, -1);
77     for (int i = 0; i < n; i++) {
78         cout << ans[i] << " \n"[i == n-1];
79     }
80     return 0;
81 }
```

## 588: Ring Road 2

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

It is well known that Berland has  $n$  cities, which form the Silver ring - cities  $i$  and  $i + 1$  ( $1 \leq i < n$ ) are connected by a road, as well as the cities  $n$  and  $1$ . The government have decided to build  $m$  new roads. The list of the roads to build was prepared. Each road will connect two cities. Each road should be a curve which lies inside or outside the ring. New roads will have no common points with the ring (except the endpoints of the road).

Now the designers of the constructing plan wonder if it is possible to build the roads in such a way

that no two roads intersect (note that the roads may intersect at their endpoints). If it is possible to do, which roads should be inside the ring, and which should be outside?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<int> a(m), b(m);
10    for (int i = 0; i < m; i++) {
11        cin >> a[i] >> b[i];
12        a[i]--, b[i]--;
13        if (a[i] > b[i]) {
14            swap(a[i], b[i]);
15        }
16    }
17    vector<vector<int>> adj(m);
18    for (int i = 0; i < m; i++) {
19        for (int j = 0; j < m; j++) {
20            if (a[i] < a[j] && a[i] < b[i] && b[i] < b[j]) {
21                adj[i].push_back(j);
22                adj[j].push_back(i);
23            }
24        }
25    }
26    vector<int> c(m, -1);
27    for (int i = 0; i < m; i++) {
28        if (c[i] == -1) {
29            queue<int> q;
30            q.push(i);
31            c[i] = 0;
32            while (!q.empty()) {
33                int x = q.front();
34                q.pop();
35                for (auto y : adj[x]) {
36                    if (c[y] == -1) {
37                        c[y] = !c[x];
38                        q.push(y);
39                    } else if (c[x] == c[y]) {
40                        cout << "Impossible\n";
41                        return 0;
42                    }
43                }
44            }
45        }
46    }
47    for (int i = 0; i < m; i++) {
48        cout << "io"[c[i]];
49    }
50    cout << "\n";
51    return 0;
52 }
```

### 589: Between

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an integer  $n$ , as well as  $m$  pairs of integers  $(a_i, b_i)$ , where  $1 \leq a_i, b_i \leq n$ ,  $a_i \neq b_i$ .

You want to construct a sequence satisfying the following requirements:

All elements in the sequence are integers between 1 and  $n$ .

There is exactly one element with value 1 in the sequence.

For each  $i$  ( $1 \leq i \leq m$ ), between any two elements (on different positions) in the sequence with value  $a_i$ , there is at least one element with value  $b_i$ .

The sequence constructed has the maximum length among all possible sequences satisfying the above properties.

Sometimes, it is possible that such a sequence can be arbitrarily long, in which case you should output “INFINITE”. Otherwise, you should output “FINITE” and the sequence itself. If there are multiple possible constructions that yield the maximum length, output any.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<int> a(m), b(m);
8     vector<vector<int>> adj(n);
9     for (int i = 0; i < m; i++) {
10         cin >> a[i] >> b[i];
11         a[i]--, b[i]--;
12         adj[b[i]].push_back(a[i]);
13     }
14     vector<int> dis(n, -1);
15     vector<int> q;
16     q.push_back(0);
17     dis[0] = 0;
18     for (int i = 0; i < q.size(); i++) {
19         int x = q[i];
20         for (auto y : adj[x]) {
21             if (dis[y] == -1) {
22                 dis[y] = dis[x] + 1;
23                 q.push_back(y);
24             }
25         }
26     }

```

```

27     if (q.size() < n) {
28         cout << "INFINITE\n";
29         return;
30     }
31     vector<vector<int>> f(n);
32     for (auto x : q) {
33         for (int i = 0; i <= dis[x]; i++) {
34             f[i].push_back(x);
35         }
36     }
37     vector<int> ans;
38     for (auto &v : f) {
39         ans.insert(ans.end(), v.rbegin(), v.rend());
40     }
41     cout << "FINITE\n";
42     cout << ans.size() << "\n";
43     for (int i = 0; i < ans.size(); i++) {
44         cout << ans[i] + 1 << " \n"[i == ans.size() - 1];
45     }
46 }
47 int main() {
48     ios::sync_with_stdio(false);
49     cin.tie(nullptr);
50     int t;
51     cin >> t;
52     while (t--) {
53         solve();
54     }
55     return 0;
56 }
```

## 590: Scheme

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

To learn as soon as possible the latest news about their favourite fundamentally new operating system, BolgenOS community from Nizhni Tagil decided to develop a scheme. According to this scheme a community member, who is the first to learn the news, calls some other member, the latter, in his turn, calls some third member, and so on; i.e. a person with index  $i$  got a person with index  $f_i$ , to whom he has to call, if he learns the news. With time BolgenOS community members understood that their scheme doesn't work sometimes - there were cases when some members didn't learn the news at all. Now they want to supplement the scheme: they add into the scheme some instructions of type  $(x_i, y_i)$ , which mean that person  $x_i$  has to call person  $y_i$  as well. What is the minimum amount of instructions that they need to add so, that at the end everyone learns the news, no matter who is the first to learn it?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n;
9     cin >> n;
10    vector<int> f(n), in(n);
11    DSU dsu(n);
12    for (int i = 0; i < n; i++) {
13        cin >> f[i];
14        f[i]--;
15        in[f[i]] += 1;
16        dsu.merge(i, f[i]);
17    }
18    vector<int> a(n);
19    vector<int> vis(n, -1);
20    for (int i = 0; i < n; i++) {
21        int j = i;
22        while (vis[j] == -1) {
23            vis[j] = i;
24            j = f[j];
25        }
26        if (vis[j] == i) {
27            a[dsu.leader(j)] = j;
28        }
29    }
30    bool conn = true;
31    for (int i = 0; i < n; i++) {
32        if (!dsu.same(0, i) || in[i] != 1) {
33            conn = false;
34            break;
35        }
36    }
37    if (conn) {
38        cout << 0 << "\n";
39        return 0;
40    }
41    vector<bool> used(n);
42    vector<array<int, 2>> ed;
43    vector<array<int, 2>> ans;
44    for (int i = 0; i < n; i++) {
45        if (in[i] == 0) {
46            if (!used[dsu.leader(i)]) {
47                ed.push_back({a[dsu.leader(i)], i});
48                used[dsu.leader(i)] = true;
49            } else {
50                ans.push_back({a[dsu.leader(i)], i});
51            }
52        }
53    }
54    for (int i = 0; i < n; i++) {
55        if (!used[dsu.leader(i)]) {
56            ed.push_back({i, i});
57            used[dsu.leader(i)] = true;
58        }
59    }
60    for (int i = 0; i < ed.size(); i++) {
```

```

61         ans.push_back({ed[i][0], ed[(i + 1) % ed.size()][1]});
62     }
63     cout << ans.size() << "\n";
64     for (auto [x, y] : ans) {
65         cout << x + 1 << " " << y + 1 << "\n";
66     }
67     return 0;
68 }
```

**591: Sum Over Zero**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a_1, a_2, \dots, a_n$  of  $n$  integers. Consider  $S$  as a set of segments satisfying the following conditions.

Each element of  $S$  should be in form  $[x, y]$ , where  $x$  and  $y$  are integers between 1 and  $n$ , inclusive, and  $x \leq y$ .

No two segments in  $S$  intersect with each other. Two segments  $[a, b]$  and  $[c, d]$  intersect if and only if there exists an integer  $x$  such that  $a \leq x \leq b$  and  $c \leq x \leq d$ .

For each  $[x, y]$  in  $S$ ,  $a_x + a_{x+1} + \dots + a_y \geq 0$ .

The length of the segment  $[x, y]$  is defined as  $y - x + 1$ .  $f(S)$  is defined as the sum of the lengths of every element in  $S$ . In a formal way,  $f(S) = \sum_{[x,y] \in S} (y - x + 1)$ . Note that if  $S$  is empty,  $f(S)$  is 0.

What is the maximum  $f(S)$  among all possible  $S$ ?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template <typename T>
5 # struct Fenwick;
6 # struct Max;
7 int main() {
8     ios::sync_with_stdio(false);
9     cin.tie(nullptr);
10    int n;
11    cin >> n;
12    vector<int> a(n);
13    for (int i = 0; i < n; i++) {
14        cin >> a[i];
15    }
16    vector<i64> s(n + 1);
17    for (int i = 0; i < n; i++) {
```

```

18         s[i + 1] = s[i] + a[i];
19     }
20     auto v = s;
21     sort(v.begin(), v.end());
22     Fenwick<Max> fen(n + 1);
23     vector<int> dp(n + 1);
24     for (int i = 0; i <= n; i++) {
25         int x = lower_bound(v.begin(), v.end(), s[i]) - v.begin();
26         if (i == 0) {
27             dp[i] = 0;
28         } else {
29             dp[i] = max(dp[i - 1], i + fen.sum(x + 1).v);
30         }
31         fen.add(x, dp[i] - i);
32     }
33     cout << dp[n] << "\n";
34     return 0;
35 }
```

## 592: Tokens on Graph

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an undirected connected graph, some vertices of which contain tokens and/or bonuses. Consider a game involving one player - you.

You can move tokens according to the following rules:

At the beginning of the game, you can make exactly one turn: move any token to any adjacent vertex.

If the movement of the token ended on the bonus, then you are allowed to make another turn with any other token.

You can use different bonuses in any order. The same bonus can be used an unlimited number of times. Bonuses do not move during the game.

There can be several tokens in one vertex at the same time, but initially there is no more than one token in each vertex.

The vertex with number 1 is the finish vertex, and your task is to determine whether it is possible to hit it with any token by making turns with the tiles according to the rules described above. If a token is initially located at the vertex of 1, then the game is considered already won.

Move from the 8-th vertex to the 6-th.

Move from the 7-th vertex to the 5-th.

Move from the 6-th vertex to the 4-th.

Move from the 5-th vertex to the 6-th.

Move from the 4-th vertex to the 2-nd.

Move from the 6-th vertex to the 4-th.

Move from the 2-nd vertex to the 1-st vertex, which is the finish.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     int p, b;
8     cin >> p >> b;
9     vector<bool> piece(n), bonus(n);
10    for (int i = 0; i < p; i++) {
11        int x;
12        cin >> x;
13        x--;
14        piece[x] = true;
15    }
16    for (int i = 0; i < b; i++) {
17        int x;
18        cin >> x;
19        x--;
20        bonus[x] = true;
21    }
22    vector<vector<int>> adj(n);
23    for (int i = 0; i < m; i++) {
24        int u, v;
25        cin >> u >> v;
26        u--, v--;
27        adj[u].push_back(v);
28        adj[v].push_back(u);
29    }
30    vector<bool> infb(n);
31    for (int i = 0; i < n; i++) {
32        for (auto j : adj[i]) {
33            if (bonus[i] && bonus[j]) {
34                infb[i] = true;
35            }
36        }
37    }
38    vector<int> type(n);
39    for (int i = 0; i < n; i++) {
40        if (!piece[i]) {
41            continue;
42        }
43        for (auto j : adj[i]) {
44            if (bonus[j]) {
45                type[i] = 1;
46            }
47            if (infb[j]) {
48                type[i] = 2;
```

```

49             break;
50         }
51     }
52 }
53 vector<int> dis(n, -1);
54 queue<int> q;
55 dis[0] = 0;
56 q.push(0);
57 while (!q.empty()) {
58     int x = q.front();
59     q.pop();
60     for (auto y : adj[x]) {
61         if (bonus[y] && dis[y] == -1) {
62             dis[y] = dis[x] + 1;
63             q.push(y);
64         }
65     }
66 }
67 array<int, 3> cnt {};
68 for (int i = 0; i < n; i++) {
69     cnt[type[i]]++;
70 }
71 for (int i = 0; i < n; i++) {
72     if (piece[i]) {
73         cnt[type[i]]--;
74         int res = dis[i];
75         for (auto j : adj[i]) {
76             if (dis[j] != -1 && (res == -1 || res > dis[j] + 1)) {
77                 res = dis[j] + 1;
78             }
79         }
80         if (res != -1 && (cnt[2] > 0 || cnt[1] >= res - 1)) {
81             cout << "YES\n";
82             return;
83         }
84         cnt[type[i]]++;
85     }
86 }
87 cout << "NO\n";
88 }
89 int main() {
90     ios::sync_with_stdio(false);
91     cin.tie(nullptr);
92     int t;
93     cin >> t;
94     while (t--) {
95         solve();
96     }
97     return 0;
98 }
```

**593: Edge Reverse**

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You will be given a weighted directed graph of  $n$  nodes and  $m$  directed edges, where the  $i$ -th edge has a weight of  $w_i$  ( $1 \leq i \leq m$ ).

You need to reverse some edges of this graph so that there is at least one node in the graph from which every other node is reachable. The cost of these reversals is equal to the maximum weight of all reversed edges. If no edge reversal is required, assume the cost to be 0.

It is guaranteed that no self-loop or duplicate edge exists.

Find the minimum cost required for completing the task. If there is no solution, print a single integer  $-1$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<int> u(m), v(m), w(m);
8     for (int i = 0; i < m; i++) {
9         cin >> u[i] >> v[i] >> w[i];
10        u[i]--, v[i]--;
11    }
12    auto check = [&](int x) {
13        vector<vector<int>> adj(n);
14        for (int i = 0; i < m; i++) {
15            adj[u[i]].push_back(v[i]);
16            if (w[i] <= x) adj[v[i]].push_back(u[i]);
17        }
18        int cnt = 0, comp = 0;
19        vector<int> dfn(n, -1), low(n), bel(n, -1), stk;
20        auto dfs = [&](auto self, int x) -> void {
21            stk.push_back(x);
22            dfn[x] = low[x] = cnt++;
23            for (auto y : adj[x]) {
24                if (dfn[y] == -1) {
25                    self(self, y);
26                    low[x] = min(low[x], low[y]);
27                } else if (bel[y] == -1) {
28                    low[x] = min(low[x], dfn[y]);
29                }
30            }
31            if (dfn[x] == low[x]) {
32                int y;
33                do {
34                    y = stk.back();
35                    bel[y] = comp;
36                    stk.pop_back();
37                } while (y != x);
38                comp++;
39            }
40        };
41        for (int i = 0; i < n; i++) {
42            if (dfn[i] == -1) {
43                dfs(dfs, i);
44            }
45        }
46    };
47 }
```

```

45         }
46         vector<int> deg(comp);
47         for (int i = 0; i < n; i++) {
48             for (auto j : adj[i]) {
49                 if (bel[i] != bel[j]) {
50                     deg[bel[j]]++;
51                 }
52             }
53         }
54         return count(deg.begin(), deg.end(), 0) == 1;
55     };
56     int lo = 0, hi = 1E9 + 1;
57     while (lo < hi) {
58         int x = (lo + hi) / 2;
59         if (check(x)) hi = x;
60         else lo = x + 1;
61     }
62     int ans = lo;
63     if (ans > 1E9) ans = -1;
64     cout << ans << "\n";
65 }
66 int main() {
67     ios::sync_with_stdio(false);
68     cin.tie(nullptr);
69     int t;
70     cin >> t;
71     while (t--) {
72         solve();
73     }
74     return 0;
75 }
```

**dp****594: Small Permutation Problem (Easy Version)**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In the easy version, the  $a_i$  are in the range  $[0, n]$ ; in the hard version, the  $a_i$  are in the range  $[-1, n]$  and the definition of good permutation is slightly different. You can make hacks only if all versions of the problem are solved.

You are given an integer  $n$  and an array  $a_1, a_2 \dots, a_n$  of integers in the range  $[0, n]$ .

A permutation  $p_1, p_2, \dots, p_n$  of  $[1, 2, \dots, n]$  is good if, for each  $i$ , the following condition is true:

the number of values  $\leq i$  in  $[p_1, p_2, \dots, p_i]$  is exactly  $a_i$ .

Count the good permutations of  $[1, 2, \dots, n]$ , modulo 998 244 353.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 void solve() {
36     int n;
37     cin >> n;
38     vector<int> a(n);
39     for (int i = 0; i < n; i++) {
40         cin >> a[i];
41     }
42     int lst = 0, lstd = 0;
43     Z ans = 1;
44     auto work = [&](int x, int y) {
45         assert(x >= lst);
46         if (y < lstd || ans == 0 || x < y) {
47             ans = 0;
48             return;
49         }
50         Z res = 0;
51         for (int i = 0; i <= y - lstd; i++) {
52             res += comb.binom(x - lst, i) * comb.binom(lst - lstd, i) * comb.fact(i)
53                 * comb.binom(x - lst, y - lstd - i) * comb.binom(x - lstd - i, y - lstd -
54                     i) * comb.fact(y - lstd - i);
55         }
56         ans *= res;
57         lst = x;
58         lstd = y;
59     };
59     for (int i = 0; i < n; i++) {
60         if (a[i] != -1) {
61             work(i + 1, a[i]);
62         }
63     }
64 }
```

```

62         }
63     }
64     work(n, n);
65     cout << ans << "\n";
66 }
67 int main() {
68     ios::sync_with_stdio(false);
69     cin.tie(nullptr);
70     int t;
71     cin >> t;
72     while (t--) {
73         solve();
74     }
75     return 0;
76 }
```

### 595: Compressed Tree

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a tree consisting of  $n$  vertices. A number is written on each vertex; the number on vertex  $i$  is equal to  $a_i$ .

You can perform the following operation any number of times (possibly zero):

choose a vertex which has at most 1 incident edge and remove this vertex from the tree.

Note that you can delete all vertices.

After all operations are done, you're compressing the tree. The compression process is done as follows. While there is a vertex having exactly 2 incident edges in the tree, perform the following operation:

delete this vertex, connect its neighbors with an edge.

It can be shown that if there are multiple ways to choose a vertex to delete during the compression process, the resulting tree is still the same.

Your task is to calculate the maximum possible sum of numbers written on vertices after applying the aforementioned operation any number of times, and then compressing the tree.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
```

```

4  void solve() {
5      int n;
6      cin >> n;
7      vector<int> a(n);
8      for (int i = 0; i < n; i++) {
9          cin >> a[i];
10     }
11     vector<vector<int>> adj(n);
12     for (int i = 1; i < n; i++) {
13         int u, v;
14         cin >> u >> v;
15         u--, v--;
16         adj[u].push_back(v);
17         adj[v].push_back(u);
18     }
19     vector<i64> dp(n);
20     i64 ans = 0;
21     auto dfs = [&](auto self, int x, int p) -> void {
22         dp[x] = a[x];
23         array<i64, 3> max;
24         max.fill(-1E18);
25         i64 pos = 0;
26         for (auto y : adj[x]) {
27             if (y == p) {
28                 continue;
29             }
30             self(self, y, x);
31             dp[x] = max(dp[x], dp[y]);
32             i64 v = dp[y];
33             for (int i = 0; i < 3; i++) {
34                 if (v > max[i]) {
35                     swap(v, max[i]);
36                 }
37             }
38             pos += max(0LL, v);
39         }
40         dp[x] = max(dp[x], max[0] + max[1] + max(0LL, max[2]) + pos + a[x]);
41         ans = max(ans, max[0] + a[x]);
42         ans = max(ans, 1LL * a[x]);
43         ans = max(ans, max[0] + max[1]);
44         ans = max(ans, max[0] + max[1] + max[2] + pos + a[x]);
45     };
46     dfs(dfs, 0, -1);
47     cout << ans << "\n";
48 }
49 int main() {
50     ios::sync_with_stdio(false);
51     cin.tie(nullptr);
52     int t;
53     cin >> t;
54     while (t--) {
55         solve();
56     }
57     return 0;
58 }
```

### 596: Rubik's Cube Coloring (hard version)

- Time limit: 3 seconds
- Memory limit: 256 megabytes

- Input file: standard input
- Output file: standard output

It is the hard version of the problem. The difference is that in this version, there are nodes with already chosen colors.

Theofanis is starving, and he wants to eat his favorite food, sheftalia. However, he should first finish his homework. Can you help him with this problem?

You have a perfect binary tree of  $2^k - 1$  nodes - a binary tree where all vertices  $i$  from 1 to  $2^{k-1} - 1$  have exactly two children: vertices  $2i$  and  $2i + 1$ . Vertices from  $2^{k-1}$  to  $2^k - 1$  don't have any children. You want to color its vertices with the 6 Rubik's cube colors (White, Green, Red, Blue, Orange and Yellow).

Let's call a coloring good when all edges connect nodes with colors that are neighboring sides in the Rubik's cube.

More formally:

a white node can not be neighboring with white and yellow nodes;

a yellow node can not be neighboring with white and yellow nodes;

a green node can not be neighboring with green and blue nodes;

a blue node can not be neighboring with green and blue nodes;

a red node can not be neighboring with red and orange nodes;

an orange node can not be neighboring with red and orange nodes;

However, there are  $n$  special nodes in the tree, colors of which are already chosen.

You want to calculate the number of the good colorings of the binary tree. Two colorings are considered different if at least one node is colored with a different color.

The answer may be too large, so output the answer modulo  $10^9 + 7$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
```

```

14  constexpr i64 mul(i64 a, i64 b, i64 p) {
15      i64 res = a * b - i64(1.L * a * b / p) * p;
16      res %= p;
17      if (res < 0) {
18          res += p;
19      }
20      return res;
21  }
22  template<i64 P>
23  # struct MLong;
24  template<>
25  i64 MLong<0LL>::Mod = i64(1E18) + 9;
26  template<int P>
27  # struct MInt;
28  template<>
29  int MInt<0>::Mod = 998244353;
30  template<int V, int P>
31  constexpr MInt<P> CInv = MInt<P>(V).inv();
32  constexpr int P = 1000000007;
33  using Z = MInt<P>;
34  int main() {
35      ios::sync_with_stdio(false);
36      cin.tie(nullptr);
37      int k;
38      cin >> k;
39      int n;
40      cin >> n;
41      map<i64, int> f;
42      map<string, int> id;
43      id["white"] = 0;
44      id["yellow"] = 1;
45      id["green"] = 2;
46      id["blue"] = 3;
47      id["red"] = 4;
48      id["orange"] = 5;
49      for (int i = 0; i < n; i++) {
50          i64 x;
51          cin >> x;
52          string s;
53          cin >> s;
54          f[x] = id[s];
55          while (x > 1) {
56              x /= 2;
57              if (!f.contains(x)) {
58                  f[x] = -1;
59              } else {
60                  break;
61              }
62          }
63      }
64      vector<array<Z, 6>> g(k);
65      g[k - 1].fill(1);
66      for (int i = k - 2; i >= 0; i--) {
67          auto sum = accumulate(g[i + 1].begin(), g[i + 1].end(), Z(0));
68          for (int j = 0; j < 6; j++) {
69              Z v = sum - g[i + 1][j] - g[i + 1][j ^ 1];
70              g[i][j] = v * v;
71          }
72      }
73      auto get = [&](auto self, i64 x) {
74          int d = __lg(x);
75          if (!f.contains(x)) {
76              return g[d];
77          }

```

```

78         if (d == k - 1) {
79             array<Z, 6> dp{};
80             int c = f[x];
81             dp[c] = 1;
82             return dp;
83         }
84         auto l = self(self, 2 * x), r = self(self, 2 * x + 1);
85         auto suml = accumulate(l.begin(), l.end(), Z(0));
86         auto sumr = accumulate(r.begin(), r.end(), Z(0));
87         array<Z, 6> dp{};
88         int c = f[x];
89         for (int i = 0; i < 6; i++) {
90             if (c == i || c == -1) {
91                 Z vl = suml - l[i] - l[i ^ 1];
92                 Z vr = sumr - r[i] - r[i ^ 1];
93                 dp[i] = vl * vr;
94             }
95         }
96         return dp;
97     };
98     auto ans = get(get, 1);
99     auto sum = accumulate(ans.begin(), ans.end(), Z(0));
100    cout << sum << "\n";
101    return 0;
102 }
```

## 597: Another MEX Problem

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array of integers  $a$  of size  $n$ . You can choose a set of non-overlapping subarrays of the given array (note that some elements may be not included in any subarray, this is allowed). For each selected subarray, calculate the MEX of its elements, and then calculate the bitwise XOR of all the obtained MEX values. What is the maximum bitwise XOR that can be obtained?

The MEX (minimum excluded) of an array is the smallest non-negative integer that does not belong to the array. For instance:

The MEX of  $[2, 2, 1]$  is 0, because 0 does not belong to the array.

The MEX of  $[3, 1, 0, 1]$  is 2, because 0 and 1 belong to the array, but 2 does not.

The MEX of  $[0, 3, 1, 2]$  is 4, because 0, 1, 2 and 3 belong to the array, but 4 does not.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    int k = 1;
12    while (k <= n) {
13        k *= 2;
14    }
15    vector<vector<int>> pre(n + 1, vector<int>(k));
16    pre[0][0] = 1;
17    vector<int> mex(n);
18    vector<int> cnt(n + 1);
19    vector<int> res(k);
20    for (int i = 1; i <= n; i++) {
21        vector<int> nmex(n);
22        cnt.assign(n + 1, 0);
23        int t = 0;
24        for (int j = i - 1; j >= 0; j--) {
25            cnt[a[j]]++;
26            while (cnt[t]) {
27                t++;
28            }
29            nmex[j] = t;
30        }
31        for (int j = i - 1; j >= 0; j--) {
32            bool p1 = j < i - 1 && (j == i - 2 || mex[j] > mex[j + 1]);
33            bool p2 = (j == i - 1 || nmex[j] > nmex[j + 1]);
34            if (p1 != p2 || (mex[j] != nmex[j])) {
35                if (p1) {
36                    for (int v = 0; v < k; v++) {
37                        if (pre[j][v]) {
38                            res[v ^ mex[j]]--;
39                        }
40                    }
41                }
42                if (p2) {
43                    for (int v = 0; v < k; v++) {
44                        if (pre[j][v]) {
45                            res[v ^ nmex[j]]++;
46                        }
47                    }
48                }
49            }
50        }
51        for (int v = 0; v < k; v++) {
52            pre[i][v] = pre[i - 1][v] || res[v];
53        }
54        mex = move(nmex);
55    }
56    int ans = k - 1;
57    while (!pre[n][ans]) {
58        ans--;
59    }
60    cout << ans << "\n";
61 }
62 int main() {
63     ios::sync_with_stdio(false);
64     cin.tie(nullptr);

```

```

65     int t;
66     cin >> t;
67     while (t--) {
68         solve();
69     }
70     return 0;
71 }
```

### 598: Imagination Castle

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Given a chessboard of size  $N \times M$  ( $N$  rows and  $M$  columns). Each row is numbered from 1 to  $N$  from top to bottom and each column is numbered from 1 to  $M$  from left to right. The tile in row  $r$  and column  $c$  is denoted as  $(r, c)$ . There exists  $K$  distinct special tiles on the chessboard with the  $i$ -th special tile being tile  $(X_i, Y_i)$ . It is guaranteed that tile  $(1, 1)$  is not a special tile.

A new chess piece has been invented, which is the castle. The castle moves similarly to a rook in regular chess, but slightly differently. In one move, a castle that is on some tile can only move to another tile in the same row or in the same column, but only in the right direction or the down direction. Formally, in one move, the castle on tile  $(r, c)$  can only move to tile  $(r', c')$  if and only if one of the following two conditions is satisfied:

$$r' = r \text{ and } c' > c.$$

$$c' = c \text{ and } r' > r.$$

Chaneka and Bhinneka will play a game. In the beginning of the game, there is a castle in tile  $(1, 1)$ . The two players will play alternatingly with Chaneka going first. In one turn, the player on that turn must move the castle following the movement rules of the castle.

If a player moves the castle to a special tile on her turn, then that player wins the game and the game ends. If on a turn, the castle cannot be moved, the player on that turn loses and the game ends.

Given the size of the board and the locations of each special tile. Determine the winner of this game if Chaneka and Bhinneka plays optimally.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
```

```

4  int main() {
5      ios::sync_with_stdio(false);
6      cin.tie(nullptr);
7      int N, M, K;
8      cin >> N >> M >> K;
9      set<int> s;
10     vector<vector<int>> a(N);
11     for (int i = 0; i < K; i++) {
12         int x, y;
13         cin >> x >> y;
14         x--, y--;
15         a[x].push_back(y);
16     }
17     for (int i = 0; i < M; i++) {
18         s.insert(i);
19     }
20     for (int i = N - 1; i >= 0; i--) {
21         if (!s.empty()) {
22             int x = *s.rbegin();
23             bool ok = true;
24             for (auto y : a[i]) {
25                 if (y > x) {
26                     ok = false;
27                 }
28             }
29             if (ok) {
30                 s.erase(x);
31                 if (i == 0 && x == 0) {
32                     cout << "Bhinneka\n";
33                     return 0;
34                 }
35             }
36             for (auto y : a[i]) {
37                 s.erase(y);
38             }
39         }
40     }
41     cout << "Chaneka\n";
42     return 0;
43 }
```

## 599: Digital Wallet

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

There are  $N$  arrays, each array has  $M$  positive integer elements. The  $j$ -th element of the  $i$ -th array is  $A_{i,j}$ .

Initially, Chaneka's digital wallet contains 0 money. Given an integer  $K$ . Chaneka will do  $M - K + 1$  operations. In the  $p$ -th operation, Chaneka does the following procedure:

Choose any array. Let's say Chaneka chooses the  $x$ -th array.

Choose an index  $y$  in that array such that  $p \leq y \leq p + K - 1$ .

Add the value of  $A_{x,y}$  to the total money in the wallet.

Change the value of  $A_{x,y}$  into 0.

Determine the maximum total money that can be earned!

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class Info>
5 # struct SegmentTree;
6 constexpr int inf = 1E9;
7 # struct Info;
8 Info operator+(Info a, Info b) {
9     Info c;
10    c.ans = min({a.ans, b.ans, a.pre + b.suf});
11    c.cnt = a.cnt + b.cnt;
12    c.pre = min(a.pre - b.cnt, b.pre);
13    c.suf = min(a.suf, b.suf - a.cnt);
14    return c;
15 }
16 int main() {
17     ios::sync_with_stdio(false);
18     cin.tie(nullptr);
19     int N, M, K;
20     cin >> N >> M >> K;
21     vector<array<int, 2>> A(N * M);
22     for (int i = 0; i < N; i++) {
23         for (int j = 0; j < M; j++) {
24             int x;
25             cin >> x;
26             A[i * M + j] = {x, j};
27         }
28     }
29     sort(A.begin(), A.end(), greater());
30     SegmentTree<Info> seg(M);
31     for (int i = 0; i < M; i++) {
32         seg.modify(i, {inf, 0, -max(0, i - K + 1) + 1, min(i, M - K)});
33     }
34     vector<int> cnt(M);
35     i64 ans = 0;
36     for (auto [v, i] : A) {
37         cnt[i]++;
38         seg.modify(i, {min(i, M - K) - max(0, i - K + 1) + 1 - cnt[i], cnt[i], -max(0, i - K + 1) + 1 - cnt[i], min(i, M - K) - cnt[i]});
39         if (seg.rangeQuery(0, M).ans < 0) {
40             cnt[i]--;
41             seg.modify(i, {min(i, M - K) - max(0, i - K + 1) + 1 - cnt[i], cnt[i], -max(0, i - K + 1) + 1 - cnt[i], min(i, M - K) - cnt[i]});
42         } else {
43             ans += v;
44         }
45     }
46     cout << ans << "\n";
47     return 0;
48 }
```

## 600: Non-Intersecting Subpermutations

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given two integers  $n$  and  $k$ .

For an array of length  $n$ , let's define its cost as the maximum number of contiguous subarrays of this array that can be chosen so that:

each element belongs to at most one subarray;

the length of each subarray is exactly  $k$ ;

each subarray contains each integer from 1 to  $k$  exactly once.

For example, if  $n = 10$ ,  $k = 3$  and the array is  $[1, 2, 1, 3, 2, 3, 2, 3, 1, 3]$ , its cost is 2 because, for example, we can choose the subarrays from the 2-nd element to the 4-th element and from the 7-th element to the 9-th element, and we can show that it's impossible to choose more than 2 subarrays.

Calculate the sum of costs over all arrays of length  $n$  consisting of integers from 1 to  $k$ , and print it modulo 998244353.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>

```

```

27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int n, k;
38     cin >> n >> k;
39     vector<array<Z, 2>> dp(k);
40     dp[0][0] = 1;
41     for (int i = 0; i < n; i++) {
42         vector<array<Z, 2>> g(k);
43         array<Z, 2> res;
44         for (int j = k - 1; j >= 0; j--) {
45             if (j < k - 1) {
46                 g[j + 1][0] += (k - j) * dp[j][0];
47                 g[j + 1][1] += (k - j) * dp[j][1];
48             } else {
49                 g[0][0] += dp[j][0];
50                 g[0][1] += dp[j][1] + dp[j][0];
51             }
52             if (j > 0) {
53                 res[0] += dp[j][0];
54                 res[1] += dp[j][1];
55                 g[j][0] += res[0];
56                 g[j][1] += res[1];
57             }
58         }
59         dp = move(g);
60     }
61     Z ans = 0;
62     for (int i = 0; i < k; i++) {
63         ans += dp[i][1];
64     }
65     cout << ans << "\n";
66     return 0;
67 }

```

## 601: Balanced String

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a binary string  $s$  (a binary string is a string consisting of characters 0 and/or 1).

Let's call a binary string balanced if the number of subsequences 01 (the number of indices  $i$  and  $j$  such that  $1 \leq i < j \leq n$ ,  $s_i = 0$  and  $s_j = 1$ ) equals to the number of subsequences 10 (the number of indices  $k$  and  $l$  such that  $1 \leq k < l \leq n$ ,  $s_k = 1$  and  $s_l = 0$ ) in it.

For example, the string 1000110 is balanced, because both the number of subsequences 01 and the

number of subsequences 10 are equal to 6. On the other hand, 11010 is not balanced, because the number of subsequences 01 is 1, but the number of subsequences 10 is 5.

You can perform the following operation any number of times: choose two characters in  $s$  and swap them. Your task is to calculate the minimum number of operations to make the string  $s$  balanced.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int inf = 1E9;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     string s;
9     cin >> s;
10    int n = s.size();
11    int c1 = count(s.begin(), s.end(), '1'), c0 = n - c1;
12    int need = (n * (n - 1) / 2 - c0 * (c0 - 1) / 2 + c1 * (c1 - 1) / 2) / 2;
13    vector<int> dp(c1 + 1, vector<int>(need + 1, inf));
14    dp[0][0] = 0;
15    for (int i = 0; i < n; i++) {
16        for (int j = c1 - 1; j >= 0; j--) {
17            for (int k = 0; k + i <= need; k++) {
18                dp[j + 1][k + i] = min(dp[j + 1][k + i], dp[j][k] + (s[i] == '0'));
19            }
20        }
21    }
22    cout << dp[c1][need] << "\n";
23    return 0;
24 }
```

## 602: Railguns

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Tema is playing a very interesting computer game.

During the next mission, Tema's character found himself on an unfamiliar planet. Unlike Earth, this planet is flat and can be represented as an  $n \times m$  rectangle.

Tema's character is located at the point with coordinates  $(0, 0)$ . In order to successfully complete the mission, he needs to reach the point with coordinates  $(n, m)$  alive.

Let the character of the computer game be located at the coordinate  $(i, j)$ . Every second, starting from the first, Tema can:

either use vertical hyperjump technology, after which his character will end up at coordinate  $(i + 1, j)$  at the end of the second;

or use horizontal hyperjump technology, after which his character will end up at coordinate  $(i, j + 1)$  at the end of the second;

or Tema can choose not to make a hyperjump, in which case his character will not move during this second;

The aliens that inhabit this planet are very dangerous and hostile. Therefore, they will shoot from their railguns  $r$  times.

Each shot completely penetrates one coordinate vertically or horizontally. If the character is in the line of its impact at the time of the shot (at the end of the second), he dies.

Since Tema looked at the game's source code, he knows complete information about each shot - the time, the penetrated coordinate, and the direction of the shot.

What is the minimum time for the character to reach the desired point? If he is doomed to die and cannot reach the point with coordinates  $(n, m)$ , output  $-1$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m, r;
6     cin >> n >> m >> r;
7     vector<array<int, 3>> shot(r);
8     vector<int> tm;
9     for (int i = 0; i < r; i++) {
10         int t, d, p;
11         cin >> t >> d >> p;
12         shot[i] = {t, d, p};
13         tm.push_back(t);
14     }
15     sort(tm.begin(), tm.end());
16     tm.erase(unique(tm.begin(), tm.end()), tm.end());
17     r = tm.size();
18     vector<vector<bool>> sh(n + 1, vector(m + 1, vector(r + 1, false)));
19     for (auto [t, d, p] : shot) {
20         t = lower_bound(tm.begin(), tm.end(), t) - tm.begin();
21         if (d == 1) {
22             for (int j = 0; j <= m; j++) {
23                 sh[p][j][t] = true;
24             }
25         } else {

```

```
26         for (int i = 0; i <= n; i++) {
27             sh[i][p][t] = true;
28         }
29     }
30 }
31 vector<vector<vector<int>> dp(n + 1, vector<vector<int>(m + 1, vector<int>(r + 1, -1)));
32 dp[0][0][0] = 0;
33 for (int i = 0; i <= n; i++) {
34     for (int j = 0; j <= m; j++) {
35         if (i > 0) {
36             for (int k = 0; k <= r; k++) {
37                 if (dp[i - 1][j][k] != -1) {
38                     if (dp[i][j][k] == -1 || dp[i][j][k] > dp[i - 1][j][k] + 1) {
39                         dp[i][j][k] = dp[i - 1][j][k] + 1;
40                     }
41                     if (k < r && !sh[i][j][k]) {
42                         dp[i][j][k + 1] = tm[k];
43                     }
44                 }
45             }
46         }
47         if (j > 0) {
48             for (int k = 0; k <= r; k++) {
49                 if (dp[i][j - 1][k] != -1) {
50                     if (dp[i][j][k] == -1 || dp[i][j][k] > dp[i][j - 1][k] + 1) {
51                         dp[i][j][k] = dp[i][j - 1][k] + 1;
52                     }
53                     if (k < r && !sh[i][j][k]) {
54                         dp[i][j][k + 1] = tm[k];
55                     }
56                 }
57             }
58         }
59         for (int k = 0; k < r; k++) {
60             if (dp[i][j][k] >= tm[k]) {
61                 dp[i][j][k] = -1;
62             }
63         }
64         for (int k = 0; k < r; k++) {
65             if (dp[i][j][k] != -1 && !sh[i][j][k]) {
66                 dp[i][j][k + 1] = tm[k];
67             }
68         }
69     }
70     int ans = -1;
71     for (int k = 0; k <= r; k++) {
72         if (dp[n][m][k] != -1) {
73             ans = dp[n][m][k];
74             break;
75         }
76     }
77 }
78 cout << ans << "\n";
79 }
80 int main() {
81     ios::sync_with_stdio(false);
82     cin.tie(nullptr);
83     int t;
84     cin >> t;
85     while (t--) {
86         solve();
87     }
88     return 0;
89 }
```

## 603: Combinatorics Problem

- Time limit: 4 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

Recall that the binomial coefficient  $\binom{x}{y}$  is calculated as follows ( $x$  and  $y$  are non-negative integers):

if  $x < y$ , then  $\binom{x}{y} = 0$ ;

otherwise,  $\binom{x}{y} = \frac{x!}{y!(x-y)!}$ .

You are given an array  $a_1, a_2, \dots, a_n$  and an integer  $k$ . You have to calculate a new array  $b_1, b_2, \dots, b_n$ , where

$$b_1 = (\binom{1}{k} \cdot a_1) \bmod 998244353;$$

$$b_2 = (\binom{2}{k} \cdot a_1 + \binom{1}{k} \cdot a_2) \bmod 998244353;$$

$$b_3 = (\binom{3}{k} \cdot a_1 + \binom{2}{k} \cdot a_2 + \binom{1}{k} \cdot a_3) \bmod 998244353, \text{ and so on.}$$

$$\text{Formally, } b_i = \left( \sum_{j=1}^i \binom{i-j+1}{k} \cdot a_j \right) \bmod 998244353.$$

Note that the array is given in a modified way, and you have to output it in a modified way as well.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 998244353;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, x, y, m, k;
9     cin >> n;
10    vector<int> a(n);
11    cin >> a[0] >> x >> y >> m >> k;
12    for (int i = 1; i < n; i++) {
13        a[i] = (1LL * a[i - 1] * x + y) % m;
14    }
15    for (int i = 0; i <= k; i++) {
16        for (int j = 1; j < n; j++) {
17            a[j] = (a[j] + a[j - 1]) % P;
18        }
19    }
20    i64 ans = 0;
21    for (int i = k - 1; i < n; i++) {
22        ans ^= 1LL * (i + 1) * a[i - k + 1];
23    }

```

```

24     cout << ans << "\n";
25     return 0;
26 }
```

## 604: Petya and Coloring

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Little Petya loves counting. He wants to count the number of ways to paint a rectangular checkered board of size  $n \times m$  ( $n$  rows,  $m$  columns) in  $k$  colors. Besides, the coloring should have the following property: for any vertical line that passes along the grid lines and divides the board in two non-empty parts the number of distinct colors in both these parts should be the same. Help Petya to count these colorings.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
```

```

34 # struct Comb comb;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int n, m, k;
39     cin >> n >> m >> k;
40     vector<vector<Z>> S(n + 1, vector<Z>(n + 1));
41     for (int i = 0; i <= n; i++) {
42         S[i][0] = (!i);
43         for (int j = 1; j <= i; j++) {
44             S[i][j] = S[i - 1][j] * j + S[i - 1][j - 1];
45         }
46     }
47     if (m == 1) {
48         cout << power(Z(k), n) << "\n";
49         return 0;
50     }
51     Z ans = 0;
52     for (int i = 0; i <= n; i++) {
53         for (int j = 0; i + j <= n; j++) {
54             ans += comb.binom(k, i) * comb.binom(k - i, j) * comb.binom(k - i - j, j) *
55                 comb.fac(i + j)
56                 * comb.fac(i + j) * S[n][i + j] * S[n][i + j] * power(Z(i), n * (m - 2));
57         }
58     }
59     cout << ans << "\n";
60 }

```

## 605: Nuclear Fusion

- Time limit: 1.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There is the following puzzle popular among nuclear physicists.

A reactor contains a set of  $n$  atoms of some chemical elements. We shall understand the phrase “atomic number” as the number of this atom’s element in the periodic table of the chemical elements.

You are allowed to take any two different atoms and fuse a new one from them. That results in a new atom, whose number is equal to the sum of the numbers of original atoms. The fusion operation can be performed several times.

The aim is getting a new pre-given set of  $k$  atoms.

The puzzle’s difficulty is that it is only allowed to fuse two atoms into one, it is not allowed to split an atom into several atoms. You are suggested to try to solve the puzzle.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

57         t.push_back(a[f[i]]);
58     }
59     reverse(t.begin(), t.end());
60     for (int i = 0, j = 0, v = 0; i < n; i++) {
61         if (v) {
62             cout << "+";
63         }
64         cout << elem[t[i]];
65         v += t[i];
66         if (v == b[j]) {
67             cout << "->" << elem[b[j]] << "\n";
68             v = 0;
69             j++;
70         }
71     }
72     return 0;
73 }
```

## 606: Lesson Timetable

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

When Petya has free from computer games time, he attends university classes. Every day the lessons on Petyas faculty consist of two double classes. The floor where the lessons take place is a long corridor with M classrooms numbered from 1 to M, situated along it.

All the students of Petyas year are divided into N groups. Petya has noticed recently that these groups timetable has the following peculiarity: the number of the classroom where the first lesson of a group takes place does not exceed the number of the classroom where the second lesson of this group takes place.

Once Petya decided to count the number of ways in which one can make a lesson timetable for all these groups. The timetable is a set of  $2N$  numbers: for each group the number of the rooms where the first and the second lessons take place. Unfortunately, he quickly lost the track of his calculations and decided to count only the timetables that satisfy the following conditions:

- 1) On the first lesson in classroom  $i$  exactly  $X_i$  groups must be present.
- 2) In classroom  $i$  no more than  $Y_i$  groups may be placed.

Help Petya count the number of timetables satisfying all those conditions As there can be a lot of such timetables, output modulo  $10^9 + 7$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 n = int(input())
2 x = list(map(int, input().split()))
3 y = list(map(int, input().split()))
4 P = 10 ** 9 + 7
5 N = sum(x)
6 fac = [0] * (N+1)
7 invfac = [0] * (N+1)
8 fac[0] = 1
9 for i in range(1, N+1) :
10     fac[i] = fac[i - 1] * i % P
11 invfac[N] = pow(fac[N], P - 2, P)
12 for i in range(N, 0, -1) :
13     invfac[i - 1] = invfac[i] * i % P
14 pre = 0
15 dp = [0] * (N+1)
16 dp[0] = 1
17 for i in range(n) :
18     g = [0] * (N+1)
19     for j in range(pre+1) :
20         res = pre+x[i]-j
21         for k in range(min(res+1, y[i]+1)) :
22             g[j+k] = (g[j+k] + dp[j] * invfac[k] * fac[res] * invfac[res - k]) % P
23     pre += x[i]
24     dp = g
25 ans = dp[N] * fac[N] % P
26 for i in x :
27     ans = ans * invfac[i] % P
28 print(ans)

```

## 607: Bath Queue

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  students living in the campus. Every morning all students wake up at the same time and go to wash. There are  $m$  rooms with wash basins. The  $i$ -th of these rooms contains  $a_i$  wash basins. Every student independently select one the rooms with equal probability and goes to it. After all students selected their rooms, students in each room divide into queues by the number of wash basins so that the size of the largest queue is the least possible. Calculate the expected value of the size of the largest queue among all rooms.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;

```

```

3  using i64 = long long;
4  int main() {
5      ios::sync_with_stdio(false);
6      cin.tie(nullptr);
7      int n, m;
8      cin >> n >> m;
9      vector<int> a(m);
10     for (int i = 0; i < m; i++) {
11         cin >> a[i];
12     }
13     vector<vector<double>> binom(n + 1, vector<double>(n + 1));
14     for (int i = 0; i <= n; i++) {
15         binom[i][0] = 1;
16         for (int j = 1; j <= i; j++) {
17             binom[i][j] = binom[i - 1][j] + binom[i - 1][j - 1];
18         }
19     }
20     vector<vector<double>> dp(n + 1, vector<double>(n + 1));
21     dp[0][0] = 1;
22     for (int i = 0; i < m; i++) {
23         vector<vector<double>> g(n + 1, vector<double>(n + 1));
24         for (int x = 0; x <= n; x++) {
25             for (int y = 0; y <= x; y++) {
26                 for (int z = 0; x + z <= n; z++) {
27                     g[x + z][max(y, (z + a[i] - 1) / a[i])] += dp[x][y] * binom[x + z][x];
28                 }
29             }
30         }
31         dp = g;
32     }
33     double ans = 0;
34     for (int i = 0; i <= n; i++) {
35         ans += dp[n][i] * i;
36     }
37     ans /= pow(m, n);
38     cout << fixed << setprecision(10);
39     cout << ans << "\n";
40     return 0;
41 }

```

## 608: Chain Chips

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given an undirected graph consisting of  $n$  vertices and  $n - 1$  edges. The  $i$ -th edge has weight  $a_i$ ; it connects the vertices  $i$  and  $i + 1$ .

Initially, each vertex contains a chip. Each chip has an integer written on it; the integer written on the chip in the  $i$ -th vertex is  $i$ .

In one operation, you can choose a chip (if there are multiple chips in a single vertex, you may choose any one of them) and move it along one of the edges of the graph. The cost of this operation is equal to

the weight of the edge.

The cost of the graph is the minimum cost of a sequence of such operations that meets the following condition:

after all operations are performed, each vertex contains exactly one chip, and the integer on each chip is not equal to the index of the vertex where that chip is located.

You are given  $q$  queries of the form:

$k \ x$  - change the weight of the  $k$ -th edge (the one which connects the vertices  $k$  and  $k + 1$ ) to  $x$ .

After each query, print the cost of the graph. Note that you don't actually move any chips; when you compute the cost, the chips are on their initial positions.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class Info,
5          class Merge = plus<Info>>
6 # struct SegmentTree;
7 constexpr i64 inf = 1E18;
8 # struct Info;
9 Info operator+(Info a, Info b) {
10     Info c;
11     c.a[0][0] = c.a[1][1] = inf;
12     for (int i = 0; i < 2; i++) {
13         for (int j = 0; j < 2; j++) {
14             for (int k = 0; k < 2; k++) {
15                 c.a[i][j] = min(c.a[i][j], a.a[i][k] + b.a[k][j]);
16             }
17         }
18     }
19     return c;
20 }
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     int n;
25     cin >> n;
26     vector<int> a(n - 1);
27     SegmentTree<Info> seg(n - 1);
28     for (int i = 0; i < n - 1; i++) {
29         cin >> a[i];
30         seg.modify(i, a[i]);
31     }
32     int q;
33     cin >> q;
34     while (q--) {
35         int x, y;
36         cin >> x >> y;
37         x--;
38         seg.modify(x, y);
39         cout << 2 * seg.rangeQuery(0, n - 1).a[0][1] << "\n";

```

```

40     }
41     return 0;
42 }
```

## 609: Sequence

- Time limit: 1 second
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

Little Petya likes to play very much. And most of all he likes to play the following game:

He is given a sequence of N integer numbers. At each step it is allowed to increase the value of any number by 1 or to decrease it by 1. The goal of the game is to make the sequence non-decreasing with the smallest number of steps. Petya is not good at math, so he asks for your help.

The sequence a is called non-decreasing if  $a_1 \leq a_2 \leq \dots \leq a_N$  holds, where N is the length of the sequence.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int N;
8     cin >> N;
9     priority_queue<int> pq;
10    vector<int> a(N);
11    for (int i = 0; i < N; i++) {
12        cin >> a[i];
13    }
14    i64 ans = 0;
15    for (int i = 0; i < N; i++) {
16        if (pq.empty() || a[i] > pq.top()) {
17            pq.push(a[i]);
18        } else {
19            ans += pq.top() - a[i];
20            pq.push(a[i]);
21            pq.push(a[i]);
22            pq.pop();
23        }
24    }
25    cout << ans << "\n";
26    return 0;
27 }
```

## 610: Vlad and the Nice Paths (hard version)

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

This is hard version of the problem, it differs from the easy one only by constraints on  $n$  and  $k$ .

Vlad found a row of  $n$  tiles and the integer  $k$ . The tiles are indexed from left to right and the  $i$ -th tile has the color  $c_i$ . After a little thought, he decided what to do with it.

You can start from any tile and jump to any number of tiles right, forming the path  $p$ . Let's call the path  $p$  of length  $m$  nice if:

$p$  can be divided into blocks of length exactly  $k$ , that is,  $m$  is divisible by  $k$ ;

$c_{p_1} = c_{p_2} = \dots = c_{p_k}$ ;

$c_{p_{k+1}} = c_{p_{k+2}} = \dots = c_{p_{2k}}$ ;

...

$c_{p_{m-k+1}} = c_{p_{m-k+2}} = \dots = c_{p_m}$ ;

Your task is to find the number of nice paths of maximum length. Since this number may be too large, print it modulo  $10^9 + 7$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
```

```

22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 void update(pair<int, Z> &a, pair<int, Z> b) {
36     if (b.first > a.first) {
37         a = b;
38     } else if (a.first == b.first) {
39         a.second += b.second;
40     }
41 }
42 void solve() {
43     int n, k;
44     cin >> n >> k;
45     vector<int> c(n);
46     for (int i = 0; i < n; i++) {
47         cin >> c[i];
48     }
49     vector<pair<int, Z>> dp(n + 1);
50     dp[0] = {0, 1};
51     pair<int, Z> ans{0, 1};
52     for (int i = 1; i <= n; i++) {
53         int cnt = 0;
54         for (int j = i - 1; j >= 0; j--) {
55             cnt += (c[j] == c[i - 1]);
56             if (cnt >= k) {
57                 update(dp[i], {dp[j].first + 1, dp[j].second * comb.binom(cnt - 1, k - 1)
58                         });
59             }
60             update(ans, dp[i]);
61         }
62         cout << ans.second << "\n";
63     }
64     int main() {
65         ios::sync_with_stdio(false);
66         cin.tie(nullptr);
67         int t;
68         cin >> t;
69         while (t--) {
70             solve();
71         }
72         return 0;
73     }

```

## 611: A Simple Task

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

Given a simple graph, output the number of simple cycles in it. A simple cycle is a cycle with no repeated vertices or edges.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector g(n, vector<bool>(n));
10    for (int i = 0; i < m; i++) {
11        int x, y;
12        cin >> x >> y;
13        x--, y--;
14        g[x][y] = g[y][x] = true;
15    }
16    i64 ans = 0;
17    for (int N = 3; N <= n; N++) {
18        vector dp(1 << N, vector(N, 0LL));
19        dp[1 << (N - 1)][N - 1] = 1;
20        for (int s = (1 << (N - 1)); s < (1 << N); s++) {
21            for (int i = 0; i < N; i++) {
22                if (__builtin_popcount(s) >= 3 && g[N - 1][i]) {
23                    ans += dp[s][i];
24                }
25                if (dp[s][i]) {
26                    for (int j = 0; j < N; j++) {
27                        if (g[i][j] && (~s >> j & 1)) {
28                            dp[s | 1 << j][j] += dp[s][i];
29                        }
30                    }
31                }
32            }
33        }
34    }
35    ans /= 2;
36    cout << ans << "\n";
37    return 0;
38 }
```

## 612: There Should Be a Lot of Maximums

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a tree (a connected graph without cycles). Each vertex of the tree contains an integer.

Let's define the MAD (maximum double) parameter of the tree as the maximum integer that occurs in the vertices of the tree at least 2 times. If no number occurs in the tree more than once, then we assume MAD = 0.

Note that if you remove an edge from the tree, it splits into two trees. Let's compute the MAD parameters of the two trees and take the maximum of the two values. Let the result be the value of the deleted edge.

For each edge, find its value. Note that we don't actually delete any edges from the tree, the values are to be found independently.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct HLD;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n;
9     cin >> n;
10    vector<int> A(n - 1), B(n - 1);
11    HLD t(n);
12    for (int i = 1; i < n; i++) {
13        int u, v;
14        cin >> u >> v;
15        u--, v--;
16        A[i - 1] = u;
17        B[i - 1] = v;
18        t.addEdge(u, v);
19    }
20    vector<int> a(n);
21    map<int, vector<int>, greater<>> f;
22    for (int i = 0; i < n; i++) {
23        cin >> a[i];
24        f[a[i]].push_back(i);
25    }
26    vector<int> ans(n);
27    int S = -1, T = -1;
28    int min = 0;
29    for (auto [v, e] : f) {
30        if (e.size() < 2) {
31            continue;
32        }
33        if (e.size() >= 3) {
34            min = v;
35            break;
36        }
37        int A = e[0], B = e[1];
38        if (S == -1) {
39            t.work(A);
40            ans.assign(n, v);
41            S = A, T = B;
42            for (int i = T; i != S; i = t.parent[i]) {

```

```

43             ans[i] = 0;
44         }
45     } else {
46         while (S != T && t.rootedLca(T, A, B) != T) {
47             ans[T] = v;
48             T = t.parent[T];
49         }
50         while (S != T && t.rootedLca(S, A, B) != S) {
51             S = t.rootedParent(T, S);
52             ans[S] = v;
53         }
54     }
55 }
56 if (S == -1) {
57     t.work(0);
58 }
59 for (int i = 0; i < n - 1; i++) {
60     if (t.parent[A[i]] == B[i]) {
61         swap(A[i], B[i]);
62     }
63     cout << max(ans[B[i]], min) << "\n";
64 }
65 return 0;
66 }
```

### 613: Minibuses on Venus (easy version)

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the easy version of the problem. The only difference between the three versions is the constraints on  $n$  and  $k$ . You can make hacks only if all versions of the problem are solved.

Maxim is a minibus driver on Venus.

To ride on Maxim's minibus, you need a ticket. Each ticket has a number consisting of  $n$  digits. However, as we know, the residents of Venus use a numeral system with base  $k$ , rather than the decimal system. Therefore, the ticket number can be considered as a sequence of  $n$  integers from 0 to  $k - 1$ , inclusive.

The residents of Venus consider a ticket to be lucky if there is a digit on it that is equal to the sum of the remaining digits, modulo  $k$ . For example, if  $k = 10$ , then the ticket 7135 is lucky because  $7 + 1 + 5 \equiv 3 \pmod{10}$ . On the other hand, the ticket 7136 is not lucky because no digit is equal to the sum of the others modulo 10.

Once, while on a trip, Maxim wondered: how many lucky tickets exist? At the same time, Maxim understands that this number can be very large, so he is interested only in the answer modulo some prime number  $m$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 using Z = MInt<0>;
33 vector<Z> operator*(vector<Z> a, vector<Z> b){
34     int n = a.size();
35     vector<Z> c(n);
36     for (int i = 0; i < n; i++) {
37         for (int j = 0; j < n; j++) {
38             c[(i + j) % n] += a[i] * b[j];
39         }
40     }
41     return c;
42 }
43 vector<Z> power(vector<Z> a, i64 n) {
44     vector<Z> res(a.size());
45     res[0] = 1;
46     for (; n; n /= 2, a = a * a) {
47         if (n & 1) {
48             res = res * a;
49         }
50     }
51     return res;
52 }
53 int main() {
54     ios::sync_with_stdio(false);
55     cin.tie(nullptr);
56     int n, k, m;
57     cin >> n >> k >> m;
58     Z::setMod(m);
59     Z ans = 0;
60     if (k % 2 == 1) {
61         vector<Z> a(k, 1);
62         a[0] = 0;

```

```

63         auto g = power(a, n);
64         for (int s = 0; s < k; s++) {
65             ans += g[(s - n % k * (1LL * s * (k + 1) / 2) % k + k) % k];
66         }
67     } else {
68         vector<Z> a(k, 1);
69         auto f = power(a, n);
70         a[0] = a[k / 2] = 0;
71         auto g = power(a, n);
72         for (int s = 0; s < k; s++) {
73             if (s % 2 == 1) {
74                 ans += f[s];
75             } else {
76                 ans += g[(s - n % k * (s / 2) % k + k) % k];
77             }
78         }
79     }
80     ans = power(Z(k), n) - ans;
81     cout << ans << "\n";
82     return 0;
83 }
```

## implementation

### 614: Sweets for Everyone!

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Christmas celebrations are coming to Whoville. Cindy Lou Who and her parents Lou Lou Who and Betty Lou Who decided to give sweets to all people in their street. They decided to give the residents of each house on the street, one kilogram of sweets. So they need as many kilos of sweets as there are homes on their street.

The street, where the Lou Who family lives can be represented as  $n$  consecutive sections of equal length. You can go from any section to a neighbouring one in one unit of time. Each of the sections is one of three types: an empty piece of land, a house or a shop. Cindy Lou and her family can buy sweets in a shop, but no more than one kilogram of sweets in one shop (the vendors care about the residents of Whoville not to overeat on sweets).

After the Lou Who family leave their home, they will be on the first section of the road. To get to this section of the road, they also require one unit of time. We can assume that Cindy and her mom and dad can carry an unlimited number of kilograms of sweets. Every time they are on a house section, they can give a kilogram of sweets to the inhabitants of the house, or they can simply move to another section. If the family have already given sweets to the residents of a house, they can't do it again. Similarly, if they are on the shop section, they can either buy a kilo of sweets in it or skip this shop. If they've

bought a kilo of sweets in a shop, the seller of the shop remembered them and the won't sell them a single candy if they come again. The time to buy and give sweets can be neglected. The Lou Whos do not want the people of any house to remain without food.

The Lou Whos want to spend no more than  $t$  time units of time to give out sweets, as they really want to have enough time to prepare for the Christmas celebration. In order to have time to give all the sweets, they may have to initially bring additional  $k$  kilos of sweets.

Cindy Lou wants to know the minimum number of  $k$  kilos of sweets they need to take with them, to have time to give sweets to the residents of each house in their street.

Your task is to write a program that will determine the minimum possible value of  $k$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, t;
8     cin >> n >> t;
9     string s;
10    cin >> s;
11    int cntH = count(s.begin(), s.end(), 'H');
12    int ri = n-1;
13    while (s[ri] != 'H') {
14        ri--;
15    }
16    if (t < ri+1) {
17        cout << -1 << "\n";
18        return 0;
19    }
20    vector<int> house, shop;
21    for (int i = 0; i < n; i++) {
22        if (s[i] == 'H') {
23            house.push_back(i);
24        }
25        if (s[i] == 'S') {
26            shop.push_back(i);
27        }
28    }
29    auto check = [&](int x) {
30        if (x + shop.size() < house.size()) {
31            return false;
32        }
33        int lst = -1;
34        int ans = t+1;
35        int res = 0;
36        int rightmost = max(house.back(), shop[house.size()-1-x]);
37        for (int i = x; i < house.size(); i++) {
38            ans = min(ans, res + house[i]+1 + 2 * (rightmost - house[i]));
39            if (house[i] < shop[i-x]) {
40                res += 2 * (shop[i-x] - max(lst, house[i]));
41                lst = shop[i-x];
42            }
43        }
44    }

```

```

43         }
44         return ans <= t;
45     };
46     int lo = 0, hi = cntH;
47     while (lo < hi) {
48         int x = (lo + hi) / 2;
49         if (check(x)) {
50             hi = x;
51         } else {
52             lo = x+1;
53         }
54     }
55     cout << lo << "\n";
56     return 0;
57 }

```

## 615: Bowls

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: input.txt
- Output file: output.txt

Once Petya was in such a good mood that he decided to help his mum with the washing-up. There were  $n$  dirty bowls in the sink. From the geometrical point of view each bowl looks like a blunted cone. We can disregard the width of the walls and bottom. Petya puts the clean bowls one on another naturally, i. e. so that their vertical axes coincide (see the picture). You will be given the order in which Petya washes the bowls. Determine the height of the construction, i.e. the distance from the bottom of the lowest bowl to the top of the highest one.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 #ifdef ONLINE_JUDGE
5     ifstream fin("input.txt");
6     ofstream fout("output.txt");
7 #else
8     #define fin cin
9     #define fout cout
10 #endif
11 int main() {
12     int n;
13     fin >> n;
14     double ans = 0;
15     vector<double> h(n), r(n), R(n);
16     for (int i = 0; i < n; i++) {
17         fin >> h[i] >> r[i] >> R[i];

```

```

18     }
19     vector<double> dp(n);
20     for (int i = 0; i < n; i++) {
21         for (int j = 0; j < i; j++) {
22             double x = 0;
23             if (r[i] >= R[j]) {
24                 x = h[j];
25             } else {
26                 x = max({0., (r[i] - r[j]) / (R[j] - r[j]) * h[j],
27                           min((R[i] - r[j]) / (R[j] - r[j]) * h[j] - h[i], h[j] - (R[j] - r[i])
28                           ] / (R[i] - r[i]) * h[i]}));
29             dp[i] = max(dp[i], dp[j] + x);
30         }
31         ans = max(ans, dp[i] + h[i]);
32     }
33     cout << fixed << setprecision(10);
34     cout << ans << "\n";
35     return 0;
36 }
```

## 616: How Many Squares?

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

You are given a 0-1 rectangular matrix. What is the number of squares in it? A square is a solid square frame (border) with linewidth equal to 1. A square should be at least 2x2. We are only interested in two types of squares:

squares with each side parallel to a side of the matrix;

squares with each side parallel to a diagonal of the matrix.

Regardless of type, a square must contain at least one 1 and can't touch (by side or corner) any foreign 1. Of course, the lengths of the sides of each square should be equal.

How many squares are in the given matrix?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int dx[] = {0, 0, -1, 1, -1, -1, 1, 1};
5 constexpr int dy[] = {-1, 1, 0, 0, -1, 1, -1, 1};
6 void solve() {
7     int n, m;
8     cin >> n >> m;
9     vector<string> s(n);
```

```

10     for (int i = 0; i < n; i++) {
11         cin >> s[i];
12     }
13     int ans = 0;
14     vector vis(n, vector<bool>(m));
15     for (int i = 0; i < n; i++) {
16         for (int j = 0; j < m; j++) {
17             if (!vis[i][j] && s[i][j] == '1') {
18                 queue<pair<int, int>> q;
19                 q.emplace(i, j);
20                 vis[i][j] = true;
21                 int minx = i, maxx = i;
22                 int miny = j, maxy = j;
23                 int cnt = 0;
24                 while (!q.empty()) {
25                     auto [x, y] = q.front();
26                     q.pop();
27                     minx = min(minx, x);
28                     maxx = max(maxx, x);
29                     miny = min(miny, y);
30                     maxy = max(maxy, y);
31                     cnt += 1;
32                     for (int k = 0; k < 8; k++) {
33                         int nx = x + dx[k];
34                         int ny = y + dy[k];
35                         if (0 <= nx && nx < n && 0 <= ny && ny < m && s[nx][ny] == '1'
36                                         && !vis[nx][ny]) {
37                             vis[nx][ny] = true;
38                             q.emplace(nx, ny);
39                         }
40                     }
41                 }
42                 if ([&]() {
43                     if (maxx - minx != maxy - miny) {
44                         return false;
45                     }
46                     if (maxx == minx) {
47                         return false;
48                     }
49                     if (cnt != 4 * (maxx - minx)) {
50                         return false;
51                     }
52                     for (int x = minx; x <= maxx; x++) {
53                         if (s[x][miny] != '1') {
54                             return false;
55                         }
56                         if (s[x][maxy] != '1') {
57                             return false;
58                         }
59                     }
60                     for (int y = miny; y <= maxy; y++) {
61                         if (s[minx][y] != '1') {
62                             return false;
63                         }
64                         if (s[maxx][y] != '1') {
65                             return false;
66                         }
67                     }
68                     return true;
69                 }()) {
70                     ans += 1;
71                 }
72                 if ([&]() {
73                     if (maxx - minx != maxy - miny) {

```

```

74         return false;
75     }
76     if (maxx == minx) {
77         return false;
78     }
79     if (((maxx - minx) % 2 != 0) {
80         return false;
81     }
82     if (cnt != 2 * (maxx - minx)) {
83         return false;
84     }
85     int d = (maxx - minx) / 2;
86     for (int i = 0; i <= d; i++) {
87         if (s[minx + i][miny + d - i] != '1') {
88             return false;
89         }
90         if (s[minx + i][maxy - d + i] != '1') {
91             return false;
92         }
93         if (s[maxx - i][miny + d - i] != '1') {
94             return false;
95         }
96         if (s[maxx - i][maxy - d + i] != '1') {
97             return false;
98         }
99     }
100    return true;
101 }() {
102     ans += 1;
103 }
104 }
105 }
106 cout << ans << "\n";
107 }
108 int main() {
109     ios::sync_with_stdio(false);
110     cin.tie(nullptr);
111     int t;
112     cin >> t;
113     while (t--) {
114         solve();
115     }
116     return 0;
117 }
```

### 617: Persistent Bookcase

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Recently in school Alina has learned what are the persistent data structures: they are data structures that always preserves the previous version of itself and access to it when it is modified.

After reaching home Alina decided to invent her own persistent data structure. Inventing didn't take

long: there is a bookcase right behind her bed. Alina thinks that the bookcase is a good choice for a persistent data structure. Initially the bookcase is empty, thus there is no book at any position at any shelf.

The bookcase consists of  $n$  shelves, and each shelf has exactly  $m$  positions for books at it. Alina enumerates shelves by integers from 1 to  $n$  and positions at shelves - from 1 to  $m$ . Initially the bookcase is empty, thus there is no book at any position at any shelf in it.

Alina wrote down  $q$  operations, which will be consecutively applied to the bookcase. Each of the operations has one of four types:

1  $i\ j$  - Place a book at position  $j$  at shelf  $i$  if there is no book at it.

2  $i\ j$  - Remove the book from position  $j$  at shelf  $i$  if there is a book at it.

3  $i$  - Invert book placing at shelf  $i$ . This means that from every position at shelf  $i$  which has a book at it, the book should be removed, and at every position at shelf  $i$  which has not book at it, a book should be placed.

4  $k$  - Return the books in the bookcase in a state they were after applying  $k$ -th operation. In particular,  $k = 0$  means that the bookcase should be in initial state, thus every book in the bookcase should be removed from its position.

After applying each of operation Alina is interested in the number of books in the bookcase. Alina got 'A' in the school and had no problem finding this values. Will you do so?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m, q;
8     cin >> n >> m >> q;
9     vector<vector<array<int, 4>>> adj(q + 1);
10    for (int i = 0; i < q; i++) {
11        int t;
12        cin >> t;
13        if (t == 1) {
14            int x, y;
15            cin >> x >> y;
16            x--, y--;
17            adj[i].push_back({t, i + 1, x, y});
18        } else if (t == 2) {
19            int x, y;
20            cin >> x >> y;
21            x--, y--;
22            adj[i].push_back({t, i + 1, x, y});
```

```

23         } else if (t == 3) {
24             int x;
25             cin >> x;
26             x--;
27             adj[i].push_back({t, i + 1, x});
28         } else {
29             int x;
30             cin >> x;
31             adj[x].push_back({t, i + 1});
32         }
33     }
34     vector<bitset<1000>> a(n);
35     bitset<1000> b {};
36     for (int i = 0; i < m; i++) {
37         b[i] = 1;
38     }
39     vector<int> ans(q + 1);
40     function<void(int)> dfs = [&](int i) {
41         for (auto [t, j, x, y] : adj[i]) {
42             if (t == 1) {
43                 int v = a[x][y];
44                 a[x][y] = 1;
45                 ans[j] = ans[i] + 1 - v;
46                 dfs(j);
47                 a[x][y] = v;
48             } else if (t == 2) {
49                 int v = a[x][y];
50                 a[x][y] = 0;
51                 ans[j] = ans[i] - v;
52                 dfs(j);
53                 a[x][y] = v;
54             } else if (t == 3) {
55                 int v = a[x].count();
56                 a[x] ^= b;
57                 ans[j] = ans[i] + m - 2 * v;
58                 dfs(j);
59                 a[x] ^= b;
60             } else {
61                 ans[j] = ans[i];
62                 dfs(j);
63             }
64         }
65     };
66     dfs(0);
67     for (int i = 1; i <= q; i++) {
68         cout << ans[i] << "\n";
69     }
70     return 0;
71 }

```

**618: Symmetree**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Kid was gifted a tree of  $n$  vertices with the root in the vertex 1. Since he really like symmetrical objects,

Kid wants to find out if this tree is symmetrical.

Formally, a tree is symmetrical if there exists an order of children such that:

The subtree of the leftmost child of the root is a mirror image of the subtree of the rightmost child;  
the subtree of the second-left child of the root is a mirror image of the subtree of the second-right child  
of the root;

...

if the number of children of the root is odd, then the subtree of the middle child should be symmetrical.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 vector<bool> sym;
5 map<vector<int>, int> tree;
6 void solve() {
7     int n;
8     cin >> n;
9     vector<vector<int>> adj(n);
10    for (int i = 1; i < n; i++) {
11        int u, v;
12        cin >> u >> v;
13        u--, v--;
14        adj[u].push_back(v);
15        adj[v].push_back(u);
16    }
17    auto dfs = [&](auto self, int x, int p) -> int {
18        vector<int> a;
19        for (auto y : adj[x]) {
20            if (y == p) {
21                continue;
22            }
23            a.push_back(self(self, y, x));
24        }
25        sort(a.begin(), a.end());
26        if (!tree.contains(a)) {
27            tree[a] = sym.size();
28            int i = 0;
29            while (i + 1 < a.size() && a[i] == a[i + 1]) {
30                i += 2;
31            }
32            bool ok = true;
33            if (i < a.size()) {
34                if (!sym[a[i]]) {
35                    ok = false;
36                }
37                i += 1;
38                while (i + 1 < a.size() && a[i] == a[i + 1]) {
39                    i += 2;
40                }
41            }
42        }
43    };
44    dfs(dfs, 0, -1);
45}

```

```

41             if (i != a.size()) {
42                 ok = false;
43             }
44         }
45         sym.push_back(ok);
46     }
47     return tree[a];
48 };
49 int x = dfs(dfs, 0, -1);
50 if (sym[x]) {
51     cout << "YES\n";
52 } else {
53     cout << "NO\n";
54 }
55 }
56 int main() {
57     ios::sync_with_stdio(false);
58     cin.tie(nullptr);
59     int t;
60     cin >> t;
61     while (t--) {
62         solve();
63     }
64     return 0;
65 }
```

## 619: Hospital Queue

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  people (numbered from 1 to  $n$ ) signed up for a doctor's appointment. The doctor has to choose in which order he will appoint these people. The  $i$ -th patient should be appointed among the first  $p_i$  people. There are also  $m$  restrictions of the following format: the  $i$ -th restriction is denoted by two integers  $(a_i, b_i)$  and means that the patient with the index  $a_i$  should be appointed earlier than the patient with the index  $b_i$ .

For example, if  $n = 4, p = [2, 3, 2, 4], m = 1, a = [3]$  and  $b = [1]$ , then the only order of appointment of patients that does not violate the restrictions is  $[3, 1, 2, 4]$ . For  $n = 3, p = [3, 3, 3], m = 0, a = []$  and  $b = []$ , any order of appointment is valid.

For each patient, calculate the minimum position in the order that they can have among all possible orderings that don't violate the restrictions.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
```

```
3  using i64 = long long;
4  int main() {
5      ios::sync_with_stdio(false);
6      cin.tie(nullptr);
7      int n, m;
8      cin >> n >> m;
9      vector<int> p(n);
10     for (int i = 0; i < n; i++) {
11         cin >> p[i];
12     }
13     vector<vector<int>> adj(n);
14     vector<int> deg(n);
15     vector<int> order;
16     for (int i = 0; i < m; i++) {
17         int u, v;
18         cin >> u >> v;
19         u--;
20         v--;
21         adj[u].push_back(v);
22         deg[v]++;
23     }
24     for (int i = 0; i < n; i++) {
25         if (deg[i] == 0) {
26             order.push_back(i);
27         }
28     }
29     for (int i = 0; i < n; i++) {
30         int x = order[i];
31         for (auto y : adj[x]) {
32             if (--deg[y] == 0) {
33                 order.push_back(y);
34             }
35         }
36     }
37     for (int i = n - 1; i >= 0; i--) {
38         int x = order[i];
39         for (auto y : adj[x]) {
40             p[x] = min(p[x], p[y]);
41         }
42     }
43     for (int x = 0; x < n; x++) {
44         vector<int> reach(n);
45         reach[x] = 1;
46         for (int i = n - 1; i >= 0; i--) {
47             int x = order[i];
48             for (auto y : adj[x]) {
49                 reach[x] |= reach[y];
50             }
51         }
52         vector<int> cnt(n + 1);
53         int reachable = 0;
54         for (int i = 0; i < n; i++) {
55             if (!reach[i]) {
56                 cnt[p[i]]++;
57             } else {
58                 reachable++;
59             }
60         }
61         for (int i = 1; i <= n; i++) {
62             cnt[i] += cnt[i - 1];
63         }
64         int ans = p[x];
65         while (cnt[ans - 1] + reachable <= ans - 1) {
66             ans--;
67         }
68     }
69 }
```

```

67         cout << ans << " \n"[x == n - 1];
68     }
69     return 0;
70 }
```

## 620: Monsters (hard version)

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. In this version, you need to find the answer for every prefix of the monster array.

In a computer game, you are fighting against  $n$  monsters. Monster number  $i$  has  $a_i$  health points, all  $a_i$  are integers. A monster is alive while it has at least 1 health point.

You can cast spells of two types:

Deal 1 damage to any single alive monster of your choice.

Deal 1 damage to all alive monsters. If at least one monster dies (ends up with 0 health points) as a result of this action, then repeat it (and keep repeating while at least one monster dies every time).

Dealing 1 damage to a monster reduces its health by 1.

Spells of type 1 can be cast any number of times, while a spell of type 2 can be cast at most once during the game.

For every  $k = 1, 2, \dots, n$ , answer the following question. Suppose that only the first  $k$  monsters, with numbers  $1, 2, \dots, k$ , are present in the game. What is the smallest number of times you need to cast spells of type 1 to kill all  $k$  monsters?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int inf = 1E9;
5 void solve() {
6     int n;
7     cin >> n;
8     vector<int> a(n);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
```

```
11     }
12     auto v = a;
13     sort(v.begin(), v.end());
14     int last = 0;
15     for (int i = 0; i < n; i++) {
16         v[i] = min(v[i], last + 1);
17         last = v[i];
18     }
19     multiset<pair<int, int>> s;
20     i64 res = 0;
21     for (int i = 0; i < n; i++) {
22         s.emplace(v[i], v[i]);
23         res += a[i] - v[i];
24     }
25     s.emplace(0, 0);
26     vector<i64> ans(n);
27     for (int i = n - 1; i >= 0; i--) {
28         ans[i] = res;
29         res -= a[i];
30         auto it = prev(s.upper_bound({a[i], inf}));
31         auto [l, r] = *it;
32         s.erase(it);
33         if (a[i] >= r) {
34             res += r;
35             r--;
36             if (l <= r) {
37                 s.emplace(l, r);
38             }
39         } else {
40             res += a[i];
41             if (a[i] < r) {
42                 s.emplace(a[i] + 1, r);
43             }
44             if (l <= a[i] - 1) {
45                 s.emplace(l, a[i] - 1);
46             }
47         }
48         it = s.lower_bound({a[i], 0});
49         // for (auto [l, r] : s) {
50         //     cerr << "(" << l << "," << r << ")";
51         // }
52         // cerr << "\n";
53         while (it != s.end() && it->first > prev(it)->second + 1) {
54             auto l = prev(it);
55             auto [x, y] = *l;
56             auto [a, b] = *it;
57             res += b - a + 1;
58             s.erase(l);
59             it = s.erase(it);
60             s.emplace(x, y + 1 + b - a);
61         }
62     }
63     for (int i = 0; i < n; i++) {
64         cout << ans[i] << " \n"[i == n - 1];
65     }
66 }
67 int main() {
68     ios::sync_with_stdio(false);
69     cin.tie(nullptr);
70     int t;
71     cin >> t;
72     while (t--) {
73         solve();
74     }
```

```

75     return 0;
76 }
```

## math

### 621: Evaluate It and Back Again

- Time limit: 3 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

Aidan and Nadia are long-time friends with a shared passion for mathematics. Each of them has a favorite number: Aidan's favorite number is  $p$ , and Nadia's is  $q$ .

To commemorate their friendship, their friends want to make a present: a plaque with an arithmetic expression whose value is equal to their favorite numbers. At first glance, it sounds impossible, but the answer is simple: Aidan reads left-to-right, while Nadia reads right-to-left, so the same expression can have different values for them.

For example, if 2023-12-13 is written on the plaque, then Aidan would calculate the result as  $2023 - 12 - 13 = 1998$ , and Nadia would calculate it as  $31 - 21 - 3202 = -3192$ .

Find an arithmetic expression that, when read left-to-right, evaluates to  $p$ , and, when read right-to-left, evaluates to  $q$ . Its length must be at most 1000 characters. It's guaranteed that such an expression exists for all valid inputs.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     i64 p, q;
8     cin >> p >> q;
9     i64 d = p - q;
10    string ans = "0";
11    if (d % 9 != 0) {
12        int x = (d % 9 + 9) % 9;
13        x = x * 5 % 9;
14        d -= 2 * x;
15        p -= x;
16        ans += "+";
17        ans += to_string(x);
```

```

18         ans += "-0";
19     }
20     while (d != 0) {
21         i64 x = 9;
22         while (x < 999999999999999999LL && x * 10 + 9 <= abs(d)) {
23             x = x * 10 + 9;
24         }
25         int t = abs(d) / x;
26         if (t > 8) {
27             t = 8;
28         }
29         if (d > 0) {
30             d -= t * x;
31             ans += "+";
32             ans += to_string(x + t + 2 + t * x);
33             p -= x + t + 2 + t * x;
34         } else {
35             d += t * x;
36             ans += "+";
37             ans += to_string(x + t + 2);
38             p -= x + t + 2;
39         }
40     }
41     while (p < 0) {
42         i64 x = 1;
43         while (x < 111111111111111111111111LL && x < -p) {
44             x = x * 10 + 1;
45         }
46         p += x;
47         ans += "+0-";
48         ans += to_string(x);
49         ans += "-0";
50     }
51     while (p > 0) {
52         i64 x = 1;
53         while (x < 111111111111111111111111LL && x * 10 + 1 <= p) {
54             x = x * 10 + 1;
55         }
56         int t = p / x;
57         if (t > 9) {
58             t = 9;
59         }
60         ans += "+";
61         ans += to_string(t * x);
62         p -= t * x;
63     }
64     cout << ans << "\n";
65     return 0;
66 }

```

**622: Hemose in ICPC ?**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an interactive problem!

In the last regional contest Hemose, ZeyadKhattab and YahiaSherif - members of the team Carpe Diem - did not qualify to ICPC because of some unknown reasons. Hemose was very sad and had a bad day after the contest, but ZeyadKhattab is very wise and knows Hemose very well, and does not want to see him sad.

Zeyad knows that Hemose loves tree problems, so he gave him a tree problem with a very special device.

Hemose has a weighted tree with  $n$  nodes and  $n - 1$  edges. Unfortunately, Hemose doesn't remember the weights of edges.

Let's define  $Dist(u, v)$  for  $u \neq v$  as the greatest common divisor of the weights of all edges on the path from node  $u$  to node  $v$ .

Hemose has a special device. Hemose can give the device a set of nodes, and the device will return the largest  $Dist$  between any two nodes from the set. More formally, if Hemose gives the device a set  $S$  of nodes, the device will return the largest value of  $Dist(u, v)$  over all pairs  $(u, v)$  with  $u, v \in S$  and  $u \neq v$ .

Hemose can use this Device at most 12 times, and wants to find any two distinct nodes  $a, b$ , such that  $Dist(a, b)$  is maximum possible. Can you help him?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int query(vector<int> a) {
5     cout << "? " << a.size();
6     for (auto x : a) {
7         cout << " " << x + 1;
8     }
9     cout << endl;
10    int ans;
11    cin >> ans;
12    return ans;
13 }
14 int main() {
15     ios::sync_with_stdio(false);
16     cin.tie(nullptr);
17     int n;
18     cin >> n;
19     vector<vector<pair<int, int>>> adj(n);
20     vector<int> u(n - 1), v(n - 1);
21     for (int i = 0; i < n - 1; i++) {
22         cin >> u[i] >> v[i];
23         u[i]--, v[i]--;
24         adj[u[i]].emplace_back(v[i], i);
25         adj[v[i]].emplace_back(u[i], i);
26     }
27     vector<int> f(n - 1);
28     iota(f.begin(), f.end(), 0);
29     auto ver = [&](auto f) {

```

```

30         vector<int> deg(n);
31         for (auto i : f) {
32             deg[u[i]] += 1;
33             deg[v[i]] += 1;
34         }
35         vector<int> a;
36         for (int i = 0; i < n; i++) {
37             if (deg[i] > 0) {
38                 a.push_back(i);
39             }
40         }
41         return a;
42     };
43     int max = query(ver(f));
44     while (f.size() > 1) {
45         vector<int> deg(n);
46         vector<bool> inf(n - 1);
47         for (auto i : f) {
48             inf[i] = true;
49             deg[u[i]] += 1;
50             deg[v[i]] += 1;
51         }
52         vector<bool> vis(n);
53         vector<int> e;
54         for (int i = 0; i < n; i++) {
55             if (!vis[i] && deg[i] > 0) {
56                 queue<int> q;
57                 vis[i] = true;
58                 q.push(i);
59                 while (!q.empty()) {
60                     int x = q.front();
61                     q.pop();
62                     for (auto [y, j] : adj[x]) {
63                         if (inf[j] && !vis[y]) {
64                             e.push_back(j);
65                             vis[y] = true;
66                             q.push(y);
67                         }
68                     }
69                 }
70             }
71         }
72         int cnt = f.size() / 2;
73         if (query(ver(vector(e.begin(), e.begin() + cnt))) == max) {
74             f = vector(e.begin(), e.begin() + cnt);
75         } else {
76             f = vector(e.begin() + cnt, e.end());
77         }
78     }
79     cout << "!" << u[f[0]] + 1 << " " << v[f[0]] + 1 << endl;
80     return 0;
81 }

```

### 623: Vika and Bonuses

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A new bonus system has been introduced at Vika's favorite cosmetics store, "Golden Pear"!

The system works as follows: suppose a customer has  $b$  bonuses. Before paying for the purchase, the customer can choose one of two options:

Get a discount equal to the current number of bonuses, while the bonuses are not deducted.

Accumulate an additional  $x$  bonuses, where  $x$  is the last digit of the number  $b$ . As a result, the customer's account will have  $b + x$  bonuses.

For example, if a customer had 24 bonuses, he can either get a discount of 24 or accumulate an additional 4 bonuses, after which his account will have 28 bonuses.

At the moment, Vika has already accumulated  $s$  bonuses.

The girl knows that during the remaining time of the bonus system, she will make  $k$  more purchases at the "Golden Pear" store network.

After familiarizing herself with the rules of the bonus system, Vika became interested in the maximum total discount she can get.

Help the girl answer this question.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     i64 s, k;
6     cin >> s >> k;
7     i64 ans = s * k;
8     if (s % 2 == 1) {
9         s += s % 10;
10        k--;
11        ans = max(ans, s * k);
12    }
13    if (s % 10 == 0) {
14        cout << ans << "\n";
15        return;
16    }
17    for (int t = 0; t < 4 && t <= k; t++) {
18        i64 lo = 0, hi = (k - t) / 4;
19        auto get = [&](i64 n) {
20            i64 x = s;
21            for (int i = 0; i < t; i++) {
22                x += x % 10;
23            }
24            x += 20 * n;
25            return x * (k - 4 * n - t);
26        };
27        while (lo < hi) {
28            i64 x = (lo + hi) / 2;

```

```

29         if (get(x) < get(x + 1)) {
30             lo = x + 1;
31         } else {
32             hi = x;
33         }
34     }
35     ans = max(ans, get(lo));
36 }
37 cout << ans << "\n";
38 }
39 int main() {
40     ios::sync_with_stdio(false);
41     cin.tie(nullptr);
42     int t;
43     cin >> t;
44     while (t--) {
45         solve();
46     }
47     return 0;
48 }
```

## 624: In Search of Truth (Easy Version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The only difference between easy and hard versions is the maximum number of queries. In this version, you are allowed to ask at most 2023 queries.

This is an interactive problem.

You are playing a game. The circle is divided into  $n$  sectors, sectors are numbered from 1 to  $n$  in some order. You are in the adjacent room and do not know either the number of sectors or their numbers. There is also an arrow that initially points to some sector. Initially, the host tells you the number of the sector to which the arrow points. After that, you can ask the host to move the arrow  $k$  sectors counterclockwise or clockwise at most 2023 times. And each time you are told the number of the sector to which the arrow points.

Your task is to determine the integer  $n$  - the number of sectors in at most 2023 queries.

It is guaranteed that  $1 \leq n \leq 10^6$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
```

```

2  using namespace std;
3  using i64 = long long;
4  constexpr int B = 400;
5  int main() {
6      ios::sync_with_stdio(false);
7      cin.tie(nullptr);
8      int x;
9      cin >> x;
10     int N = x;
11     mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
12     for (int i = 0; i < B; i++) {
13         cout << "+" << rng() % 1000000000 + 1 << endl;
14         cin >> x;
15         N = max(N, x);
16     }
17     int C = (1000 - B) / 2;
18     map<int, int> f;
19     f[x] = 0;
20     for (int i = 1; i < C; i++) {
21         cout << "-" << i << endl;
22         cin >> x;
23         if (!f.count(x)) {
24             f[x] = i;
25         }
26     }
27     cout << "+" << C - 1 + N << endl;
28     cin >> x;
29     int n = N;
30     while (!f.count(x)) {
31         cout << "+" << C << endl;
32         cin >> x;
33         n += C;
34     }
35     n += f[x];
36     cout << "!" << n << endl;
37     return 0;
38 }
```

## 625: LuoTianyi and the Floating Islands (Hard Version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. The only difference is that in this version  $k \leq n$ . You can make hacks only if both versions of the problem are solved.

LuoTianyi now lives in a world with  $n$  floating islands. The floating islands are connected by  $n - 1$  undirected air routes, and any two of them can reach each other by passing the routes. That means, the  $n$  floating islands form a tree.

One day, LuoTianyi wants to meet her friends: Chtholly, Nephren, William, .... Totally, she wants to meet  $k$  people. She doesn't know the exact positions of them, but she knows that they are in pairwise

distinct islands. She define an island is good if and only if the sum of the distances<sup>†</sup> from it to the islands with  $k$  people is the minimal among all the  $n$  islands.

Now, LuoTianyi wants to know that, if the  $k$  people are randomly set in  $k$  distinct of the  $n$  islands, then what is the expect number of the good islands? You just need to tell her the expect number modulo  $10^9 + 7$ .

<sup>†</sup>The distance between two islands is the minimum number of air routes you need to take to get from one island to the other.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int n, k;
39     cin >> n >> k;
40     vector<vector<int>> adj(n);
41     for (int i = 1; i < n; i++) {
42         int u, v;
43         cin >> u >> v;
44         u--, v--;
45         adj[u].push_back(v);
46         adj[v].push_back(u);

```

```

47      }
48      Z ans = 1;
49      if (k % 2 == 0) {
50          Z inv = 1 / comb.binom(n, k);
51          vector<int> siz(n);
52          function<void(int, int)> dfs = [&](int x, int p) {
53              siz[x] = 1;
54              for (auto y : adj[x]) {
55                  if (y == p) {
56                      continue;
57                  }
58                  dfs(y, x);
59                  siz[x] += siz[y];
60                  ans += comb.binom(siz[y], k / 2) * comb.binom(n - siz[y], k / 2) * inv;
61              }
62          };
63          dfs(0, -1);
64      }
65      cout << ans << "\n";
66      return 0;
67  }

```

## 626: Strange town

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Volodya has recently visited a very odd town. There are N tourist attractions in the town and every two of them are connected by a bidirectional road. Each road has some travel price (natural number) assigned to it and all prices are distinct. But the most striking thing about this town is that each city sightseeing tour has the same total price! That is, if we choose any city sightseeing tour - a cycle which visits every attraction exactly once - the sum of the costs of the tour roads is independent of the tour. Volodya is curious if you can find such price system with all road prices not greater than 1000.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    a[0] = 0;
11    a[1] = 1;

```

```

12     set<int> s{0, 1};
13     for (int i = 2, x = 0; i < n; i++) {
14         while (s.count(x)) {
15             x++;
16         }
17         a[i] = x;
18         for (int j = 0; j <= i; j++) {
19             for (int k = 0; k <= i; k++) {
20                 if (j < i) {
21                     s.insert(a[i] + a[j] - a[k]);
22                 }
23                 if (j < k) {
24                     s.insert(a[j] + a[k] - a[i]);
25                 }
26             }
27         }
28     }
29     for (int i = 0; i < n; i++) {
30         for (int j = 0; j < n; j++) {
31             cout << (i == j ? 0 : a[i] + a[j]) << " \n"[j == n-1];
32         }
33     }
34 }
35 }
```

## 627: Magic Triples (Hard Version)

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. The only difference is that in this version,  $a_i \leq 10^9$ .

For a given sequence of  $n$  integers  $a$ , a triple  $(i, j, k)$  is called magic if:

$1 \leq i, j, k \leq n$ .

$i, j, k$  are pairwise distinct.

there exists a positive integer  $b$  such that  $a_i \cdot b = a_j$  and  $a_j \cdot b = a_k$ .

Kolya received a sequence of integers  $a$  as a gift and now wants to count the number of magic triples for it. Help him with this task!

Note that there are no constraints on the order of integers  $i, j$  and  $k$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 vector<int> minp, primes;
5 void sieve(int n) {
6     minp.assign(n + 1, 0);
7     primes.clear();
8     for (int i = 2; i <= n; i++) {
9         if (minp[i] == 0) {
10             minp[i] = i;
11             primes.push_back(i);
12         }
13         for (auto p : primes) {
14             if (i * p > n) {
15                 break;
16             }
17             minp[i * p] = p;
18             if (p == minp[i]) {
19                 break;
20             }
21         }
22     }
23 }
24 vector<vector<int>> divs;
25 void solve() {
26     int n;
27     cin >> n;
28     vector<int> a(n);
29     for (int i = 0; i < n; i++) {
30         cin >> a[i];
31     }
32     sort(a.begin(), a.end());
33     i64 ans = 0;
34     for (int i = 0; i < n; i++) {
35         int x = a[i];
36         int s = 1;
37         for (auto v : primes) {
38             if (v * v * v > x) {
39                 break;
40             }
41             int t = 0;
42             while (x % v == 0) {
43                 x /= v;
44                 t++;
45                 if (t % 2 == 0) {
46                     s *= v;
47                 }
48             }
49         }
50         if (x > 1) {
51             int u = sqrt(x);
52             if (u * u == x) {
53                 s *= u;
54             }
55         }
56         for (auto b : divs[s]) {
57             auto [l1, r1] = equal_range(a.begin(), a.end(), a[i] / b);
58             auto [l2, r2] = equal_range(a.begin(), a.end(), a[i] / (b*b));
59             ans += 1LL * (r1-l1) * (r2-l2);
60         }
61     }
62     for (int i = 0, j = 0; i < n; i = j) {
63         while (j < n && a[i] == a[j]) {
64             j++;
65         }
66     }
67 }
```

```

65         }
66         ans += 1LL * (j-i) * (j-i-1) * (j-i-2);
67     }
68     cout << ans << "\n";
69 }
70 int main() {
71     ios::sync_with_stdio(false);
72     cin.tie(nullptr);
73     int n = 4E4;
74     sieve(n);
75     divs.resize(n+1);
76     for (int i = 2; i <= n; i++) {
77         for (int j = i; j <= n; j += i) {
78             divs[j].push_back(i);
79         }
80     }
81     int t;
82     cin >> t;
83     while (t--) {
84         solve();
85     }
86     return 0;
87 }
```

## 628: Square Root of Permutation

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A permutation of length  $n$  is an array containing each integer from 1 to  $n$  exactly once. For example,  $q = [4, 5, 1, 2, 3]$  is a permutation. For the permutation  $q$  the square of permutation is the permutation  $p$  that  $p[i] = q[q[i]]$  for each  $i = 1 \dots n$ . For example, the square of  $q = [4, 5, 1, 2, 3]$  is  $p = q^2 = [2, 3, 4, 5, 1]$ .

This problem is about the inverse operation: given the permutation  $p$  you task is to find such permutation  $q$  that  $q^2 = p$ . If there are several such  $q$  find any of them.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> p(n);
10    for (int i = 0; i < n; i++) {
11        cin >> p[i];
12        p[i]--;
13    }
```

```

14     vector<vector<int>> f;
15     vector<bool> vis(n);
16     vector<int> q(n);
17     for (int i = 0; i < n; i++) {
18         if (!vis[i]) {
19             int j = i;
20             vector<int> a;
21             while (!vis[j]) {
22                 vis[j] = true;
23                 a.push_back(j);
24                 j = p[j];
25             }
26             if (a.size() % 2 == 1) {
27                 int t = (a.size() + 1) / 2;
28                 for (int j = 0; j < a.size(); j++) {
29                     q[a[j]] = a[(j + t) % a.size()];
30                 }
31             } else {
32                 f.push_back(a);
33             }
34         }
35     }
36     sort(f.begin(), f.end(), [&](auto a, auto b) {
37         return a.size() < b.size();
38     });
39     if (f.size() % 2) {
40         cout << -1 << "\n";
41         return 0;
42     }
43     for (int i = 0; i < f.size(); i += 2) {
44         if (f[i].size() != f[i+1].size()) {
45             cout << -1 << "\n";
46             return 0;
47         }
48         for (int j = 0; j < f[i].size(); j++) {
49             q[f[i][j]] = f[i+1][j];
50             q[f[i+1][j]] = f[i][(j+1) % f[i].size()];
51         }
52     }
53     for (int i = 0; i < n; i++) {
54         cout << q[i]+1 << " \n"[i == n-1];
55     }
56     return 0;
57 }
```

## 629: Berland Square

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Last year the world's largest square was built in Berland. It is known that the square can be represented as an infinite plane with an introduced Cartesian system of coordinates. On that square two sets of concentric circles were painted. Let's call the set of concentric circles with radii  $1, 2, \dots, K$  and the center in the point  $(z, 0)$  a  $(K, z)$ -set. Thus, on the square were painted a  $(N, x)$ -set and a  $(M, y)$ -set. You

have to find out how many parts those sets divided the square into.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int N, x, M, y;
8     cin >> N >> x >> M >> y;
9     int d = abs(x - y);
10    if (d > N + M) {
11        cout << N + M + 1 << "\n";
12        return 0;
13    }
14    i64 ans = 2 + N + M;
15    for (int i = 1; i <= N; i++) {
16        ans -= (abs(i - min(M, i)) <= d && d <= i + M);
17    }
18    for (int i = 1; i <= M; i++) {
19        ans -= (abs(i - min(N, i)) <= d && d <= i + N);
20    }
21    for (int i = 1; i <= N; i++) {
22        int lo = max({1, d - i, i - d});
23        int hi = min({M, i + d});
24        ans += max(0, hi - lo + 1);
25        lo = max({1, d + 1 - i, i - d + 1});
26        hi = min({M, i + d - 1});
27        ans += max(0, hi - lo + 1);
28    }
29    cout << ans << "\n";
30    return 0;
31 }
```

## 630: Li Hua and Array

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Li Hua wants to solve a problem about  $\varphi$  - Euler's totient function. Please recall that  $\varphi(x) = \sum_{i=1}^x [\gcd(i, x) = 1]$ .<sup>†,‡</sup>

He has a sequence  $a_1, a_2, \dots, a_n$  and he wants to perform  $m$  operations:

“1  $l$   $r$ ” ( $1 \leq l \leq r \leq n$ ) - for each  $x \in [l, r]$ , change  $a_x$  into  $\varphi(a_x)$ .

“2  $l$   $r$ ” ( $1 \leq l \leq r \leq n$ ) - find out the minimum changes needed to make sure  $a_l = a_{l+1} = \dots = a_r$ . In each change, he chooses one  $x \in [l, r]$ , change  $a_x$  into  $\varphi(a_x)$ . Each operation of this type is

independent, which means the array doesn't actually change.

Suppose you were Li Hua, please solve this problem.

<sup>†</sup>  $\gcd(x, y)$  denotes the greatest common divisor (GCD) of integers  $x$  and  $y$ .

<sup>‡</sup> The notation  $[\text{cond}]$  equals 1 if the condition cond is true, and 0 otherwise.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 vector<int> minp, primes, phi;
5 void sieve(int n) {
6     minp.assign(n + 1, 0);
7     phi.assign(n + 1, 0);
8     primes.clear();
9     phi[1] = 1;
10    for (int i = 2; i <= n; i++) {
11        if (minp[i] == 0) {
12            minp[i] = i;
13            phi[i] = i - 1;
14            primes.push_back(i);
15        }
16        for (auto p : primes) {
17            if (i * p > n) {
18                break;
19            }
20            minp[i * p] = p;
21            if (p == minp[i]) {
22                phi[i * p] = phi[i] * p;
23                break;
24            }
25            phi[i * p] = phi[i] * (p - 1);
26        }
27    }
28 }
29 template<class Info,
30         class Merge = plus<Info>>
31 # struct SegmentTree;
32 # struct Info;
33 Info operator+(Info a, Info b) {
34     if (!a.a) {
35         return b;
36     }
37     if (!b.a) {
38         return a;
39     }
40     Info c;
41     while (a.a != b.a) {
42         if (a.a > b.a) {
43             a.sum += a.cnt;
44             a.a = phi[a.a];
45         } else {
46             b.sum += b.cnt;
47             b.a = phi[b.a];
48         }
49     }
50     c.a = a.a;
51     c.cnt = a.cnt + b.cnt;

```

```

52     c.sum = a.sum + b.sum;
53     return c;
54 }
55 # struct DSU;
56 int main() {
57     ios::sync_with_stdio(false);
58     cin.tie(nullptr);
59     sieve(5E6);
60     int n, m;
61     cin >> n >> m;
62     vector<int> a(n);
63     for (int i = 0; i < n; i++) {
64         cin >> a[i];
65     }
66     SegmentTree seg(vector<Info>(a.begin(), a.end()));
67     DSU dsu(n + 1);
68     while (m--) {
69         int t, l, r;
70         cin >> t >> l >> r;
71         l--;
72         if (t == 1) {
73             for (int i = dsu.leader(l); i < r; i = dsu.leader(i + 1)) {
74                 if (a[i] == 1) {
75                     dsu.merge(i + 1, i);
76                 } else {
77                     seg.change(i, a[i]);
78                     a[i] = phi[a[i]];
79                 }
80             }
81         } else {
82             auto info = seg.rangeQuery(l, r);
83             cout << info.sum << "\n";
84         }
85     }
86     return 0;
87 }
```

### 631: Memory and Scores

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Memory and his friend Lexa are competing to get higher score in one popular computer game. Memory starts with score  $a$  and Lexa starts with score  $b$ . In a single turn, both Memory and Lexa get some integer in the range  $[-k; k]$  (i.e. one integer among  $-k, -k+1, -k+2, \dots, -2, -1, 0, 1, 2, \dots, k-1, k$ ) and add them to their current scores. The game has exactly  $t$  turns. Memory and Lexa, however, are not good at this game, so they both always get a random integer at their turn.

Memory wonders how many possible games exist such that he ends with a strictly higher score than Lexa. Two games are considered to be different if in at least one turn at least one player gets different score. There are  $(2k + 1)2t$  games in total. Since the answer can be very large, you should print it

modulo 109 + 7. Please solve this problem for Memory.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int a, b, k, t;
38     cin >> a >> b >> k >> t;
39     k *= 2;
40     vector<Z> dp(k * t + 1);
41     dp[0] = 1;
42     for (int i = 0; i < t; i++) {
43         for (int j = 1; j <= k * t; j++) {
44             dp[j] += dp[j - 1];
45         }
46         for (int j = k * t; j >= k + 1; j--) {
47             dp[j] -= dp[j - k - 1];
48         }
49     }
50     Z ans = 0;
51     Z sum = 0;
52     for (int i = 0, j = -1; i <= k * t; i++) {
53         while (j + 1 <= k * t && a + i > b + j + 1) {
54             j++;
55             sum += dp[j];
56         }
57         ans += dp[i] * sum;
58     }

```

```

59     cout << ans << "\n";
60     return 0;
61 }
```

## 632: ZS and The Birthday Paradox

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

ZS the Coder has recently found an interesting concept called the Birthday Paradox. It states that given a random set of 23 people, there is around 50% chance that some two of them share the same birthday. ZS the Coder finds this very interesting, and decides to test this with the inhabitants of Udayland.

In Udayland, there are  $2n$  days in a year. ZS the Coder wants to interview  $k$  people from Udayland, each of them has birthday in one of  $2n$  days (each day with equal probability). He is interested in the probability of at least two of them have the birthday at the same day.

ZS the Coder knows that the answer can be written as an irreducible fraction . He wants to find the values of A and B (he does not like to deal with floating point numbers). Can you help him?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
```

```

28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000003;
33 using Z = MInt<P>;
34 using ZE = MInt<P - 1>;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     i64 n, k;
39     cin >> n >> k;
40     if (n <= 60 && k > (1LL << n)) {
41         cout << 1 << " " << 1 << "\n";
42         return 0;
43     }
44     auto e = ZE(n) * (k - 1);
45     ZE ek = 0;
46     i64 x = k - 1;
47     while (x > 0) {
48         ek += x / 2;
49         x /= 2;
50     }
51     Z den = power(Z(2), int(e - ek));
52     Z num = 1;
53     if (k > P) {
54         num = 0;
55     } else {
56         Z pn = power(Z(2), n);
57         for (int i = 1; i < k; i++) {
58             num *= pn - i;
59         }
60         num /= power(Z(2), int(ek));
61     }
62     num = den - num;
63     cout << num << " " << den << "\n";
64     return 0;
65 }
```

### 633: City Union

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given  $n \times m$  grid. Some cells are filled and some are empty.

A city is a maximal (by inclusion) set of filled cells such that it is possible to get from any cell in the set to any other cell in the set by moving to adjacent (by side) cells, without moving into any cells not in the set. In other words, a city is a connected component of filled cells with edges between adjacent (by side) cells.

Initially, there are two cities on the grid. You want to change some empty cells into filled cells so that both of the following are satisfied:

There is one city on the resulting grid.

The shortest path between any two filled cells, achievable only by moving onto filled cells, is equal to the Manhattan distance between them.

The Manhattan distance between two cells  $(a, b)$  and  $(c, d)$  is equal to  $|a - c| + |b - d|$ .

Find a way to add filled cells that satisfies these conditions and minimizes the total number of filled cells.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<string> s(n);
8     for (int i = 0; i < n; i++) {
9         cin >> s[i];
10    }
11    vector<int> minr(n, m), maxr(n, -1), minc(m, n), maxc(m, -1);
12    for (int i = 0; i < n; i++) {
13        for (int j = 0; j < m; j++) {
14            if (s[i][j] == '#') {
15                minr[i] = min(minr[i], j);
16                maxr[i] = max(maxr[i], j);
17                minc[j] = min(minc[j], i);
18                maxc[j] = max(maxc[j], i);
19            }
20        }
21    }
22    function<void(int, int)> work = [&](int x, int y) -> void {
23        s[x][y] = '#';
24        minr[x] = min(minr[x], y);
25        maxr[x] = max(maxr[x], y);
26        minc[y] = min(minc[y], x);
27        maxc[y] = max(maxc[y], x);
28        for (int j = minr[x]; j <= maxr[x]; j++) {
29            if (s[x][j] == '.') {
30                work(x, j);
31            }
32        }
33        for (int i = minc[y]; i <= maxc[y]; i++) {
34            if (s[i][y] == '.') {
35                work(i, y);
36            }
37        }
38    };
39    for (int i = 0; i < n; i++) {
40        for (int j = minr[i]; j <= maxr[i]; j++) {
41            if (s[i][j] == '.') {
42                work(i, j);
43            }
44        }
45    }
46    for (int j = 0; j < m; j++) {
47        for (int i = minc[j]; i <= maxc[j]; i++) {

```

```

48         if (s[i][j] == '.') {
49             work(i, j);
50         }
51     }
52 }
53 int u1 = 0;
54 while (minr[u1] > maxr[u1]) {
55     u1++;
56 }
57 int d1 = u1;
58 while (d1 + 1 < n && minr[d1 + 1] <= maxr[d1] && maxr[d1 + 1] >= minr[d1]) {
59     d1++;
60 }
61 int u2 = d1 + 1;
62 while (u2 < n && minr[u2] > maxr[u2]) {
63     u2++;
64 }
65 if (u2 < n) {
66     int d2 = u2;
67     while (d2 + 1 < n && minr[d2 + 1] <= maxr[d2] && maxr[d2 + 1] >= minr[d2]) {
68         d2++;
69     }
70     int r1 = -1, l2 = m;
71     int r2 = -1, l1 = m;
72     for (int i = u1; i <= d1; i++) {
73         r1 = max(r1, maxr[i]);
74         l1 = min(l1, minr[i]);
75     }
76     for (int i = u2; i <= d2; i++) {
77         r2 = max(r2, maxr[i]);
78         l2 = min(l2, minr[i]);
79     }
80     if (r1 < l2) {
81         work(d1, r1);
82         work(u2, l2);
83         for (int i = d1 + 1; i <= u2; i++) {
84             work(i, r1);
85         }
86     } else {
87         work(d1, l1);
88         work(u2, r2);
89         for (int i = d1 + 1; i <= u2; i++) {
90             work(i, l1);
91         }
92     }
93 }
94 for (int i = 0; i < n; i++) {
95     cout << s[i] << "\n";
96 }
97 cout << "\n";
98 }
99 int main() {
100     ios::sync_with_stdio(false);
101     cin.tie(nullptr);
102     int t;
103     cin >> t;
104     while (t--) {
105         solve();
106     }
107     return 0;
108 }
```

### 634: Explosions?

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are playing yet another game where you kill monsters using magic spells. There are  $n$  cells in the row, numbered from 1 to  $n$ . Initially, the  $i$ -th cell contains the  $i$ -th monster with  $h_i$  health.

You have a basic spell that costs 1 MP and deals 1 damage to the monster you choose. You can cast it any number of times. Also, you have a special scroll with “Explosion” spell you can use only once. You want to finish killing monsters with explosion, that’s why you, firstly, cast the basic spell several times (possibly, zero), and then after that, you cast one “Explosion”.

How does “Explosion” spell work? Firstly, you choose the power of the spell: if you pour  $x$  MP into it, “Explosion” will deal  $x$  damage. Secondly, you choose some monster  $i$ , which will be targeted by the spell. That’s what happens next:

if its current health  $h_i > x$ , then he stays alive with health decreased by  $x$ ;

if  $h_i \leq x$ , the  $i$ -th monster dies with an explosion that deals  $h_i - 1$  damage to monsters in the neighboring cells  $i - 1$  and  $i + 1$ , if these cells exist and monsters inside are still alive;

if the damage dealt by the explosion is enough to kill the monster  $i - 1$  (or  $i + 1$ ), i. e. the current  $h_{i-1} \leq h_i - 1$  (or  $h_{i+1} \leq h_i - 1$ ), then that monster also dies creating a secondary explosion of power  $h_{i-1} - 1$  (or  $h_{i+1} - 1$ ) that may deals damage to their neighbors, and so on, until the explosions end.

Your goal is to kill all the remaining monsters with those “chaining” explosions, that’s why you need a basic spell to decrease  $h_i$  of some monsters or even kill them beforehand (monsters die when their current health  $h_i$  becomes less or equal to zero). Note that monsters don’t move between cells, so, for example, monsters  $i$  and  $i + 2$  will never become neighbors.

What is the minimum total MP you need to kill all monsters in the way you want? The total MP is counted as the sum of the number of basic spells you cast and the power  $x$  of explosion scroll you’ve chosen.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;

```

```

7     vector<int> h(n);
8     for (int i = 0; i < n; i++) {
9         cin >> h[i];
10    }
11    i64 ans = 1E18;
12    auto work = [&](auto h) {
13        vector<i64> s(n + 1);
14        for (int i = 0; i < n; i++) {
15            s[i + 1] = s[i] + h[i];
16        }
17        for (int i = 0; i < n; i++) {
18            h[i] -= i;
19        }
20        vector<i64> s1(n + 1);
21        for (int i = 0; i < n; i++) {
22            s1[i + 1] = s1[i] + h[i];
23        }
24        vector<i64> ans(n);
25        deque<int> q;
26        int p = 0;
27        i64 sum = 0;
28        for (int i = 0; i < n; i++) {
29            while (!q.empty() && h[i] < h[q.back()]) {
30                int x = q.back();
31                q.pop_back();
32                if (q.empty()) {
33                    sum -= 1LL * h[x] * (x - p + 1);
34                } else {
35                    sum -= 1LL * h[x] * (x - q.back());
36                }
37            }
38            if (q.empty()) {
39                sum += 1LL * h[i] * (i - p + 1);
40            } else {
41                sum += 1LL * h[i] * (i - q.back());
42            }
43            q.push_back(i);
44            while (h[q[0]] + p < 0) {
45                p++;
46                sum -= h[q[0]];
47                if (q[0] < p) {
48                    q.pop_front();
49                }
50            }
51            ans[i] = (s1[i + 1] - s1[p]) - sum + s[p];
52        }
53        return ans;
54    };
55    auto ans1 = work(h);
56    reverse(h.begin(), h.end());
57    auto ans2 = work(h);
58    reverse(ans2.begin(), ans2.end());
59    reverse(h.begin(), h.end());
60    for (int i = 0; i < n; i++) {
61        ans = min(ans, ans1[i] + ans2[i] + h[i]);
62    }
63    cout << ans << "\n";
64 }
65 int main() {
66     ios::sync_with_stdio(false);
67     cin.tie(nullptr);
68     int t;
69     cin >> t;
70     while (t--) {

```

```

71     solve();
72 }
73 return 0;
74 }
```

## 635: Gaining Rating

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp is playing chess on one popular website. He has  $n$  opponents he can play with. The  $i$ -th opponent has rating equal to  $a_i$ . Monocarp's initial rating is  $x$ . Monocarp wants to raise his rating to the value  $y$  ( $y > x$ ).

When Monocarp is playing against one of the opponents, he will win if his current rating is bigger or equal to the opponent's rating. If Monocarp wins, his rating is increased by 1, otherwise it is decreased by 1. The rating of his opponent does not change.

Monocarp wants to gain rating  $y$  playing as few games as possible. But he can't just grind it, playing against weak opponents. The website has a rule that you should play against all opponents as evenly as possible. Speaking formally, if Monocarp wants to play against an opponent  $i$ , there should be no other opponent  $j$  such that Monocarp has played more games against  $i$  than against  $j$ .

Calculate the minimum possible number of games Monocarp needs to gain rating  $y$  or say it's impossible. Note that ratings of Monocarp's opponents don't change, while Monocarp's rating does change.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     i64 x, y;
7     cin >> n >> x >> y;
8     vector<i64> a(n);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11     }
12     sort(a.begin(), a.end());
13     vector<i64> pre(n + 1);
14     for (int i = 0; i < n; i++) {
15         pre[i + 1] = max(pre[i], a[i] - i);
16     }
```

```

17     i64 ans = 0;
18     int last = 0;
19     while (x < y) {
20         int j = upper_bound(pre.begin(), pre.end(), x) - pre.begin() - 1;
21         if (x + j >= y) {
22             ans += y - x;
23             break;
24         }
25         int delta = j - (n - j);
26         if (delta <= 0) {
27             cout << -1 << "\n";
28             return;
29         }
30         if (delta == last) {
31             if (j == n || y < pre[j + 1]) {
32                 i64 k = (y - x - j + delta - 1) / delta;
33                 ans += k * n;
34                 x += delta * k;
35                 ans += y - x;
36                 break;
37             } else {
38                 i64 k = max(1LL, (pre[j + 1] - x - j + delta - 1) / delta);
39                 ans += k * n;
40                 x += delta * k;
41                 continue;
42             }
43         }
44         last = delta;
45         x += delta;
46         ans += n;
47     }
48     cout << ans << "\n";
49 }
50 int main() {
51     ios::sync_with_stdio(false);
52     cin.tie(nullptr);
53     int t;
54     cin >> t;
55     while (t--) {
56         solve();
57     }
58     return 0;
59 }
```

## 636: Node Pairs

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Let's call an ordered pair of nodes  $(u, v)$  in a directed graph unidirectional if  $u \neq v$ , there exists a path from  $u$  to  $v$ , and there are no paths from  $v$  to  $u$ .

A directed graph is called  $p$ -reachable if it contains exactly  $p$  ordered pairs of nodes  $(u, v)$  such that  $u < v$  and  $u$  and  $v$  are reachable from each other. Find the minimum number of nodes required to

create a  $p$ -reachable directed graph.

Also, among all such  $p$ -reachable directed graphs with the minimum number of nodes, let  $G$  denote a graph which maximizes the number of unidirectional pairs of nodes. Find this number.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int p;
8     cin >> p;
9     vector<int> dp(p + 1);
10    for (int i = 1; i <= p; i++) {
11        dp[i] = 1E9;
12        for (int j = 2; j * (j - 1) / 2 <= i; j++) {
13            dp[i] = min(dp[i], dp[i - j * (j - 1) / 2] + j);
14        }
15    }
16    cout << dp[p] << " " << 1LL * dp[p] * (dp[p] - 1) / 2 - p << "\n";
17    return 0;
18 }
```

### 637: Valid Bitonic Permutations

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given five integers  $n, i, j, x$ , and  $y$ . Find the number of bitonic permutations  $B$ , of the numbers 1 to  $n$ , such that  $B_i = x$ , and  $B_j = y$ . Since the answer can be large, compute it modulo  $10^9 + 7$ .

A bitonic permutation is a permutation of numbers, such that the elements of the permutation first increase till a certain index  $k$ ,  $2 \leq k \leq n - 1$ , and then decrease till the end. Refer to notes for further clarification.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
```

```

8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 template<int P>
15 # struct MInt;
16 constexpr int P = 1000000007;
17 using Z = MInt<P>;
18 void solve() {
19     int n, i, j, x, y;
20     cin >> n >> i >> j >> x >> y;
21     i--, j--, x--, y--;
22     vector<Z> dp(n + 1, vector<Z>(n + 1));
23     dp[0][n] = 1;
24     for (int l = 0; l <= n; l++) {
25         for (int r = n; r > l; r--) {
26             int v = l + (n - r);
27             if ((l != i || v == x) && (l != j || v == y)) {
28                 dp[l + 1][r] += dp[l][r];
29             }
30             if ((r - 1 != i || v == x) && (r - 1 != j || v == y)) {
31                 dp[l][r - 1] += dp[l][r];
32             }
33         }
34     }
35     Z ans = 0;
36     for (int i = 0; i <= n; i++) {
37         ans += dp[i][i];
38     }
39     ans /= 2;
40     if (i == x && j == y) {
41         ans -= 1;
42     }
43     if (i == n - 1 - x && j == n - 1 - y) {
44         ans -= 1;
45     }
46     cout << ans << "\n";
47 }
48 int main() {
49     ios::sync_with_stdio(false);
50     cin.tie(nullptr);
51     int t;
52     cin >> t;
53     while (t--) {
54         solve();
55     }
56     return 0;
57 }

```

### 638: Game of the Year

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp and Polycarp are playing a computer game. This game features  $n$  bosses for the playing to kill, numbered from 1 to  $n$ .

They will fight each boss the following way:

Monocarp makes  $k$  attempts to kill the boss;

Polycarp makes  $k$  attempts to kill the boss;

Monocarp makes  $k$  attempts to kill the boss;

Polycarp makes  $k$  attempts to kill the boss;

...

Monocarp kills the  $i$ -th boss on his  $a_i$ -th attempt. Polycarp kills the  $i$ -th boss on his  $b_i$ -th attempt. After one of them kills the  $i$ -th boss, they move on to the  $(i + 1)$ -st boss. The attempt counters reset for both of them. Once one of them kills the  $n$ -th boss, the game ends.

Find all values of  $k$  from 1 to  $n$  such that Monocarp kills all bosses.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n), b(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    for (int i = 0; i < n; i++) {
12        cin >> b[i];
13    }
14    vector<int> d(n + 1);
15    for (int i = 0; i < n; i++) {
16        if (a[i] > b[i]) {
17            d[b[i]]++;
18            d[a[i]]--;
19        }
20    }
21    for (int i = 1; i <= n; i++) {
22        d[i] += d[i - 1];
23    }
24    vector<int> ans;
25    for (int k = 1; k <= n; k++) {
26        int ok = 1;
27        for (int i = k; i <= n; i += k) {
28            if (d[i]) {
29                ok = 0;
30                break;
31            }
32        }
33    }
34}
```

```

33         if (ok) ans.push_back(k);
34     }
35     cout << ans.size() << "\n";
36     for (auto x : ans) {
37         cout << x << " \n"[x == ans.back()];
38     }
39 }
40 int main() {
41     ios::sync_with_stdio(false);
42     cin.tie(nullptr);
43     int t;
44     cin >> t;
45     while (t--) {
46         solve();
47     }
48     return 0;
49 }
```

### 639: Partial Sorting

- Time limit: 1.5 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

Consider a permutation<sup>†</sup>  $p$  of length  $3n$ . Each time you can do one of the following operations:

Sort the first  $2n$  elements in increasing order.

Sort the last  $2n$  elements in increasing order.

We can show that every permutation can be made sorted in increasing order using only these operations.

Let's call  $f(p)$  the minimum number of these operations needed to make the permutation  $p$  sorted in increasing order.

Given  $n$ , find the sum of  $f(p)$  over all  $(3n)!$  permutations  $p$  of size  $3n$ .

Since the answer could be very large, output it modulo a prime  $M$ .

<sup>†</sup> A permutation of length  $n$  is an array consisting of  $n$  distinct integers from 1 to  $n$  in arbitrary order. For example,  $[2, 3, 1, 5, 4]$  is a permutation, but  $[1, 2, 2]$  is not a permutation (2 appears twice in the array), and  $[1, 3, 4]$  is also not a permutation ( $n = 3$  but there is 4 in the array).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int P;
5 using i64 = long long;
```

```

6  // assume -P <= x < 2P
7  int norm(int x) {
8      if (x < 0) {
9          x += P;
10     }
11     if (x >= P) {
12         x -= P;
13     }
14     return x;
15 }
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25 }
26 # struct Z;
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(nullptr);
30     int n;
31     cin >> n >> P;
32     vector<Z> fac(3 * n + 1), invfac(3 * n + 1);
33     fac[0] = 1;
34     for (int i = 1; i <= 3 * n; i++) {
35         fac[i] = fac[i - 1] * i;
36     }
37     invfac[3 * n] = fac[3 * n].inv();
38     for (int i = 3 * n; i; i--) {
39         invfac[i - 1] = invfac[i] * i;
40     }
41     Z ans = 3 * fac[3 * n];
42     ans -= fac[2 * n] * invfac[n] * fac[2 * n] * 2;
43     for (int i = 0; i <= n; i++) {
44         ans += fac[n] * invfac[i] * invfac[n - i] * fac[n] * invfac[i] * invfac[n - i]
45             * fac[n + i] * invfac[i] * fac[n] * fac[n];
46     }
47     ans -= fac[2 * n] * 2;
48     ans += fac[n];
49     ans -= 1;
50     cout << ans << "\n";
51     return 0;
52 }

```

### 640: Yet Another Array Counting Problem

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

The position of the leftmost maximum on the segment  $[l; r]$  of array  $x = [x_1, x_2, \dots, x_n]$  is the smallest integer  $i$  such that  $l \leq i \leq r$  and  $x_i = \max(x_l, x_{l+1}, \dots, x_r)$ .

You are given an array  $a = [a_1, a_2, \dots, a_n]$  of length  $n$ . Find the number of integer arrays  $b = [b_1, b_2, \dots, b_n]$  of length  $n$  that satisfy the following conditions:

$1 \leq b_i \leq m$  for all  $1 \leq i \leq n$ ;

for all pairs of integers  $1 \leq l \leq r \leq n$ , the position of the leftmost maximum on the segment  $[l; r]$  of the array  $b$  is equal to the position of the leftmost maximum on the segment  $[l; r]$  of the array  $a$ .

Since the answer might be very large, print its remainder modulo  $10^9 + 7$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 1000000007;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
8     if (x < 0) {
9         x += P;
10    }
11    if (x >= P) {
12        x -= P;
13    }
14    return x;
15 }
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25 }
26 # struct Z;
27 void solve() {
28     int n, m;
29     cin >> n >> m;
30     vector<int> a(n);
31     for (int i = 0; i < n; i++) {
32         cin >> a[i];
33     }
34     vector<int> lc(n, -1), rc(n, -1), s;
35     for (int i = 0; i < n; i++) {
36         while (!s.empty() && a[i] > a[s.back()]) {
37             int x = s.back();
38             rc[x] = lc[i];
39             lc[i] = x;
40             s.pop_back();
41         }
42         s.push_back(i);
43     }
44     while (s.size() > 1) {
45         int x = s.back();
46         s.pop_back();
47         rc[s.back()] = x;

```

```

48     }
49     vector<vector<Z>> dp(n, vector<Z>(m, 1));
50     function<void(int)> dfs = [&](int x) {
51         if (lc[x] != -1) {
52             dfs(lc[x]);
53             for (int i = 0; i < m; i++) {
54                 dp[x][i] *= i ? dp[lc[x]][i - 1] : 0;
55             }
56         }
57         if (rc[x] != -1) {
58             dfs(rc[x]);
59             for (int i = 0; i < m; i++) {
60                 dp[x][i] *= dp[rc[x]][i];
61             }
62         }
63         for (int i = 1; i < m; i++) {
64             dp[x][i] += dp[x][i - 1];
65         }
66     };
67     dfs(s[0]);
68     cout << dp[s[0]][m - 1] << "\n";
69 }
70 int main() {
71     ios::sync_with_stdio(false);
72     cin.tie(nullptr);
73     int t;
74     cin >> t;
75     while (t--) {
76         solve();
77     }
78     return 0;
79 }
```

## misc

### 641: Salyg1n and Array (hard version)

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. The only difference between the versions is the limit on the number of queries. In this version, you can make no more than 57 queries. You can make hacks only if both versions of the problem are solved.

This is an interactive problem!

salyg1n has given you a positive integer  $k$  and wants to play a game with you. He has chosen an array of  $n$  integers  $a_1, a_2, \dots, a_n$  ( $1 \leq a_i \leq 10^9$ ). You must print  $a_1 \oplus a_2 \oplus \dots \oplus a_n$ , where  $\oplus$  denotes the bitwise XOR operation. You can make queries of the following type:

?  $i$ : in response to this query, you will receive  $a_i \oplus a_{i+1} \oplus \dots \oplus a_{i+k-1}$ . Also, after this

query, the subarray  $a_i, a_{i+1}, \dots, a_{i+k-1}$  will be reversed, i.e., the chosen array  $a$  will become:  $a_1, a_2, \dots, a_{i-1}, a_{i+k-1}, a_{i+k-2}, \dots, a_{i+1}, a_i, a_{i+k}, \dots, a_n$ .

You can make no more than 57 queries to answer the problem.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     int ans = 0;
8     int i = 0;
9     while (i + k <= n) {
10         cout << "? " << i + 1 << endl;
11         int res;
12         cin >> res;
13         ans ^= res;
14         i += k;
15     }
16     int t = n - i;
17     cout << "? " << n - k - t / 2 + 1 << endl;
18     int res;
19     cin >> res;
20     ans ^= res;
21     cout << "? " << n - k + 1 << endl;
22     cin >> res;
23     ans ^= res;
24     cout << "! " << ans << endl;
25 }
26 int main() {
27     ios::sync_with_stdio(false);
28     cin.tie(nullptr);
29     int t;
30     cin >> t;
31     while (t--) {
32         solve();
33     }
34     return 0;
35 }
```

## 642: Playoff Fixing

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

$2^k$  teams participate in a playoff tournament. The teams are numbered from 1 to  $2^k$ , in order of decreasing strength. So, team 1 is the strongest one, team  $2^k$  is the weakest one. A team with a smaller number always defeats a team with a larger number.

First of all, the teams are arranged in some order during a procedure called seeding. Each team is assigned another unique value from 1 to  $2^k$ , called a seed, that represents its starting position in the playoff.

The tournament consists of  $2^k - 1$  games. They are held as follows: the teams are split into pairs: team with seed 1 plays against team with seed 2, team with seed 3 plays against team with seed 4 (exactly in this order), and so on (so,  $2^{k-1}$  games are played in that phase). When a team loses a game, it is eliminated.

After that, only  $2^{k-1}$  teams remain. If only one team remains, it is declared the champion; otherwise,  $2^{k-2}$  games are played: in the first one of them, the winner of the game “seed 1 vs seed 2” plays against the winner of the game “seed 3 vs seed 4”, then the winner of the game “seed 5 vs seed 6” plays against the winner of the game “seed 7 vs seed 8”, and so on. This process repeats until only one team remains.

After the tournament ends, the teams are assigned places according to the tournament phase when they were eliminated. In particular:

the winner of the tournament gets place 1;

the team eliminated in the finals gets place 2;

both teams eliminated in the semifinals get place 3;

all teams eliminated in the quarterfinals get place 5;

all teams eliminated in the 1/8 finals get place 9, and so on.

Now that we established the rules, we do a little rigging. In particular, we want:

team 1 (not team with seed 1) to take place 1;

team 2 to take place 2;

teams 3 and 4 to take place 3;

teams from 5 to 8 to take place 5, and so on.

For example, this picture describes one of the possible ways the tournament can go with  $k = 3$ , and the resulting places of the teams:

Some seeds are already reserved for some teams (we are not the only ones rigging the tournament, apparently). We have to fill the rest of the seeds with the remaining teams to achieve the desired placements. How many ways are there to do that? Since that value might be large, print it modulo 998 244 353.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int k;
39     cin >> k;
40     int n = 1 << k;
41     vector<int> a(n, -1);
42     vector<int> s(n, 1);
43     for (int i = 0; i < n; i++) {
44         int x;
45         cin >> x;
46         if (x > 0) {
47             x--;
48             a[x] = i;
49             s[i] = 0;
50         }
51     }
52     Z ans = 1;
53     while (n > 1) {
54         vector<int> ns(n / 2);
55         for (int i = 0; i < n / 2; i++) {
56             ns[i] = s[2 * i] + s[2 * i + 1];
57         }
58         vector<bool> vis(n / 2);
59         for (auto &x : a) {
60             if (x != -1) {
61                 x /= 2;
62             }

```

```

63         }
64         int cnt = n / 2;
65         for (int i = n / 2; i < n; i++) {
66             if (a[i] >= 0) {
67                 if (vis[a[i]]) {
68                     cout << 0 << "\n";
69                     return 0;
70                 }
71                 vis[a[i]] = true;
72                 cnt--;
73             }
74         }
75         ans *= comb.fac(cnt);
76         for (int i = 0; i < n / 2; i++) {
77             if (!vis[i]) {
78                 ans *= ns[i]--;
79             }
80         }
81         a.resize(n / 2);
82         s = ns;
83         n /= 2;
84     }
85     cout << ans << "\n";
86     return 0;
87 }
```

## 643: Sausage Maximization

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The Bitlandians are quite weird people. They have their own problems and their own solutions. They have their own thoughts and their own beliefs, they have their own values and their own merits. They have their own dishes and their own sausages!

In Bitland a sausage is an array of integers! A sausage's deliciousness is equal to the bitwise excluding OR (the xor operation) of all integers in that sausage.

One day, when Mr. Bitkoch (the local cook) was going to close his BitRestaurant, BitHaval and BitAryo, the most famous citizens of Bitland, entered the restaurant and each ordered a sausage.

But Mr. Bitkoch had only one sausage left. So he decided to cut a prefix (several, may be zero, first array elements) of the sausage and give it to BitHaval and a postfix (several, may be zero, last array elements) of the sausage and give it to BitAryo. Note that one or both pieces of the sausage can be empty. Of course, the cut pieces mustn't intersect (no array element can occur in both pieces).

The pleasure of BitHaval and BitAryo is equal to the bitwise XOR of their sausages' deliciousness. An empty sausage's deliciousness equals zero.

Find a way to cut a piece of sausage for BitHaval and BitAryo that maximizes the pleasure of these worthy citizens.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int N = 100000 * 40;
5 int trie[N][2];
6 int cnt = 1;
7 void add(i64 x) {
8     int p = 1;
9     for (int i = 39; i >= 0; i--) {
10         int d = x >> i & 1;
11         if (!trie[p][d]) {
12             trie[p][d] = ++cnt;
13         }
14         p = trie[p][d];
15     }
16 }
17 i64 query(i64 x) {
18     int p = 1;
19     i64 ans = 0;
20     for (int i = 39; i >= 0; i--) {
21         int d = ~x >> i & 1;
22         if (!trie[p][d]) {
23             d ^= 1;
24         } else {
25             ans |= 1LL << i;
26         }
27         p = trie[p][d];
28     }
29     return ans;
30 }
31 int main() {
32     ios::sync_with_stdio(false);
33     cin.tie(nullptr);
34     int n;
35     cin >> n;
36     vector<i64> a(n);
37     for (int i = 0; i < n; i++) {
38         cin >> a[i];
39     }
40     vector<i64> s(n+1);
41     for (int i = 0; i < n; i++) {
42         s[i+1] = s[i] ^ a[i];
43     }
44     i64 ans = 0;
45     for (int i = 0; i <= n; i++) {
46         add(s[i]);
47         ans = max(ans, query(s[i] ^ s[n]));
48     }
49     cout << ans << "\n";
50     return 0;
51 }
```

## 644: Cannon

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Bertown is under siege! The attackers have blocked all the ways out and their cannon is bombarding the city. Fortunately, Berland intelligence managed to intercept the enemies' shooting plan. Let's introduce the Cartesian system of coordinates, the origin of which coincides with the cannon's position, the Ox axis is directed rightwards in the city's direction, the Oy axis is directed upwards (to the sky). The cannon will make  $n$  more shots. The cannon balls' initial speeds are the same in all the shots and are equal to  $V$ , so that every shot is characterized by only one number  $\alpha_i$  which represents the angle at which the cannon fires. Due to the cannon's technical peculiarities this angle does not exceed 45 angles ( $/4$ ). We disregard the cannon sizes and consider the firing made from the point  $(0, 0)$ .

The balls fly according to the known physical laws of a body thrown towards the horizon at an angle:

Think of the acceleration of gravity  $g$  as equal to 9.8.

Bertown defends  $m$  walls. The  $i$ -th wall is represented as a vertical segment  $(x_i, 0) - (x_i, y_i)$ . When a ball hits a wall, it gets stuck in it and doesn't fly on. If a ball doesn't hit any wall it falls on the ground ( $y = 0$ ) and stops. If the ball exactly hits the point  $(x_i, y_i)$ , it is considered stuck.

Your task is to find for each ball the coordinates of the point where it will be located in the end.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr double g = 9.8;
5 constexpr double Pi = numbers::pi;
6 int main() {
7     ios::sync_with_stdio(false);
8     cin.tie(nullptr);
9     int n, V;
10    cin >> n >> V;
11    vector<double> alpha(n);
12    for (int i = 0; i < n; i++) {
13        cin >> alpha[i];
14    }
15    auto get = [&](double a, double x) {
16        double t = x / V / cos(a);
17        double y = V * sin(a) * t - g * t * t / 2;
18        return y;
19    };
20    vector<int> o(n);

```

```

21     iota(o.begin(), o.end(), 0);
22     sort(o.begin(), o.end(), [&](int i, int j) {
23         return alpha[i] < alpha[j];
24     });
25     vector<double> pre(n, 1E9);
26     int m;
27     cin >> m;
28     for (int i = 0; i < m; i++) {
29         double x, y;
30         cin >> x >> y;
31         int p = partition_point(o.begin(), o.end(), [&](int i) {
32             return get(alpha[i], x) <= y;
33         }) - o.begin();
34         if (p) {
35             pre[p-1] = min(pre[p-1], x);
36         }
37     }
38     for (int i = n-1; i; i--) {
39         pre[i-1] = min(pre[i-1], pre[i]);
40     }
41     vector<double> ans(n);
42     for (int i = 0; i < n; i++) {
43         ans[o[i]] = pre[i];
44     }
45     cout << fixed << setprecision(10);
46     for (int i = 0; i < n; i++) {
47         ans[i] = min(ans[i], 2. * V * V * sin(alpha[i]) * cos(alpha[i]) / g);
48         cout << ans[i] << " " << get(alpha[i], ans[i]) << "\n";
49     }
50     return 0;
51 }
```

## 645: Hercule Poirot Problem

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Today you are to solve the problem even the famous Hercule Poirot can't cope with! That's why this crime has not yet been solved and this story was never included in Agatha Christie's detective story books.

You are not informed on what crime was committed, when and where the corpse was found and other details. We only know that the crime was committed in a house that has  $n$  rooms and  $m$  doors between the pairs of rooms. The house residents are very suspicious, that's why all the doors can be locked with keys and all the keys are different. According to the provided evidence on Thursday night all the doors in the house were locked, and it is known in what rooms were the residents, and what kind of keys had any one of them. The same is known for the Friday night, when all the doors were also locked. On Friday it was raining heavily, that's why nobody left the house and nobody entered it. During the day the house residents could

open and close doors to the neighboring rooms using the keys at their disposal (every door can be opened and closed from each side);

move freely from a room to a room if a corresponding door is open;

give keys to one another, being in one room.

“Little grey matter” of Hercule Poirot are not capable of coping with such amount of information. Find out if the positions of people and keys on the Thursday night could result in the positions on Friday night, otherwise somebody among the witnesses is surely lying.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, m, k;
9     cin >> n >> m >> k;
10    vector<vector<int>> adj(n);
11    vector<int> u(m), v(m);
12    for (int i = 0; i < m; i++) {
13        cin >> u[i] >> v[i];
14        u[i]--, v[i]--;
15        adj[u[i]].push_back(v[i]);
16        adj[v[i]].push_back(u[i]);
17        if (u[i] > v[i]) {
18            swap(u[i], v[i]);
19        }
20    }
21    auto get = [&]() {
22        DSU dsu(n);
23        vector<set<string>> people(n);
24        vector<set<int>> key(n);
25        function<void(int, int)> merge = [&](int x, int y) {
26            x = dsu.find(x);
27            y = dsu.find(y);
28            if (x > y) {
29                swap(x, y);
30            }
31            people[x].merge(people[y]);
32            key[x].merge(key[y]);
33            dsu.merge(x, y);
34            for (auto z : key[x]) {
35                if (dsu.same(u[z], x) ^ dsu.same(v[z], x)) {
36                    merge(u[z], v[z]);
37                    break;
38                }
39            }
40        };
41        for (int i = 0; i < k; i++) {
42            string s;
43            cin >> s;

```

```

44         int x, e;
45         cin >> x >> e;
46         x--;
47         x = dsu.find(x);
48         people[x].insert(s);
49         while (e--) {
50             x = dsu.find(x);
51             int y;
52             cin >> y;
53             y--;
54             key[x].insert(y);
55             if (dsu.same(u[y], x) ^ dsu.same(v[y], x)) {
56                 merge(u[y], v[y]);
57             }
58         }
59         vector<int> bel(n);
60         for (int i = 0; i < n; i++) {
61             bel[i] = dsu.find(i);
62         }
63         return tuple(people, key, bel);
64     };
65     auto a = get();
66     auto b = get();
67     if (a == b) {
68         cout << "YES\n";
69     } else {
70         cout << "NO\n";
71     }
72     return 0;
73 }
74 }
```

## 646: Nearest vectors

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given the set of vectors on the plane, each of them starting at the origin. Your task is to find a pair of vectors with the minimal non-oriented angle between them.

Non-oriented angle is non-negative value, minimal between clockwise and counterclockwise direction angles. Non-oriented angle is always between 0 and  $\pi$ . For example, opposite directions vectors have angle equals to  $\pi$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 # struct Point;
```

```

6  template<class T>
7  T dot(Point<T> a, Point<T> b) {
8      return a.x * b.x + a.y * b.y;
9  }
10 template<class T>
11 T cross(Point<T> a, Point<T> b) {
12     return a.x * b.y - a.y * b.x;
13 }
14 template<class T>
15 T square(Point<T> p) {
16     return dot(p, p);
17 }
18 template<class T>
19 double length(Point<T> p) {
20     return sqrt(double(square(p)));
21 }
22 long double length(Point<long double> p) {
23     return sqrt(square(p));
24 }
25 template<class T>
26 # struct Line;
27 template<class T>
28 Point<T> rotate(Point<T> a) {
29     return Point(-a.y, a.x);
30 }
31 template<class T>
32 int sgn(Point<T> a) {
33     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
34 }
35 template<class T>
36 bool pointOnLineLeft(Point<T> p, Line<T> l) {
37     return cross(l.b - l.a, p - l.a) > 0;
38 }
39 template<class T>
40 Point<T> lineIntersection(Line<T> l1, Line<T> l2) {
41     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1
        .a - l1.b));
42 }
43 template<class T>
44 bool pointOnSegment(Point<T> p, Line<T> l) {
45     return cross(p - l.a, l.b - l.a) == 0 && min(l.a.x, l.b.x) <= p.x && p.x <= max(l.a.x,
        l.b.x)
        && min(l.a.y, l.b.y) <= p.y && p.y <= max(l.a.y, l.b.y);
46 }
47 template<class T>
48 bool pointInPolygon(Point<T> a, vector<Point<T>> p) {
49     int n = p.size();
50     for (int i = 0; i < n; i++) {
51         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
52             return true;
53         }
54     }
55     int t = 0;
56     for (int i = 0; i < n; i++) {
57         auto u = p[i];
58         auto v = p[(i + 1) % n];
59         if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
60             t ^= 1;
61         }
62         if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
63             t ^= 1;
64         }
65     }
66     return t == 1;
67 }
```

```

68 }
69 // 0 : not intersect
70 // 1 : strictly intersect
71 // 2 : overlap
72 // 3 : intersect at endpoint
73 template<class T>
74 tuple<int, Point<T>, Point<T>> segmentIntersection(Line<T> l1, Line<T> l2) {
75     if (max(l1.a.x, l1.b.x) < min(l2.a.x, l2.b.x)) {
76         return {0, Point<T>(), Point<T>()};
77     }
78     if (min(l1.a.x, l1.b.x) > max(l2.a.x, l2.b.x)) {
79         return {0, Point<T>(), Point<T>()};
80     }
81     if (max(l1.a.y, l1.b.y) < min(l2.a.y, l2.b.y)) {
82         return {0, Point<T>(), Point<T>()};
83     }
84     if (min(l1.a.y, l1.b.y) > max(l2.a.y, l2.b.y)) {
85         return {0, Point<T>(), Point<T>()};
86     }
87     if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
88         if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
89             return {0, Point<T>(), Point<T>()};
90         } else {
91             auto maxx1 = max(l1.a.x, l1.b.x);
92             auto minx1 = min(l1.a.x, l1.b.x);
93             auto maxy1 = max(l1.a.y, l1.b.y);
94             auto miny1 = min(l1.a.y, l1.b.y);
95             auto maxx2 = max(l2.a.x, l2.b.x);
96             auto minx2 = min(l2.a.x, l2.b.x);
97             auto maxy2 = max(l2.a.y, l2.b.y);
98             auto miny2 = min(l2.a.y, l2.b.y);
99             Point<T> p1(max(minx1, minx2), max(miny1, miny2));
100            Point<T> p2(min(maxx1, maxx2), min(maxy1, maxy2));
101            if (!pointOnSegment(p1, l1)) {
102                swap(p1.y, p2.y);
103            }
104            if (p1 == p2) {
105                return {3, p1, p2};
106            } else {
107                return {2, p1, p2};
108            }
109        }
110    }
111    auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
112    auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
113    auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
114    auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
115    if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) || (cp3 < 0
116        && cp4 < 0)) {
117        return {0, Point<T>(), Point<T>()};
118    }
119    Point p = lineIntersection(l1, l2);
120    if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
121        return {1, p, p};
122    } else {
123        return {3, p, p};
124    }
125 template<class T>
126 bool segmentInPolygon(Line<T> l, vector<Point<T>> p) {
127     int n = p.size();
128     if (!pointInPolygon(l.a, p)) {
129         return false;
130     }

```

```
131     if (!pointInPolygon(l.b, p)) {
132         return false;
133     }
134     for (int i = 0; i < n; i++) {
135         auto u = p[i];
136         auto v = p[(i + 1) % n];
137         auto w = p[(i + 2) % n];
138         auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
139         if (t == 1) {
140             return false;
141         }
142         if (t == 0) {
143             continue;
144         }
145         if (t == 2) {
146             if (pointOnSegment(v, l) && v != l.a && v != l.b) {
147                 if (cross(v - u, w - v) > 0) {
148                     return false;
149                 }
150             }
151         } else {
152             if (p1 != u && p1 != v) {
153                 if (pointOnLineLeft(l.a, Line(v, u))
154                     || pointOnLineLeft(l.b, Line(v, u))) {
155                     return false;
156                 }
157             } else if (p1 == v) {
158                 if (l.a == v) {
159                     if (pointOnLineLeft(u, l)) {
160                         if (pointOnLineLeft(w, l)
161                             && pointOnLineLeft(w, Line(u, v))) {
162                             return false;
163                         }
164                     } else {
165                         if (pointOnLineLeft(w, l)
166                             || pointOnLineLeft(w, Line(u, v))) {
167                             return false;
168                         }
169                     }
170                 } else if (l.b == v) {
171                     if (pointOnLineLeft(u, Line(l.b, l.a))) {
172                         if (pointOnLineLeft(w, Line(l.b, l.a))
173                             && pointOnLineLeft(w, Line(u, v))) {
174                             return false;
175                         }
176                     } else {
177                         if (pointOnLineLeft(w, Line(l.b, l.a))
178                             || pointOnLineLeft(w, Line(u, v))) {
179                             return false;
180                         }
181                     }
182                 } else {
183                     if (pointOnLineLeft(u, l)) {
184                         if (pointOnLineLeft(w, Line(l.b, l.a))
185                             || pointOnLineLeft(w, Line(u, v))) {
186                             return false;
187                         }
188                     } else {
189                         if (pointOnLineLeft(w, l)
190                             || pointOnLineLeft(w, Line(u, v))) {
191                             return false;
192                         }
193                     }
194                 }
195             }
196         }
197     }
198 }
```

```

195         }
196     }
197     return true;
198 }
199 template<class T>
200 vector<Point<T>> hp(vector<vector<Line<T>> lines) {
201     sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
202         auto d1 = l1.b - l1.a;
203         auto d2 = l2.b - l2.a;
204         if (sgn(d1) != sgn(d2)) {
205             return sgn(d1) == 1;
206         }
207         return cross(d1, d2) > 0;
208     });
209     deque<Line<T>> ls;
210     deque<Point<T>> ps;
211     for (auto l : lines) {
212         if (ls.empty()) {
213             ls.push_back(l);
214             continue;
215         }
216         while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
217             ps.pop_back();
218             ls.pop_back();
219         }
220         while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
221             ps.pop_front();
222             ls.pop_front();
223         }
224         if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
225             if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
226                 if (!pointOnLineLeft(ls.back().a, l)) {
227                     assert(ls.size() == 1);
228                     ls[0] = l;
229                 }
230                 continue;
231             }
232             return {};
233         }
234         ps.push_back(lineIntersection(ls.back(), l));
235         ls.push_back(l);
236     }
237     while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
238         ps.pop_back();
239         ls.pop_back();
240     }
241     if (ls.size() <= 2) {
242         return {};
243     }
244     ps.push_back(lineIntersection(ls[0], ls.back()));
245     return vector(ps.begin(), ps.end());
246 }
247 template<class T>
248 # struct Frac;
249 int main() {
250     ios::sync_with_stdio(false);
251     cin.tie(nullptr);
252     int n;
253     cin >> n;
254     vector<Point<i64>> p(n);
255     for (int i = 0; i < n; i++) {
256         cin >> p[i].x >> p[i].y;
257     }

```

```

259     int a = 0, b = 1;
260     int S = -1;
261     Frac<__int128> ans(1E18);
262     vector<int> o(n);
263     iota(o.begin(), o.end(), 0);
264     sort(o.begin(), o.end(), [&](int i, int j) {
265         if (sgn(p[i]) != sgn(p[j])) {
266             return sgn(p[i]) == 1;
267         } else {
268             return cross(p[i], p[j]) > 0;
269         }
270     });
271     for (int i = 0; i < n; i++) {
272         int x = o[i], y = o[(i+1) % n];
273         Frac<__int128> res(dot(p[x], p[y]) * dot(p[x], p[y]), square(p[x]) * square(p[y]));
274         ;
275         int s = dot(p[x], p[y]) > 0 ? 1 : -1;
276         if (s > S || (S == s && (res - ans) * s > 0)) {
277             ans = res;
278             S = s;
279             a = x;
280             b = y;
281         }
282         cout << a+1 << " " << b+1 << "\n";
283     }
284 }
```

## 647: New Game with a Chess Piece

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: input.txt
- Output file: output.txt

Petya and Vasya are inventing a new game that requires a rectangular board and one chess piece. At the beginning of the game the piece stands in the upper-left corner of the board. Two players move the piece in turns. Each turn the chess piece can be moved either one square to the right or one square down or jump  $k$  squares diagonally down and to the right. The player who cant move the piece loses.

The guys havent yet thought what to call the game or the best size of the board for it. Your task is to write a program that can determine the outcome of the game depending on the board size.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 #ifdef ONLINE_JUDGE
5     ifstream fin("input.txt");

```

```

6     ofstream fout("output.txt");
7 #else
8     #define fin cin
9     #define fout cout
10 #endif
11 int main() {
12     int t, k;
13     fin >> t >> k;
14     while (t--) {
15         int n, m;
16         fin >> n >> m;
17         n--, m--;
18         int ans;
19         if (min(n, m) % (k + 1) == k) {
20             ans = 1;
21         } else {
22             ans = (n + m - min(n, m)) / (k + 1) * (k != 1) % 2;
23         }
24         fout << "-+"[ans] << "\n";
25     }
26     return 0;
27 }
```

**648: Test**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Sometimes it is hard to prepare tests for programming problems. Now Bob is preparing tests to new problem about strings - input data to his problem is one string. Bob has 3 wrong solutions to this problem. The first gives the wrong answer if the input data contains the substring s1, the second enters an infinite loop if the input data contains the substring s2, and the third requires too much memory if the input data contains the substring s3. Bob wants these solutions to fail single test. What is the minimal length of test, which couldn't be passed by all three Bob's solutions?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 vector<int> KMP(string s) {
5     int n = s.size();
6     vector<int> f(n + 1);
7     for (int i = 1, j = 0; i < n; i++) {
8         while (j && s[i] != s[j]) {
9             j = f[j];
10        }
```

```
11         j += (s[i] == s[j]);
12         f[i + 1] = j;
13     }
14     return f;
15 }
16 int main() {
17     ios::sync_with_stdio(false);
18     cin.tie(nullptr);
19     vector<string> s(3);
20     for (int i = 0; i < 3; i++) {
21         cin >> s[i];
22     }
23     sort(s.begin(), s.end(), [&](auto a, auto b) {
24         return a.size() > b.size();
25     });
26     vector<string> t;
27     for (auto s : s) {
28         int n = s.size();
29         auto f = KMP(s);
30         bool ok = true;
31         for (auto t : t) {
32             int j = 0;
33             for (auto c : t) {
34                 while (j && c != s[j]) {
35                     j = f[j];
36                 }
37                 j += (c == s[j]);
38                 if (j == n) {
39                     break;
40                 }
41             }
42             if (j == n) {
43                 ok = false;
44                 break;
45             }
46         }
47         if (ok) {
48             t.push_back(s);
49         }
50     }
51     s = move(t);
52     sort(s.begin(), s.end());
53     int ans = 1E9;
54     do {
55         int sum = 0;
56         for (auto s : s) {
57             sum += s.size();
58         }
59         for (int i = 1; i < s.size(); i++) {
60             auto f = KMP(s[i] + s[i - 1]);
61             int j = f.size() - 1;
62             while (j > min(s[i].size(), s[i - 1].size())) {
63                 j = f[j];
64             }
65             sum -= j;
66         }
67         ans = min(ans, sum);
68     } while (next_permutation(s.begin(), s.end()));
69     cout << ans << "\n";
70     return 0;
71 }
```

## 649: Berland collider

- Time limit: 1.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Recently the construction of Berland collider has been completed. Collider can be represented as a long narrow tunnel that contains  $n$  particles. We associate with collider 1-dimensional coordinate system, going from left to right. For each particle we know its coordinate and velocity at the moment of start of the collider. The velocities of the particles don't change after the launch of the collider. Berland scientists think that the big bang will happen at the first collision of particles, whose velocities differs in directions. Help them to determine how much time elapses after the launch of the collider before the big bang happens.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 # struct Point;
6 template<class T>
7 T dot(Point<T> a, Point<T> b) {
8     return a.x * b.x + a.y * b.y;
9 }
10 template<class T>
11 T cross(Point<T> a, Point<T> b) {
12     return a.x * b.y - a.y * b.x;
13 }
14 template<class T>
15 T square(Point<T> p) {
16     return dot(p, p);
17 }
18 template<class T>
19 double length(Point<T> p) {
20     return sqrt(double(square(p)));
21 }
22 long double length(Point<long double> p) {
23     return sqrt(square(p));
24 }
25 template<class T>
26 # struct Line;
27 template<class T>
28 Point<T> rotate(Point<T> a) {
29     return Point(-a.y, a.x);
30 }
31 template<class T>
32 int sgn(Point<T> a) {
33     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
34 }
```

```

35 template<class T>
36 bool pointOnLineLeft(Point<T> p, Line<T> l) {
37     return cross(l.b - l.a, p - l.a) > 0;
38 }
39 template<class T>
40 Point<T> lineIntersection(Line<T> l1, Line<T> l2) {
41     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1
        .a - l1.b));
42 }
43 template<class T>
44 bool pointOnSegment(Point<T> p, Line<T> l) {
45     return cross(p - l.a, l.b - l.a) == 0 && min(l.a.x, l.b.x) <= p.x && p.x <= max(l.a.x,
        l.b.x)
46     && min(l.a.y, l.b.y) <= p.y && p.y <= max(l.a.y, l.b.y);
47 }
48 template<class T>
49 bool pointInPolygon(Point<T> a, vector<Point<T>> p) {
50     int n = p.size();
51     for (int i = 0; i < n; i++) {
52         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
53             return true;
54         }
55     }
56     int t = 0;
57     for (int i = 0; i < n; i++) {
58         auto u = p[i];
59         auto v = p[(i + 1) % n];
60         if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
61             t ^= 1;
62         }
63         if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
64             t ^= 1;
65         }
66     }
67     return t == 1;
68 }
69 // 0 : not intersect
70 // 1 : strictly intersect
71 // 2 : overlap
72 // 3 : intersect at endpoint
73 template<class T>
74 tuple<int, Point<T>, Point<T>> segmentIntersection(Line<T> l1, Line<T> l2) {
75     if (max(l1.a.x, l1.b.x) < min(l2.a.x, l2.b.x)) {
76         return {0, Point<T>(), Point<T>()};
77     }
78     if (min(l1.a.x, l1.b.x) > max(l2.a.x, l2.b.x)) {
79         return {0, Point<T>(), Point<T>()};
80     }
81     if (max(l1.a.y, l1.b.y) < min(l2.a.y, l2.b.y)) {
82         return {0, Point<T>(), Point<T>()};
83     }
84     if (min(l1.a.y, l1.b.y) > max(l2.a.y, l2.b.y)) {
85         return {0, Point<T>(), Point<T>()};
86     }
87     if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
88         if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
89             return {0, Point<T>(), Point<T>()};
90         } else {
91             int maxx1 = max(l1.a.x, l1.b.x);
92             int minx1 = min(l1.a.x, l1.b.x);
93             int maxy1 = max(l1.a.y, l1.b.y);
94             int miny1 = min(l1.a.y, l1.b.y);
95             int maxx2 = max(l2.a.x, l2.b.x);
96             int minx2 = min(l2.a.x, l2.b.x);

```

```

97         int maxy2 = max(l2.a.y, l2.b.y);
98         int miny2 = min(l2.a.y, l2.b.y);
99         Point<T> p1(max(minx1, minx2), max(miny1, miny2));
100        Point<T> p2(min(maxx1, maxx2), min(maxy1, maxy2));
101        if (!pointOnSegment(p1, l1)) {
102            swap(p1.y, p2.y);
103        }
104        if (p1 == p2) {
105            return {3, p1, p2};
106        } else {
107            return {2, p1, p2};
108        }
109    }
110 }
111 auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
112 auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
113 auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
114 auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
115 if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) || (cp3 < 0
116     && cp4 < 0)) {
117     return {0, Point<T>(), Point<T>()};
118 }
119 Point p = lineIntersection(l1, l2);
120 if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
121     return {1, p, p};
122 } else {
123     return {3, p, p};
124 }
125 template<class T>
126 bool segmentInPolygon(Line<T> l, vector<Point<T>> p) {
127     int n = p.size();
128     if (!pointInPolygon(l.a, p)) {
129         return false;
130     }
131     if (!pointInPolygon(l.b, p)) {
132         return false;
133     }
134     for (int i = 0; i < n; i++) {
135         auto u = p[i];
136         auto v = p[(i + 1) % n];
137         auto w = p[(i + 2) % n];
138         auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
139         if (t == 1) {
140             return false;
141         }
142         if (t == 0) {
143             continue;
144         }
145         if (t == 2) {
146             if (pointOnSegment(v, l) && v != l.a && v != l.b) {
147                 if (cross(v - u, w - v) > 0) {
148                     return false;
149                 }
150             }
151         } else {
152             if (p1 != u && p1 != v) {
153                 if (pointOnLineLeft(l.a, Line(v, u))
154                     || pointOnLineLeft(l.b, Line(v, u))) {
155                     return false;
156                 }
157             } else if (p1 == v) {
158                 if (l.a == v) {
159                     if (pointOnLineLeft(u, l)) {

```

```

160             if (pointOnLineLeft(w, l)
161                 && pointOnLineLeft(w, Line(u, v))) {
162                 return false;
163             }
164         } else {
165             if (pointOnLineLeft(w, l)
166                 || pointOnLineLeft(w, Line(u, v))) {
167                 return false;
168             }
169         }
170     } else if (l.b == v) {
171         if (pointOnLineLeft(u, Line(l.b, l.a))) {
172             if (pointOnLineLeft(w, Line(l.b, l.a))
173                 && pointOnLineLeft(w, Line(u, v))) {
174                 return false;
175             }
176         } else {
177             if (pointOnLineLeft(w, Line(l.b, l.a))
178                 || pointOnLineLeft(w, Line(u, v))) {
179                 return false;
180             }
181         }
182     } else {
183         if (pointOnLineLeft(u, l)) {
184             if (pointOnLineLeft(w, Line(l.b, l.a))
185                 || pointOnLineLeft(w, Line(u, v))) {
186                 return false;
187             }
188         } else {
189             if (pointOnLineLeft(w, l)
190                 || pointOnLineLeft(w, Line(u, v))) {
191                 return false;
192             }
193         }
194     }
195 }
196 }
197 }
198
199 return true;
200 }
201 template<class T>
202 vector<Point<T>> hp(vector<Line<T>> lines) {
203     sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
204         auto d1 = l1.b - l1.a;
205         auto d2 = l2.b - l2.a;
206         if (sgn(d1) != sgn(d2)) {
207             return sgn(d1) == 1;
208         }
209         return cross(d1, d2) > 0;
210     });
211     deque<Line<T>> ls;
212     deque<Point<T>> ps;
213     for (auto l : lines) {
214         if (ls.empty()) {
215             ls.push_back(l);
216             continue;
217         }
218         while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
219             ps.pop_back();
220             ls.pop_back();
221         }
222         while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
223             ps.pop_front();
224             ls.pop_front();

```

```

224         }
225         if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
226             if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
227                 if (!pointOnLineLeft(ls.back().a, l)) {
228                     assert(ls.size() == 1);
229                     ls[0] = l;
230                 }
231                 continue;
232             }
233             return {};
234         }
235         ps.push_back(lineIntersection(ls.back(), l));
236         ls.push_back(l);
237     }
238     while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
239         ps.pop_back();
240         ls.pop_back();
241     }
242     if (ls.size() <= 2) {
243         return {};
244     }
245     ps.push_back(lineIntersection(ls[0], ls.back()));
246     return vector(ps.begin(), ps.end());
247 }
248 int main() {
249     ios::sync_with_stdio(false);
250     cin.tie(nullptr);
251     int n;
252     cin >> n;
253     vector<int> x(n), v(n);
254     for (int i = 0; i < n; i++) {
255         cin >> x[i] >> v[i];
256     }
257     vector<Point<i64>> h;
258     vector<int> order(n);
259     iota(order.begin(), order.end(), 0);
260     sort(order.begin(), order.end(), [&](int i, int j) {
261         return x[i] < x[j];
262     });
263     double ans = 2E9;
264     for (auto i : order) {
265         if (v[i] > 0) {
266             Point<i64> p(x[i], v[i]);
267             while (h.size() >= 2 && cross(h.back() - h.end()[-2], p - h.back()) >= 0) {
268                 h.pop_back();
269             }
270             h.push_back(p);
271         } else if (!h.empty()) {
272             Point<i64> p(x[i], v[i]);
273             int lo = 0, hi = h.size() - 1;
274             while (lo < hi) {
275                 int m = (lo + hi) / 2;
276                 if (cross(h[m] - p, h[m + 1] - p) < 0) {
277                     lo = m + 1;
278                 } else {
279                     hi = m;
280                 }
281             }
282             ans = min(ans, 1. * (x[i] - h[lo].x) / (h[lo].y - v[i]));
283         }
284     }
285     if (ans == 2E9) {
286         cout << -1 << "\n";
287         return 0;

```

```

288     }
289     cout << fixed << setprecision(10);
290     cout << ans << "\n";
291     return 0;
292 }
```

## 650: Deletion of Repeats

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Once Bob saw a string. It contained so many different letters, that the letters were marked by numbers, but at the same time each letter could be met in the string at most 10 times. Bob didn't like that string, because it contained repeats: a repeat of length  $x$  is such a substring of length  $2x$ , that its first half coincides character by character with its second half. Bob started deleting all the repeats from the string. He does it as follows: while it's possible, Bob takes the shortest repeat, if it is not unique, he takes the leftmost one, and deletes its left half and everything that is to the left of this repeat.

You're given the string seen by Bob. Find out, what it will look like after Bob deletes all the repeats in the way described above.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
```

```

27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr i64 P = i64(1E18) + 9;
33 using Z = MLong<P>;
34 const Z B = 114514 * 1E9;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int n;
39     cin >> n;
40     vector<int> a(n);
41     map<int, vector<int>> pos;
42     for (int i = 0; i < n; i++) {
43         cin >> a[i];
44         pos[a[i]].push_back(i + 1);
45     }
46     vector<Z> h(n + 1), pw(n + 1);
47     pw[0] = 1;
48     for (int i = 0; i < n; i++) {
49         h[i + 1] = h[i] * B + a[i];
50         pw[i + 1] = pw[i] * B;
51     }
52     auto get = [&](int l, int r) {
53         return h[r] - h[l] * pw[r - l];
54     };
55     vector<pair<int, int>> f;
56     for (auto [_, v] : pos) {
57         for (auto i : v) {
58             for (auto j : v) {
59                 if (i == j) {
60                     break;
61                 }
62                 if (i <= 2 * j && get(j, i) == get(2 * j - i, j)) {
63                     f.emplace_back(2 * (i - j), 2 * j - i);
64                 }
65             }
66         }
67     }
68     sort(f.begin(), f.end());
69     int L = 0;
70     for (auto [len, l] : f) {
71         if (l >= L) {
72             L = l + len / 2;
73         }
74     }
75     cout << n - L << "\n";
76     for (int i = L; i < n; i++) {
77         cout << a[i] << " \n"[i == n - 1];
78     }
79     return 0;
80 }
```

## 651: Queue

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

On a cold winter evening our hero Vasya stood in a railway queue to buy a ticket for Codeforces championship final. As it usually happens, the cashier said he was going to be away for 5 minutes and left for an hour. Then Vasya, not to get bored, started to analyze such a mechanism as a queue. The findings astonished Vasya.

Every man is characterized by two numbers:  $a_i$ , which is the importance of his current task (the greater the number is, the more important the task is) and number  $c_i$ , which is a picture of his conscience. Numbers  $a_i$  form the permutation of numbers from 1 to  $n$ .

Let the queue consist of  $n - 1$  people at the moment. Let's look at the way the person who came number  $n$  behaves. First, he stands at the end of the queue and does the following: if importance of the task  $a_i$  of the man in front of him is less than  $a_n$ , they swap their places (it looks like this: the man number  $n$  asks the one before him: "Erm... Excuse me please but it's very important for me... could you please let me move up the queue?"), then he again poses the question to the man in front of him and so on. But in case when  $a_i$  is greater than  $a_n$ , moving up the queue stops. However, the man number  $n$  can perform the operation no more than  $c_n$  times.

In our task let us suppose that by the moment when the man number  $n$  joins the queue, the process of swaps between  $n - 1$  will have stopped. If the swap is possible it necessarily takes place.

Your task is to help Vasya model the described process and find the order in which the people will stand in queue when all the swaps stops.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
5 # struct Info;
6 Info operator+(Info a, Info b) {
7     return {max(a.imp, b.imp), 0};
8 }
9 # struct Node;
10 void pull(Node *t) {
11     t->sum = t->info;
12     t->siz = 1;
13     if (t->l) {
14         t->sum = t->l->sum + t->sum;
15         t->siz += t->l->siz;
16     }
17     if (t->r) {
18         t->sum = t->sum + t->r->sum;
19         t->siz += t->r->siz;
20     }
21 }
22 pair<Node *, Node *> splitAt(Node *t, int p) {
23     if (!t) {

```

```

24         return {t, t};
25     }
26     if (p <= (t->l ? t->l->siz : 0)) {
27         auto [l, r] = splitAt(t->l, p);
28         t->l = r;
29         pull(t);
30         return {l, t};
31     } else {
32         auto [l, r] = splitAt(t->r, p - 1 - (t->l ? t->l->siz : 0));
33         t->r = l;
34         pull(t);
35         return {t, r};
36     }
37 }
38 void insertAt(Node *&t, int p, Node *x) {
39     if (!t) {
40         t = x;
41         return;
42     }
43     if (x->w < t->w) {
44         auto [l, r] = splitAt(t, p);
45         t = x;
46         t->l = l;
47         t->r = r;
48         pull(t);
49         return;
50     }
51     if (p <= (t->l ? t->l->siz : 0)) {
52         insertAt(t->l, p, x);
53     } else {
54         insertAt(t->r, p - 1 - (t->l ? t->l->siz : 0), x);
55     }
56     pull(t);
57 }
58 Node *merge(Node *a, Node *b) {
59     if (!a) {
60         return b;
61     }
62     if (!b) {
63         return a;
64     }
65     if (a->w < b->w) {
66         a->r = merge(a->r, b);
67         pull(a);
68         return a;
69     } else {
70         b->l = merge(a, b->l);
71         pull(b);
72         return b;
73     }
74 }
75 int query(Node *t, int v) {
76     if (!t) {
77         return 0;
78     }
79     if (t->sum.imp < v) {
80         return t->siz;
81     }
82     int res = query(t->r, v);
83     if (res != (t->r ? t->r->siz : 0)) {
84         return res;
85     }
86     if (t->info.imp > v) {
87         return res;

```

```

88     }
89     return res + 1 + query(t->l, v);
90   }
91   void dfs(Node *t) {
92     if (!t) {
93       return;
94     }
95     dfs(t->l);
96     cout << t->info.id << " ";
97     dfs(t->r);
98   }
99   int main() {
100     ios::sync_with_stdio(false);
101     cin.tie(nullptr);
102     int n;
103     cin >> n;
104     Node *t = nullptr;
105     for (int i = 0; i < n; i++) {
106       int a, c;
107       cin >> a >> c;
108       int p = i - min(c, query(t, a));
109       auto x = new Node;
110       x->info = {a, i + 1};
111       x->sum = {a, 0};
112       insertAt(t, p, x);
113     }
114     dfs(t);
115     cout << "\n";
116     return 0;
117   }

```

## 652: Vittorio Plays with LEGO Bricks

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vittorio is playing with his new LEGO Duplo bricks. All the bricks have the shape of a square cuboid with a  $2 \times 2$  square base and a height of 1. They can be arranged in the 3D space to build structures, provided that the following rules are met:

No two bricks can intersect, but they can touch on their faces.

The corners of every brick must have integer coordinates (so bricks are axis-aligned) and the  $z$  coordinates of all corners must be non-negative.

The square bases of every brick must be parallel to the ground (i.e. the plane  $z = 0$ ).

The lower base of any brick that is not touching the ground must touch the upper base of some other brick in a region of positive area (when this happens, the two bricks stay attached to each other thanks to small studs).

For example, this is a valid structure:

Vittorio wants to build a structure that includes purple bricks in the following  $n$  positions:  $(x_1, 0, h)$ ,  $(x_2, 0, h), \dots, (x_n, 0, h)$  - these are the coordinates of the centers of their lower bases; note that all of these bricks have  $y$  coordinate equal to 0 and  $z$  coordinate equal to  $h$ . Vittorio will use additional bricks of other colors to support the purple bricks. He is willing to place bricks only in positions where the center of the lower base has  $y$  coordinate equal to 0. What is the minimum number of additional bricks needed?

It can be shown that a valid construction always exists.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, h;
8     cin >> n >> h;
9     vector<int> x(n);
10    for (int i = 0; i < n; i++) {
11        cin >> x[i];
12    }
13    constexpr i64 inf = 1E18;
14    vector dp(n, vector<i64>(n, inf));
15    for (int i = 0; i < n; i++) {
16        dp[i][i] = h;
17    }
18    for (int l = n - 1; l >= 0; l--) {
19        for (int r = l + 1; r < n; r++) {
20            for (int m = l; m < r; m++) {
21                dp[l][r] = min(dp[l][r], dp[l][m] + dp[m + 1][r] - max(0, h + 1 - (x[r] -
22                                x[l] + 1) / 2));
23            }
24        }
25        cout << dp[0][n - 1] << "\n";
26    }
27 }
```

## 653: Chemistry Lab

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Monocarp is planning on opening a chemistry lab. During the first month, he's going to distribute solutions of a certain acid.

First, he will sign some contracts with a local chemistry factory. Each contract provides Monocarp with an unlimited supply of some solution of the same acid. The factory provides  $n$  contract options, numbered from 1 to  $n$ . The  $i$ -th solution has a concentration of  $x_i\%$ , the contract costs  $w_i$  burles, and Monocarp will be able to sell it for  $c_i$  burles per liter.

Monocarp is expecting  $k$  customers during the first month. Each customer will buy a liter of a  $y\%$ -solution, where  $y$  is a real number chosen uniformly at random from 0 to 100 independently for each customer. More formally, the probability of number  $y$  being less than or equal to some  $t$  is  $P(y \leq t) = \frac{t}{100}$ .

Monocarp can mix the solution that he signed the contracts with the factory for, at any ratio. More formally, if he has contracts for  $m$  solutions with concentrations  $x_1, x_2, \dots, x_m$ , then, for these solutions, he picks their volumes  $a_1, a_2, \dots, a_m$  so that  $\sum_{i=1}^m a_i = 1$  (exactly 1 since each customer wants exactly one liter of a certain solution).

The concentration of the resulting solution is  $\sum_{i=1}^m x_i \cdot a_i$ . The price of the resulting solution is  $\sum_{i=1}^m c_i \cdot a_i$ .

If Monocarp can obtain a solution of concentration  $y\%$ , then he will do it while maximizing its price (the cost for the customer). Otherwise, the customer leaves without buying anything, and the price is considered equal to 0.

Monocarp wants to sign some contracts with a factory (possibly, none or all of them) so that the expected profit is maximized - the expected total price of the sold solutions for all  $k$  customers minus the total cost of signing the contracts from the factory.

Print the maximum expected profit Monocarp can achieve.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, k;
8     cin >> n >> k;
9     vector<array<int, 3>> a(n);
10    for (int i = 0; i < n; i++) {
11        int x, w, c;
12        cin >> x >> w >> c;
13        a[i] = {x, w, c};
14    }
15    sort(a.begin(), a.end());
16    vector<double> dp(n);
17    double ans = 0;
18    for (int i = 0; i < n; i++) {
19        auto [xi, wi, ci] = a[i];
20        dp[i] = -wi;
21        for (int j = 0; j < i; j++) {

```

```

22         auto [xj, wj, cj] = a[j];
23         if (xi != xj) {
24             dp[i] = max(dp[i], dp[j] - wi + (ci + cj) * .005 * (xi - xj) * k);
25         }
26     }
27     ans = max(ans, dp[i]);
28 }
29 cout << fixed << setprecision(10);
30 cout << ans << "\n";
31 return 0;
32 }
```

## 654: Intersection and Union

- Time limit: 5 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given  $n$  segments on the coordinate axis. The  $i$ -th segment is  $[l_i, r_i]$ . Let's denote the set of all integer points belonging to the  $i$ -th segment as  $S_i$ .

Let  $A \cup B$  be the union of two sets  $A$  and  $B$ ,  $A \cap B$  be the intersection of two sets  $A$  and  $B$ , and  $A \oplus B$  be the symmetric difference of  $A$  and  $B$  (a set which contains all elements of  $A$  and all elements of  $B$ , except for the ones that belong to both sets).

Let  $[op_1, op_2, \dots, op_{n-1}]$  be an array where each element is either  $\cup$ ,  $\oplus$ , or  $\cap$ . Over all  $3^{n-1}$  ways to choose this array, calculate the sum of the following values:

$$|(((S_1 \ op_1 \ S_2) \ op_2 \ S_3) \ op_3 \ S_4) \dots \ op_{n-1} \ S_n|$$

In this expression,  $|S|$  denotes the size of the set  $S$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 998244353;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
8     if (x < 0) {
9         x += P;
10    }
11    if (x >= P) {
```

```

12         x -= P;
13     }
14     return x;
15 }
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25 }
26 # struct Z;template<class Info,
27 class Merge = plus<Info>>
28 # struct SegmentTree;
29 # struct Matrix;
30 Matrix operator+(const Matrix &a, const Matrix &b) {
31     Matrix c;
32     for (int i = 0; i < 2; i++) {
33         for (int j = 0; j < 2; j++) {
34             for (int k = 0; k < 2; k++) {
35                 c.a[i][j] += a.a[i][k] * b.a[k][j];
36             }
37         }
38     }
39     return c;
40 }
41 Matrix M0, M1;
42 int main() {
43     ios::sync_with_stdio(false);
44     cin.tie(nullptr);
45     M0.a[0][0] = 3;
46     M0.a[0][1] = 0;
47     M0.a[1][0] = 1;
48     M0.a[1][1] = 2;
49     M1.a[0][0] = 1;
50     M1.a[0][1] = 2;
51     M1.a[1][0] = 1;
52     M1.a[1][1] = 2;
53     int n;
54     cin >> n;
55     vector<int> l(n), r(n);
56     int max = 0;
57     for (int i = 0; i < n; i++) {
58         cin >> l[i] >> r[i];
59         max = max(max, r[i]);
60     }
61     max++;
62     vector<vector<int>> add(max), del(max);
63     for (int i = 0; i < n; i++) {
64         add[l[i]].push_back(i);
65         del[r[i]].push_back(i);
66     }
67     int a0 = 0;
68     SegmentTree<Matrix> seg(vector(n - 1, M0));
69     Z ans = 0;
70     for (int i = 0; i < max; i++) {
71         for (auto j : add[i]) {
72             if (j) {
73                 seg.modify(j - 1, M1);
74             } else {
75                 a0 = 1;

```

```

76         }
77     }
78     ans += seg.info[1].a[a0][1];
79     for (auto j : del[i]) {
80         if (j) {
81             seg.modify(j - 1, M0);
82         } else {
83             a0 = 0;
84         }
85     }
86 }
87 cout << ans << "\n";
88 return 0;
89 }
```

## rose

### brute force

#### 655: Vova Escapes the Matrix

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Following a world tour, Vova got himself trapped inside an  $n \times m$  matrix. Rows of this matrix are numbered by integers from 1 to  $n$  from top to bottom, and the columns are numbered by integers from 1 to  $m$  from left to right. The cell  $(i, j)$  is the cell on the intersection of row  $i$  and column  $j$  for  $1 \leq i \leq n$  and  $1 \leq j \leq m$ .

Some cells of this matrix are blocked by obstacles, while all other cells are empty. Vova occupies one of the empty cells. It is guaranteed that cells  $(1, 1), (1, m), (n, 1), (n, m)$  (that is, corners of the matrix) are blocked.

Vova can move from one empty cell to another empty cell if they share a side. Vova can escape the matrix from any empty cell on the boundary of the matrix; these cells are called exits.

Vova defines the type of the matrix based on the number of exits he can use to escape the matrix:

The 1-st type: matrices with no exits he can use to escape.

The 2-nd type: matrices with exactly one exit he can use to escape.

The 3-rd type: matrices with multiple (two or more) exits he can use to escape.

Before Vova starts moving, Misha can create more obstacles to block more cells. However, he cannot change the type of the matrix. What is the maximum number of cells Misha can block, so that the type of the matrix remains the same? Misha cannot block the cell Vova is currently standing on.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int dx[] = {-1, 1, 0, 0};
5 constexpr int dy[] = {0, 0, -1, 1};
6 void solve() {
7     int n, m;
8     cin >> n >> m;
9     vector<string> s(n);
10    int vx = 0, vy = 0;
11    int empty = 0;
12    for (int i = 0; i < n; i++) {
13        cin >> s[i];
14        for (int j = 0; j < m; j++) {
15            if (s[i][j] == 'V') {
16                vx = i, vy = j;
17            }
18            if (s[i][j] == '.') {
19                empty += 1;
20            }
21        }
22    }
23    int ans = empty;
24    vector<vector<int>> dis(n, vector(m, -1));
25    queue<pair<int, int>> q;
26    q.emplace(vx, vy);
27    dis[vx][vy] = 0;
28    int cnt = 0;
29    while (!q.empty()) {
30        auto [x, y] = q.front();
31        q.pop();
32        if (x == 0 || y == 0 || x == n - 1 || y == m - 1) {
33            cnt += 1;
34        }
35        for (int k = 0; k < 4; k++) {
36            int nx = x + dx[k];
37            int ny = y + dy[k];
38            if (0 <= nx && nx < n && 0 <= ny && ny < m && dis[nx][ny] == -1 && s[nx][ny]
39            != '#') {
40                dis[nx][ny] = dis[x][y] + 1;
41                q.emplace(nx, ny);
42            }
43        }
44        if (cnt == 0) {
45            ans = 0;
46        } else if (cnt == 1) {
47            for (int x = 0; x < n; x++) {
48                for (int y = 0; y < m; y++) {
49                    if ((x == 0 || y == 0 || x == n - 1 || y == m - 1) && dis[x][y] != -1) {
50                        ans = min(ans, dis[x][y]);
51                    }
52                }
53            }
54        }
55    }
56}
```

```

54     } else {
55         queue<array<int, 4>> q;
56         vector dise(n, vector(m, array{-1, -1}));
57         vector ex(n, vector(m, array{-1, -1}));
58         for (int x = 0; x < n; x++) {
59             for (int y = 0; y < m; y++) {
60                 if ((x == 0 || y == 0 || x == n - 1 || y == m - 1) && s[x][y] != '#') {
61                     q.push({x, y, 0, x * m + y});
62                 }
63             }
64         }
65         while (!q.empty()) {
66             auto [x, y, d, e] = q.front();
67             q.pop();
68             if (dise[x][y][0] == -1) {
69                 dise[x][y][0] = d;
70                 ex[x][y][0] = e;
71             } else if (dise[x][y][1] == -1 && e != ex[x][y][0]) {
72                 dise[x][y][1] = d;
73                 ex[x][y][1] = e;
74             } else {
75                 continue;
76             }
77             for (int k = 0; k < 4; k++) {
78                 int nx = x + dx[k];
79                 int ny = y + dy[k];
80                 if (0 <= nx && nx < n && 0 <= ny && ny < m && s[nx][ny] != '#') {
81                     q.push({nx, ny, d + 1, e});
82                 }
83             }
84         }
85         for (int x = 0; x < n; x++) {
86             for (int y = 0; y < m; y++) {
87                 if (dis[x][y] != -1 && dise[x][y][1] != -1) {
88                     ans = min(ans, dis[x][y] + dise[x][y][0] + dise[x][y][1]);
89                 }
90             }
91         }
92     }
93     ans = empty - ans;
94     cout << ans << "\n";
95 }
96 int main() {
97     ios::sync_with_stdio(false);
98     cin.tie(nullptr);
99     int t;
100    cin >> t;
101    while (t--) {
102        solve();
103    }
104    return 0;
105 }

```

## 656: Removing Graph

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Alice and Bob are playing a game on a graph. They have an undirected graph without self-loops and multiple edges. All vertices of the graph have degree equal to 2. The graph may consist of several components. Note that if such graph has  $n$  vertices, it will have exactly  $n$  edges.

Alice and Bob take turn. Alice goes first. In each turn, the player can choose  $k$  ( $l \leq k \leq r; l < r$ ) vertices that form a connected subgraph and erase these vertices from the graph, including all incident edges.

The player who can't make a step loses.

For example, suppose they are playing on the given graph with given  $l = 2$  and  $r = 3$ :

A valid vertex set for Alice to choose at the first move is one of the following:

{1, 2}

{1, 3}

{2, 3}

{4, 5}

{4, 6}

{5, 6}

{1, 2, 3}

{4, 5, 6}

Then a valid vertex set for Bob to choose at the first move is one of the following:

{1, 2}

{1, 3}

{2, 3}

{1, 2, 3}

Alice can't make a move, so she loses.

You are given a graph of size  $n$  and integers  $l$  and  $r$ . Who will win if both Alice and Bob play optimally.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 int main() {
6     ios::sync_with_stdio(false);
```

```

7     cin.tie(nullptr);
8     int n, l, r;
9     cin >> n >> l >> r;
10    DSU dsu(n);
11    for (int i = 0; i < n; i++) {
12        int u, v;
13        cin >> u >> v;
14        u--, v--;
15        dsu.merge(u, v);
16    }
17    int ans = 0;
18    for (int i = 0; i < n; i++) {
19        if (dsu.find(i) == i) {
20            int s = dsu.size(i);
21            ans ^= s >= l + r ? 0 : s / l;
22        }
23    }
24    if (ans) {
25        cout << "Alice\n";
26    } else {
27        cout << "Bob\n";
28    }
29    return 0;
30 }

```

## 657: Balancing Weapons

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You've got a job in a game studio that developed an online shooter, and your first big task is to help to balance weapons. The game has  $n$  weapons: the  $i$ -th gun has an integer fire rate  $f_i$  and an integer damage per bullet  $d_i$ . The  $i$ -th gun's total firepower is equal to  $p_i = f_i \cdot d_i$ .

You have to modify the values  $d_i$  of some guns in such a way that the new values  $d_i$  will still be integers, and the firepower of all guns will become balanced. Given an integer  $k$ , the guns are said to be balanced if  $\max_{1 \leq i \leq n} p_i - \min_{1 \leq i \leq n} p_i \leq k$ .

Since gamers that play your game don't like big changes, you need to change the values  $d_i$  for the minimum possible number of guns. What is the minimum number of guns for which you have to change these values to make the guns balanced?

Note that the new values  $d_i$  must be integers greater than 0.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> f(n), d(n);
8     for (int i = 0; i < n; i++) {
9         cin >> f[i];
10    }
11    for (int i = 0; i < n; i++) {
12        cin >> d[i];
13    }
14    int ans = 0;
15    vector<pair<i64, int>> a;
16    for (int i = 0; i < n; i++) {
17        i64 p = 1LL * f[i] * d[i];
18        a.emplace_back(max(1LL, p - k), 1);
19        a.emplace_back(p + 1, -1);
20    }
21    sort(a.begin(), a.end());
22    int res = 0;
23    i64 last = 0;
24    for (auto [x, v] : a) {
25        if (res > 0) {
26            auto len = x - last;
27            int val = 0;
28            vector<pair<i64, int>> b;
29            for (int i = 0; i < n; i++) {
30                if (f[i] > k + 1) {
31                    i64 t = last - last % f[i];
32                    for (auto l : {t, t + f[i]}) {
33                        i64 L = max(last, l + 1);
34                        i64 R = min(x, l + f[i] - k);
35                        if (L < R) {
36                            b.emplace_back(L, -1);
37                            b.emplace_back(R, 1);
38                        }
39                    }
40                }
41            }
42            b.emplace_back(x, 0);
43            i64 lst = last;
44            sort(b.begin(), b.end());
45            for (auto [x, v] : b) {
46                if (x != lst && val == 0) {
47                    ans = max(ans, res);
48                }
49                val += v;
50                lst = x;
51            }
52        }
53        res += v;
54        last = x;
55    }
56    ans = n - ans;
57    cout << ans << "\n";
58 }
59 int main() {
60     ios::sync_with_stdio(false);
61     cin.tie(nullptr);
62     int t;
63     cin >> t;
64     while (t--) {
```

```

65     solve();
66 }
67 return 0;
68 }
```

### 658: Survival of the Weakest (easy version)

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the easy version of the problem. It differs from the hard one only in constraints on  $n$ . You can make hacks only if you lock both versions.

Let  $a_1, a_2, \dots, a_n$  be an array of non-negative integers. Let  $F(a_1, a_2, \dots, a_n)$  be the sorted in the non-decreasing order array of  $n - 1$  smallest numbers of the form  $a_i + a_j$ , where  $1 \leq i < j \leq n$ . In other words,  $F(a_1, a_2, \dots, a_n)$  is the sorted in the non-decreasing order array of  $n - 1$  smallest sums of all possible pairs of elements of the array  $a_1, a_2, \dots, a_n$ . For example,  $F(1, 2, 5, 7) = [1+2, 1+5, 2+5] = [3, 6, 7]$ .

You are given an array of non-negative integers  $a_1, a_2, \dots, a_n$ . Determine the single element of the array  $\underbrace{F(F(\dots F}_{n-1}(a_1, a_2, \dots, a_n) \dots))$ . Since the answer can be quite large, output it modulo  $10^9 + 7$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
```

```

22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int n;
38     cin >> n;
39     vector<int> a(n);
40     for (int i = 0; i < n; i++) {
41         cin >> a[i];
42     }
43     sort(a.begin(), a.end());
44     int t = n - 1;
45     Z ans = 0;
46     for (int i = 0; i < t; i++) {
47         n = a.size();
48         ans *= 2;
49         priority_queue<pair<int, int>> h;
50         vector<int> f(n);
51         for (int j = 0; j < n; j++) {
52             f[j] = j + 1;
53             if (f[j] < n) {
54                 h.emplace(-a[j] - a[f[j]], j);
55             }
56         }
57         vector<int> b;
58         while (b.size() < 64 && !h.empty()) {
59             auto [s, j] = h.top();
60             h.pop();
61             b.push_back(-s);
62             f[j] += 1;
63             if (f[j] < n) {
64                 h.emplace(-a[j] - a[f[j]], j);
65             }
66         }
67         ans += b[0];
68         int v = b[0];
69         for (auto &x : b) {
70             x -= v;
71         }
72         a = b;
73     }
74     cout << ans << "\n";
75     return 0;
76 }

```

## 659: Routing

- Time limit: 2 seconds
- Memory limit: 512 megabytes

- Input file: standard input
- Output file: standard output

Ada operates a network that consists of  $n$  servers and  $m$  direct connections between them. Each direct connection between a pair of distinct servers allows bidirectional transmission of information between these two servers. Ada knows that these  $m$  direct connections allow to directly or indirectly transmit information between any two servers in this network. We say that server  $v$  is a neighbor of server  $u$  if there exists a direct connection between these two servers.

Ada needs to configure her network's WRP (Weird Routing Protocol). For each server  $u$  she needs to select exactly one of its neighbors as an auxiliary server  $a(u)$ . After all  $a(u)$  are set, routing works as follows. Suppose server  $u$  wants to find a path to server  $v$  (different from  $u$ ).

Server  $u$  checks all of its direct connections to other servers. If it sees a direct connection with server  $v$ , it knows the path and the process terminates.

If the path was not found at the first step, server  $u$  asks its auxiliary server  $a(u)$  to find the path.

Auxiliary server  $a(u)$  follows this process starting from the first step.

After  $a(u)$  finds the path, it returns it to  $u$ . Then server  $u$  constructs the resulting path as the direct connection between  $u$  and  $a(u)$  plus the path from  $a(u)$  to  $v$ .

As you can see, this procedure either produces a correct path and finishes or keeps running forever. Thus, it is critically important for Ada to configure her network's WRP properly.

Your goal is to assign an auxiliary server  $a(u)$  for each server  $u$  in the given network. Your assignment should make it possible to construct a path from any server  $u$  to any other server  $v$  using the aforementioned procedure. Or you should report that such an assignment doesn't exist.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<int> adj(n);
10    for (int i = 0; i < m; i++) {
11        int u, v;
12        cin >> u >> v;
13        u--, v--;
14        adj[u] |= 1 << v;
15        adj[v] |= 1 << u;
16    }
17    vector<int> dp(1 << n);
18    for (int i = 0; i < n; i++) {

```

```

19         dp[1 << i] |= 1 << i;
20     }
21     for (int s = 1; s < (1 << n); s++) {
22         int sadj = 0;
23         for (int i = 0; i < n; i++) {
24             if (s >> i & 1) {
25                 sadj |= adj[i];
26             }
27         }
28         int lo = __builtin_ctz(s);
29         if (sadj == (1 << n) - 1 && (adj[lo] & dp[s])) {
30             int nxt = lo;
31             vector<int> cyc;
32             int S = s;
33             while (S != (1 << lo)) {
34                 int t = __builtin_ctz(dp[S] & adj[nxt]);
35                 cyc.push_back(t);
36                 S ^= 1 << t;
37                 nxt = t;
38             }
39             cyc.push_back(lo);
40             cout << "Yes\n";
41             vector<int> a(n, -1);
42             queue<int> q;
43             for (int i = 0; i < cyc.size(); i++) {
44                 a[cyc[i]] = cyc[(i + 1) % cyc.size()];
45                 q.push(cyc[i]);
46             }
47             while (!q.empty()) {
48                 int x = q.front();
49                 q.pop();
50                 for (int y = 0; y < n; y++) {
51                     if (~adj[x] >> y & 1) {
52                         continue;
53                     }
54                     if (a[y] == -1) {
55                         a[y] = x;
56                         q.push(y);
57                     }
58                 }
59             }
60             for (int i = 0; i < n; i++) {
61                 cout << a[i] + 1 << " \n"[i == n - 1];
62             }
63             return 0;
64         }
65         int mask = 0;
66         for (int i = 0; i < n; i++) {
67             if (dp[s] >> i & 1) {
68                 mask |= adj[i];
69             }
70         }
71         for (int i = lo; i < n; i++) {
72             if (((~s & mask) >> i & 1) {
73                 dp[s | 1 << i] |= 1 << i;
74             }
75         }
76     }
77     cout << "No\n";
78     return 0;
79 }
```

## 660: Serval and Music Game

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Serval loves playing music games. He meets a problem when playing music games, and he leaves it for you to solve.

You are given  $n$  positive integers  $s_1 < s_2 < \dots < s_n$ .  $f(x)$  is defined as the number of  $i$  ( $1 \leq i \leq n$ ) that exist non-negative integers  $p_i, q_i$  such that:

$$s_i = p_i \left\lfloor \frac{s_n}{x} \right\rfloor + q_i \left\lceil \frac{s_n}{x} \right\rceil$$

Find out  $\sum_{x=1}^{s_n} x \cdot f(x)$  modulo 998 244 353.

As a reminder,  $\lfloor x \rfloor$  denotes the maximal integer that is no greater than  $x$ , and  $\lceil x \rceil$  denotes the minimal integer that is no less than  $x$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 template<int P>
15 # struct MInt;
16 template<int V, int P>
17 constexpr MInt<P> CInv = MInt<P>(V).inv();
18 constexpr int P = 998244353;
19 using Z = MInt<P>;
20 void solve() {
21     int n;
22     cin >> n;
23     vector<int> s(n);
24     for (int i = 0; i < n; i++) {
25         cin >> s[i];
26     }
27     vector<int> sum(s[n - 1] + 1), ex(s[n - 1] + 1);
28     for (int i = 0; i < n; i++) {
29         ex[s[i]] = 1;
30     }

```

```

31     for (int i = 1; i <= s[n - 1]; i++) {
32         sum[i] = sum[i - 1] + ex[i];
33     }
34     Z ans = 0;
35     for (int x = 1; x <= s[n - 1]; x++) {
36         int a = s[n - 1] / x, b = (s[n - 1] + x - 1) / x;
37         Z res = 0;
38         if (a == b) {
39             for (int i = a; i <= s[n - 1]; i += a) {
40                 res += ex[i];
41             }
42         } else {
43             for (int i = 1; i * a <= s[n - 1]; i++) {
44                 if (i >= a) {
45                     res += sum[s[n - 1]] - sum[i * a - 1];
46                     break;
47                 } else {
48                     res += sum[min(s[n - 1], i * a + i)] - sum[i * a - 1];
49                 }
50             }
51         }
52         ans += res * x;
53     }
54     cout << ans << "\n";
55 }
56 int main() {
57     ios::sync_with_stdio(false);
58     cin.tie(nullptr);
59     int t;
60     cin >> t;
61     while (t--) {
62         solve();
63     }
64     return 0;
65 }
```

## 661: Magician and Pigs (Easy Version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the easy version of the problem. The only difference between the two versions is the constraint on  $n$  and  $x$ . You can make hacks only if both versions of the problem are solved.

Little09 has been interested in magic for a long time, and it's so lucky that he meets a magician! The magician will perform  $n$  operations, each of them is one of the following three:

1  $x$ : Create a pig with  $x$  Health Points.

2  $x$ : Reduce the Health Point of all living pigs by  $x$ .

3: Repeat all previous operations. Formally, assuming that this is the  $i$ -th operation in the operation sequence, perform the first  $i - 1$  operations (including “Repeat” operations involved) in turn.

A pig will die when its Health Point is less than or equal to 0.

Little09 wants to know how many living pigs there are after all the operations. Please, print the answer modulo 998 244 353.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 template<int P>
15 # struct MInt;
16 template<int V, int P>
17 constexpr MInt<P> CInv = MInt<P>(V).inv();
18 constexpr int P = 998244353;
19 using Z = MInt<P>;
20 constexpr int inf = 1E9;
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     int n;
25     cin >> n;
26     vector<array<int, 2>> op(n);
27     for (int i = 0; i < n; i++) {
28         int t;
29         cin >> t;
30         if (t == 1) {
31             int x;
32             cin >> x;
33             op[i] = {t, x};
34         } else if (t == 2) {
35             int x;
36             cin >> x;
37             op[i] = {t, x};
38         } else {
39             op[i] = {t};
40         }
41     }
42     vector<int> pre(n + 1);
43     for (int i = 0; i < n; i++) {
44         if (op[i][0] == 2) {
45             pre[i + 1] = min(inf, pre[i] + op[i][1]);
46         } else if (op[i][0] == 3) {
47             pre[i + 1] = min(inf, pre[i] * 2);
48         } else {
49             pre[i + 1] = pre[i];
50         }
51     }
52     int c0 = 0;
53     vector<int> a;
54     i64 sum = 0;

```

```

55     for (int i = 0; i < n; i++) {
56         if (op[i][0] == 3) {
57             if (pre[i] == 0) {
58                 c0++;
59             } else {
60                 a.push_back(i);
61             }
62         } else if (op[i][0] == 2) {
63             sum += op[i][1];
64         }
65     }
66     Z ans = 0;
67     for (int i = 0; i < n; i++) {
68         if (op[i][0] == 3) {
69             c0 -= (pre[i] == 0);
70         } else if (op[i][0] == 2) {
71             sum -= op[i][1];
72         } else {
73             int x = op[i][1];
74             if (sum >= x) {
75                 continue;
76             }
77             x -= sum;
78             int j = lower_bound(a.begin(), a.end(), x, [&](int i, int x) {
79                 return pre[i] < x;
80             }) - a.begin() - 1;
81             Z res = 0;
82             int v = x;
83             while (j >= 0 && a[j] > i) {
84                 res *= 2;
85                 if (pre[a[j]] < v) {
86                     res += 1;
87                     v -= pre[a[j]];
88                 }
89                 j--;
90             }
91             res += 1;
92             res *= power(Z(2), c0);
93             ans += res;
94         }
95     }
96     cout << ans << "\n";
97     return 0;
98 }
```

## 662: Rebrending

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Kostya and Zhenya - the creators of the band “Paper” - after the release of the legendary album decided to create a new band “Day movers”, for this they need to find two new people.

They invited  $n$  people to the casting. The casting will last  $q$  days. On the  $i$ th of the days, Kostya and Zhenya want to find two people on the segment from  $l_i$  to  $r_i$  who are most suitable for their band.

Since “Day movers” are doing a modern art, musical skills are not important to them and they look only at other signs: they want the height difference between two people to be as small as possible.

Help them, and for each day, find the minimum difference in the growth of people from the casting on this segment!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template <typename T>
5 # struct Fenwick;
6 constexpr int inf = 1E9;
7 # struct Min;
8 int main() {
9     ios::sync_with_stdio(false);
10    cin.tie(nullptr);
11    int n, q;
12    cin >> n >> q;
13    vector<int> a(n);
14    for (int i = 0; i < n; i++) {
15        cin >> a[i];
16        a[i]--;
17    }
18    int lg = __lg(n) + 1;
19    vector p(lg + 1, vector<vector<int>>(n));
20    vector<vector<pair<int, int>>> qry(n);
21    for (int i = 0; i < q; i++) {
22        int l, r;
23        cin >> l >> r;
24        l--;
25        qry[l].emplace_back(r, i);
26    }
27    Fenwick<Min> fen(n);
28    vector<int> ans(q);
29    for (int l = n - 1; l >= 0; l--) {
30        for (int t = lg; t >= 0; t--) {
31            int x = a[l] >> t;
32            for (auto y : {x - 1, x, x + 1}) {
33                if (0 <= y && y < n && !p[t][y].empty()) {
34                    int s = 0;
35                    for (int i = 0; i < p[t][y].size(); i++) {
36                        int j = p[t][y][i];
37                        int d = abs(a[l] - a[j]);
38                        fen.add(j, d);
39                        if (d * 2 <= (1 << t)) {
40                            s = i + 1;
41                        }
42                    }
43                    p[t][y].erase(p[t][y].begin(), p[t][y].begin() + s);
44                }
45            }
46            p[t][x].push_back(l);
47        }
48        for (auto [r, i] : qry[l]) {
49            ans[i] = fen.sum(r).x;
50        }
    }
}

```

```

51     }
52     for (int i = 0; i < q; i++) {
53         cout << ans[i] << "\n";
54     }
55     return 0;
56 }
```

### 663: Project Manager

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  employees at Bersoft company, numbered from 1 to  $n$ . Each employee works on some days of the week and rests on the other days. You are given the lists of working days of the week for each employee.

There are regular days and holidays. On regular days, only those employees work that have the current day of the week on their list. On holidays, no one works. You are provided with a list of days that are holidays. The days are numbered from 1 onwards, day 1 is Monday.

The company receives  $k$  project offers they have to complete. The projects are numbered from 1 to  $k$  in the order of decreasing priority.

Each project consists of multiple parts, where the  $i$ -th part must be completed by the  $a_i$ -th employee. The parts must be completed in order (i. e. the  $(i + 1)$ -st part can only be started when the  $i$ -th part is completed). Each part takes the corresponding employee a day to complete.

The projects can be worked on simultaneously. However, one employee can complete a part of only one project during a single day. If they have a choice of what project to complete a part on, they always go for the project with the highest priority (the lowest index).

For each project, output the day that project will be completed on.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m, k;
8     cin >> n >> m >> k;
9     map<string, int> daynum;
10    daynum["Monday"] = 0;
```

```
11     daynum["Tuesday"] = 1;
12     daynum["Wednesday"] = 2;
13     daynum["Thursday"] = 3;
14     daynum["Friday"] = 4;
15     daynum["Saturday"] = 5;
16     daynum["Sunday"] = 6;
17     vector<array<bool, 7>> work(n);
18     for (int i = 0; i < n; i++) {
19         int t;
20         cin >> t;
21         for (int j = 0; j < t; j++) {
22             string s;
23             cin >> s;
24             work[i][daynum[s]] = true;
25         }
26     }
27     vector<int> h(m);
28     for (int i = 0; i < m; i++) {
29         cin >> h[i];
30         h[i]--;
31     }
32     int curh = 0;
33     vector<vector<int>> a(k);
34     vector<priority_queue<int, vector<int>, greater<>>> works(n);
35     vector<int> lastwork(n, -1);
36     vector<int> havework[7];
37     for (int i = 0; i < k; i++) {
38         int p;
39         cin >> p;
40         a[i].resize(p);
41         for (int j = 0; j < p; j++) {
42             cin >> a[i][j];
43             a[i][j]--;
44         }
45         works[a[i][0]].push(i);
46         for (int j = 0; j < 7; j++) {
47             if (work[a[i][0]][j]) {
48                 havework[j].push_back(a[i][0]);
49             }
50         }
51     }
52     vector<int> step(k);
53     vector<int> ans(k);
54     int complete = 0;
55     for (int t = 0; complete < k; t++) {
56         if (curh < m && h[curh] == t) {
57             curh++;
58             continue;
59         }
60         vector<int> done, nhave;
61         for (auto i : havework[t % 7]) {
62             if (lastwork[i] == t) {
63                 continue;
64             }
65             lastwork[i] = t;
66             if (works[i].empty()) {
67                 continue;
68             }
69             int x = works[i].top();
70             works[i].pop();
71             done.push_back(x);
72             nhave.push_back(i);
73         }
74         havework[t % 7] = nhave;
```

```

75         for (auto x : done) {
76             step[x]++;
77             if (step[x] < a[x].size()) {
78                 int i = a[x][step[x]];
79                 works[i].push(x);
80                 for (int j = 0; j < 7; j++) {
81                     if (work[i][j]) {
82                         havework[j].push_back(i);
83                     }
84                 }
85             } else {
86                 ans[x] = t + 1;
87                 complete++;
88             }
89         }
90     }
91     for (int i = 0; i < k; i++) {
92         cout << ans[i] << " \n"[i == k - 1];
93     }
94 }
95 }
```

## 664: Josuke and Complete Graph

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Josuke received a huge undirected weighted complete<sup>†</sup> graph  $G$  as a gift from his grandfather. The graph contains  $10^{18}$  vertices. The peculiarity of the gift is that the weight of the edge between the different vertices  $u$  and  $v$  is equal to  $\gcd(u, v)$ <sup>‡</sup>. Josuke decided to experiment and make a new graph  $G'$ . To do this, he chooses two integers  $l \leq r$  and deletes all vertices except such vertices  $v$  that  $l \leq v \leq r$ , and also deletes all the edges except between the remaining vertices.

Now Josuke is wondering how many different weights are there in  $G'$ . Since their count turned out to be huge, he asks for your help.

<sup>†</sup> A complete graph is a simple undirected graph in which every pair of distinct vertices is adjacent.

<sup>‡</sup>  $\gcd(x, y)$  denotes the greatest common divisor (GCD) of the numbers  $x$  and  $y$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     i64 l, r;
6     cin >> l >> r;
7     l--;
```

```

8     if (r >= 3 * l) {
9         cout << r / 2 << "\n";
10        return;
11    }
12    i64 ans = 0;
13    for (i64 i = 1, j; i <= r; i = j + 1) {
14        j = r / (r / i);
15        if (i <= l) j = min(j, l / (l / i));
16        if (r / i >= l / i + 2) ans += j - i + 1;
17    }
18    cout << ans << "\n";
19 }
20 int main() {
21     ios::sync_with_stdio(false);
22     cin.tie(nullptr);
23     int t;
24     cin >> t;
25     while (t--) {
26         solve();
27     }
28     return 0;
29 }
```

## 665: Function Sum

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Suppose you have an integer array  $a_1, a_2, \dots, a_n$ .

Let  $\text{lsl}(i)$  be the number of indices  $j$  ( $1 \leq j < i$ ) such that  $a_j < a_i$ .

Analogically, let  $\text{grr}(i)$  be the number of indices  $j$  ( $i < j \leq n$ ) such that  $a_j > a_i$ .

Let's name position  $i$  good in the array  $a$  if  $\text{lsl}(i) < \text{grr}(i)$ .

Finally, let's define a function  $f$  on array  $a$   $f(a)$  as the sum of all  $a_i$  such that  $i$  is good in  $a$ .

Given two integers  $n$  and  $k$ , find the sum of  $f(a)$  over all arrays  $a$  of size  $n$  such that  $1 \leq a_i \leq k$  for all  $1 \leq i \leq n$  modulo 998 244 353.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 998244353;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
8     if (x < 0) {
```

```

9         x += P;
10    }
11    if (x >= P) {
12        x -= P;
13    }
14    return x;
15 }
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25 }
26 # struct Z;
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(nullptr);
30     int n, k;
31     cin >> n >> k;
32     vector<Z> binom(n + 1, vector<Z>(n + 1));
33     for (int i = 0; i <= n; i++) {
34         binom[i][0] = 1;
35         for (int j = 1; j <= i; j++) {
36             binom[i][j] = binom[i - 1][j] + binom[i - 1][j - 1];
37         }
38     }
39     vector<Z> f(n + 2);
40     for (int k = 1; k <= n + 1; k++) {
41         for (int i = 0; i < n; i++) {
42             for (int v = 1; v <= k; v++) {
43                 vector<Z> a(n), b(n);
44                 for (int j = 0; j <= i; j++) {
45                     a[j] = binom[i][j] * power(Z(v - 1), j) * power(Z(k - v + 1), i - j);
46                 }
47                 for (int j = 0; j <= n - 1 - i; j++) {
48                     b[j] = binom[n - 1 - i][j] * power(Z(k - v), j) * power(Z(v), n - 1 - i - j);
49                 }
50                 Z s = 0;
51                 for (int j = 0; j < n; j++) {
52                     f[k] += s * b[j] * v;
53                     s += a[j];
54                 }
55             }
56         }
57     }
58     Z ans = 0;
59     for (int i = 0; i <= n + 1; i++) {
60         Z v = 1;
61         for (int j = 0; j <= n + 1; j++) {
62             if (i != j) {
63                 v *= k - j;
64                 v /= i - j;
65             }
66         }
67         ans += v * f[i];
68     }
69     cout << ans << "\n";
70     return 0;
71 }

```

## 666: Hossam and a Letter

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Hossam bought a new piece of ground with length  $n$  and width  $m$ , he divided it into an  $n \cdot m$  grid, each cell being of size  $1 \times 1$ .

Since Hossam's name starts with the letter 'H', he decided to draw the capital letter 'H' by building walls of size  $1 \times 1$  on some squares of the ground. Each square  $1 \times 1$  on the ground is assigned a quality degree: perfect, medium, or bad.

The process of building walls to form up letter 'H' has the following constraints:

The letter must consist of one horizontal and two vertical lines.

The vertical lines must not be in the same or neighboring columns.

The vertical lines must start in the same row and end in the same row (and thus have the same length).

The horizontal line should connect the vertical lines, but must not cross them.

The horizontal line can be in any row between the vertical lines (not only in the middle), except the top and the bottom one. (With the horizontal line in the top row the letter looks like 'n', and in the bottom row like 'U').

It is forbidden to build walls in cells of bad quality.

You can use at most one square of medium quality.

You can use any number of squares of perfect quality.

Find the maximum number of walls that can be used to draw the letter 'H'.

Check the note for more clarification.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<string> s(n);
10    for (int i = 0; i < n; i++) {

```

```

11         cin >> s[i];
12     }
13     vector up(n, vector<int>(m, -1)), upm(up);
14     vector dn(n, vector<int>(m, n)), dnm(dn);
15     for (int i = 1; i < n; i++) {
16         up[i] = up[i - 1];
17         upm[i] = upm[i - 1];
18         for (int j = 0; j < m; j++) {
19             if (s[i - 1][j] == '#') {
20                 up[i][j] = upm[i][j] = i - 1;
21             }
22             if (s[i - 1][j] == 'm') {
23                 upm[i][j] = i - 1;
24                 up[i][j] = max(up[i][j], upm[i - 1][j]);
25             }
26         }
27     }
28     for (int i = n - 2; i >= 0; i--) {
29         dn[i] = dn[i + 1];
30         dnm[i] = dnm[i + 1];
31         for (int j = 0; j < m; j++) {
32             if (s[i + 1][j] == '#') {
33                 dn[i][j] = dnm[i][j] = i + 1;
34             }
35             if (s[i + 1][j] == 'm') {
36                 dnm[i][j] = i + 1;
37                 dn[i][j] = min(dn[i][j], dnm[i + 1][j]);
38             }
39         }
40     }
41     int ans = 0;
42     auto upd = [&](int len, int *a) {
43         int u = min(a[0], a[1]) - 1;
44         int v = min(a[2], a[3]) - 1;
45         if (u && v) {
46             ans = max(ans, len + 2 * (u + v));
47         }
48     };
49     for (int i = 0; i < n; i++) {
50         for (int l = 0; l < m; l++) {
51             int cntm = 0;
52             for (int r = l; r < m; r++) {
53                 if (s[i][r] == '#') {
54                     break;
55                 }
56                 if (s[i][r] == 'm') {
57                     cntm++;
58                 }
59                 if (cntm > 1) {
60                     break;
61                 }
62                 if (r - l < 2) {
63                     continue;
64                 }
65                 int a[4] = {i - upm[i][l], i - upm[i][r], dnm[i][l] - i, dnm[i][r] - i};
66                 int b[4] = {i - up[i][l], i - up[i][r], dn[i][l] - i, dn[i][r] - i};
67                 upd(r - l + 1, a);
68                 if (cntm == 0) {
69                     for (int j = 0; j < 4; j++) {
70                         swap(a[j], b[j]);
71                         upd(r - l + 1, a);
72                         swap(a[j], b[j]);
73                     }
74                 }
75             }
76         }
77     }

```

```

75         }
76     }
77 }
78 cout << ans << "\n";
79 return 0;
80 }
```

### 667: Balance (Hard version)

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. The only difference is that in this version there are remove queries.

Initially you have a set containing one element - 0. You need to handle  $q$  queries of the following types:

- $x$  - add the integer  $x$  to the set. It is guaranteed that this integer is not contained in the set;
- $x$  - remove the integer  $x$  from the set. It is guaranteed that this integer is contained in the set;
- ?  $k$  - find the  $k$ -mex of the set.

In our problem, we define the  $k$ -mex of a set of integers as the smallest non-negative integer  $x$  that is divisible by  $k$  and which is not contained in the set.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int q;
8     cin >> q;
9     map f;
10    map>> g, d;
11    set s{0};
12    for (int i = 0; i < q; i++) {
13        char o;
14        i64 x;
15        cin >> o >> x;
16        if (o == '+') {
17            s.insert(x);
18            for (auto y : d[x]) {
19                g[y].erase(x);
20            }
21        }
22    }
23    for (int i = 0; i < q; i++) {
24        int k;
25        cin >> k;
26        cout << *lower_bound(s.begin(), s.end(), k) << '\n';
27    }
28}
```

```

21         } else if (o == '+') {
22             s.erase(x);
23             for (auto y : d[x]) {
24                 g[y].insert(x);
25             }
26         } else {
27             if (g[x].empty()) {
28                 while (s.contains(f[x])) {
29                     f[x] += x;
30                     d[f[x]].insert(x);
31                 }
32                 cout << f[x] << "\n";
33             } else {
34                 cout << *g[x].begin() << "\n";
35             }
36         }
37     }
38     return 0;
39 }
```

## constructive algorithms

### 668: Interactive Game with Coloring

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

This is an interactive problem. Remember to flush your output while communicating with the testing program. You may use fflush(stdout) in C++, system.out.flush() in Java, stdout.flush() in Python or flush(output) in Pascal to flush the output. If you use some other programming language, consult its documentation. You may also refer to the guide on interactive problems: <https://codeforces.com/blog/entry/45307>.

You are given a tree on  $n$  vertices; vertex 1 is the root of the tree. For every  $i \in [2, n]$ , the parent of the  $i$ -th vertex is  $p_i$ , and  $p_i < i$ .

You have to color all edges of the tree using the minimum possible number of colors such that you can win the game on that tree (every edge should be painted into exactly one color).

The game we're going to play will be conducted as follows. After you paint the edges and print their colors, the jury will place a chip into one of the vertices of the tree (except for the root). Your goal is to move this chip to the root in exactly  $d$  moves, where  $d$  is the distance from the root to that vertex (the distance is equal to the number of edges on the path). If the chip reaches the root in  $d$  moves, you win. Otherwise, you lose.

The jury won't tell you where the chip is located. You won't even know the value of  $d$  in advance.

However, at the start of each move, you will be told how many edges of each color are incident to the current vertex (this includes both the edge leading up the tree and the edges leading away from the root). You have to choose one of these colors, and the chip will be moved along the edge of the chosen color (if there are multiple edges with that color incident to the current vertex, the jury gets to choose one of them). After the chip is moved, you will be told the same information about the current vertex again, and the game continues, until you either reach the root, or you make  $d$  moves without reaching the root.

The interactor for this problem is adaptive. It means that both the starting vertex and the current vertex are not fixed and may change “on the run” depending on the output of your program. However, the state of the game will always be consistent with the information you are given: there will always be at least one starting vertex and at least one path of your chip from that vertex consistent with both the information about the colors you receive and the colors you’ve chosen during the moves.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> p(n), dep(n), deg(n);
10    p[0] = -1;
11    for (int i = 1; i < n; i++) {
12        cin >> p[i];
13        p[i]--;
14        dep[i] = dep[p[i]] + 1;
15        deg[p[i]]++;
16    }
17    int k;
18    vector<int> col(n);
19    if (*max_element(dep.begin(), dep.end()) == 1) {
20        k = 1;
21    } else {
22        vector<vector<int>> adj(n);
23        for (int i = 1; i < n; i++) {
24            adj[p[i]].push_back(i);
25        }
26        bool ok = true;
27        for (int i = 1; i < n; i++) {
28            if (p[i] == 0) {
29                int cnt[2] {};
30                auto dfs = [&](auto self, int x) -> void {
31                    if (deg[x] == 1) {
32                        cnt[dep[x] % 2]++;
33                    }
34                    for (auto y : adj[x]) {
35                        col[y] = col[x] ^ 1;
36                        self(self, y);
37                    }
38                };
39                dfs(dfs, i);
40            }
41        }
42    }
43}
```

```

37             }
38         };
39         dfs(dfs, i);
40         if (cnt[0] && cnt[1]) {
41             ok = false;
42             break;
43         } else {
44             col[i] = cnt[0] ? 1 : 0;
45             dfs(dfs, i);
46         }
47     }
48 }
49 if (!ok) {
50     k = 3;
51     for (int i = 1; i < n; i++) {
52         col[i] = dep[i] % k;
53     }
54 } else {
55     k = 2;
56 }
57 }
58 cout << k << "\n";
59 for (int i = 1; i < n; i++) {
60     cout << col[i] + 1 << " \n"[i == n - 1];
61 }
62 cout.flush();
63 while (true) {
64     int x;
65     cin >> x;
66     if (x == 1) {
67         break;
68     }
69     vector<int> a(k);
70     vector<int> u;
71     for (int i = 0; i < k; i++) {
72         cin >> a[i];
73         if (a[i] == 1) {
74             u.push_back(i);
75         }
76     }
77     if (u.size() > 1) {
78         if (k == 2) {
79             u[0] = 0;
80         } else {
81             if ((u[0] + 1) % 3 != u[1]) {
82                 swap(u[0], u[1]);
83             }
84         }
85     }
86     cout << u[0] + 1 << endl;
87 }
88 return 0;
89 }

```

### 669: Great Grids

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

An  $n \times m$  grid of characters is called great if it satisfies these three conditions:

Each character is either ‘A’, ‘B’, or ‘C’.

Every  $2 \times 2$  contiguous subgrid contains all three different letters.

Any two cells that share a common edge contain different letters.

Let  $(x, y)$  denote the cell in the  $x$ -th row from the top and  $y$ -th column from the left.

You want to construct a great grid that satisfies  $k$  constraints. Each constraint consists of two cells,  $(x_{i,1}, y_{i,1})$  and  $(x_{i,2}, y_{i,2})$ , that share exactly one corner. You want your great grid to have the same letter in cells  $(x_{i,1}, y_{i,1})$  and  $(x_{i,2}, y_{i,2})$ .

Determine whether there exists a great grid satisfying all the constraints.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m, k;
6     cin >> n >> m >> k;
7     vector<vector<pair<int, int>>> adj(n + m);
8     for (int i = 0; i < k; i++) {
9         int x1, y1, x2, y2;
10        cin >> x1 >> y1 >> x2 >> y2;
11        x1--, y1--, x2--, y2--;
12        if (y2 == y1 + 1) {
13            adj[x2].emplace_back(n + y2, 1);
14            adj[n + y2].emplace_back(x2, 1);
15        } else {
16            adj[x2].emplace_back(n + y1, 0);
17            adj[n + y1].emplace_back(x2, 0);
18        }
19    }
20    vector<int> col(n + m, -1);
21    for (int i = 0; i < n + m; i++) {
22        if (col[i] == -1) {
23            col[i] = 0;
24            queue<int> q;
25            q.push(i);
26            while (!q.empty()) {
27                auto x = q.front();
28                q.pop();
29                for (auto [y, w] : adj[x]) {
30                    if (col[y] == (w ^ col[x] ^ 1)) {
31                        cout << "NO\n";
32                        return;
33                    }
34                    if (col[y] == -1) {
35                        col[y] = w ^ col[x];
36                        q.push(y);
37                    }
38                }
39            }
40        }
41    }
42 }
```

```

39         }
40     }
41   cout << "YES\n";
42 }
43 int main() {
44   ios::sync_with_stdio(false);
45   cin.tie(nullptr);
46   int t;
47   cin >> t;
48   while (t--) {
49     solve();
50   }
51   return 0;
52 }
```

### 670: In Search of Truth (Hard Version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The only difference between easy and hard versions is the maximum number of queries. In this version, you are allowed to ask at most 1000 queries.

This is an interactive problem.

You are playing a game. The circle is divided into  $n$  sectors, sectors are numbered from 1 to  $n$  in some order. You are in the adjacent room and do not know either the number of sectors or their numbers. There is also an arrow that initially points to some sector. Initially, the host tells you the number of the sector to which the arrow points. After that, you can ask the host to move the arrow  $k$  sectors counterclockwise or clockwise at most 1000 times. And each time you are told the number of the sector to which the arrow points.

Your task is to determine the integer  $n$  - the number of sectors in at most 1000 queries.

It is guaranteed that  $1 \leq n \leq 10^6$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int B = 400;
5 int main() {
6   ios::sync_with_stdio(false);
```

```

7     cin.tie(nullptr);
8     int x;
9     cin >> x;
10    int N = x;
11    mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
12    for (int i = 0; i < B; i++) {
13        cout << "+" << rng() % 1000000000 + 1 << endl;
14        cin >> x;
15        N = max(N, x);
16    }
17    int C = (1000 - B) / 2;
18    map<int, int> f;
19    f[x] = 0;
20    for (int i = 1; i < C; i++) {
21        cout << "- " << i << endl;
22        cin >> x;
23        if (!f.count(x)) {
24            f[x] = i;
25        }
26    }
27    cout << "+" << C - 1 + N << endl;
28    cin >> x;
29    int n = N;
30    while (!f.count(x)) {
31        cout << "+" << C << endl;
32        cin >> x;
33        n += C;
34    }
35    n += f[x];
36    cout << "! " << n << endl;
37    return 0;
38 }

```

## 671: Aztec Catacombs

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Indiana Jones found ancient Aztec catacombs containing a golden idol. The catacombs consists of  $n$  caves. Each pair of caves is connected with a two-way corridor that can be opened or closed. The entrance to the catacombs is in the cave 1, the idol and the exit are in the cave  $n$ .

When Indiana goes from a cave  $x$  to a cave  $y$  using an open corridor, all corridors connected to the cave  $x$  change their state: all open corridors become closed, all closed corridors become open. Indiana wants to go from cave 1 to cave  $n$  going through as small number of corridors as possible. Help him find the optimal path, or determine that it is impossible to get out of catacombs.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<vector<int>> adj(n);
10    vector<bool> f(n);
11    for (int i = 0; i < m; i++) {
12        int u, v;
13        cin >> u >> v;
14        u--, v--;
15        adj[u].push_back(v);
16        adj[v].push_back(u);
17    }
18    vector<int> dis(n, -1);
19    vector<int> pre(n);
20    queue<int> q;
21    dis[0] = 0;
22    pre[0] = -1;
23    q.push(0);
24    for (auto x : adj[0]) {
25        f[x] = true;
26    }
27    while (!q.empty()) {
28        int x = q.front();
29        q.pop();
30        for (auto y : adj[x]) {
31            if (dis[y] == -1) {
32                dis[y] = dis[x] + 1;
33                pre[y] = x;
34                q.push(y);
35            }
36        }
37    }
38    vector<int> ans;
39    if (dis[n - 1] != -1) {
40        for (int i = n - 1; i != -1; i = pre[i]) {
41            ans.push_back(i);
42        }
43        reverse(ans.begin(), ans.end());
44    }
45    int x = -1;
46    for (int i = 1; i < n; i++) {
47        if (!f[i] && dis[i] != -1 && (x == -1 || dis[x] > dis[i])) {
48            x = i;
49        }
50    }
51    if (x != -1 && (ans.empty() || ans.size() - 1 > dis[x] + 2)) {
52        ans.clear();
53        for (int i = x; i != -1; i = pre[i]) {
54            ans.push_back(i);
55        }
56        reverse(ans.begin(), ans.end());
57        ans.push_back(0);
58        ans.push_back(n - 1);
59    }
60    // vector<int> o(n), pos(n);
61    // iota(o.begin(), o.end(), 0);
62    // sort(o.begin(), o.end(), [&](int i, int j) {
63    //     return adj[i].size() < adj[j].size();
64    // });
65}
```

```

65      // for (int i = 0; i < n; i++) {
66      //     pos[o[i]] = i;
67      //
68      vector<vector<int>> e(n);
69      for (int i = 0; i < n; i++) {
70          for (auto j : adj[i]) {
71              if (f[i] && f[j]) {
72                  e[i].push_back(j);
73              }
74          }
75      }
76      vector<bool> h(n);
77      for (int i = 0; i < n; i++) {
78          for (auto j : e[i]) {
79              h[j] = true;
80          }
81          for (auto j : e[i]) {
82              for (auto k : e[j]) {
83                  if (k != i && !h[k]) {
84                      if (ans.empty() || ans.size() > 6) {
85                          ans = {0, i, j, k, i, n - 1};
86                          break;
87                      }
88                  }
89                  if (!ans.empty() && ans.size() > 6) {
90                      break;
91                  }
92                  if (!ans.empty() && ans.size() > 6) {
93                      break;
94                  }
95              }
96          }
97          for (auto j : e[i]) {
98              h[j] = false;
99          }
100         if (!ans.empty() && ans.size() > 6) {
101             break;
102         }
103     }
104     if (ans.empty()) {
105         cout << -1 << "\n";
106         return 0;
107     }
108     cout << ans.size() - 1 << "\n";
109     for (auto x : ans) {
110         cout << x + 1 << " \n"[x == ans.back()];
111     }
112     return 0;
113 }

```

**672: Two Paths**

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: input.txt
- Output file: output.txt

Once archaeologists found m mysterious papers, each of which had a pair of integers written on them.

Ancient people were known to like writing down the indexes of the roads they walked along, as a b or b a, where a, b are the indexes of two different cities joint by the road . It is also known that the mysterious papers are pages of two travel journals (those days a new journal was written for every new journey).

During one journey the traveler could walk along one and the same road several times in one or several directions but in that case he wrote a new entry for each time in his journal. Besides, the archaeologists think that the direction the traveler took on a road had no effect upon the entry: the entry that looks like a b could refer to the road from a to b as well as to the road from b to a.

The archaeologists want to put the pages in the right order and reconstruct the two travel paths but unfortunately, they are bad at programming. Thats where you come in. Go help them!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 #ifdef ONLINE_JUDGE
5     ifstream fin("input.txt");
6     ofstream fout("output.txt");
7 #else
8     #define fin cin
9     #define fout cout
10 #endif
11 # struct DSU;
12 int main() {
13     int m;
14     fin >> m;
15     vector<int> a(m), b(m);
16     vector<int> v;
17     for (int i = 0; i < m; i++) {
18         fin >> a[i] >> b[i];
19         v.push_back(a[i]);
20         v.push_back(b[i]);
21     }
22     sort(v.begin(), v.end());
23     v.erase(unique(v.begin(), v.end()), v.end());
24     if (m == 1) {
25         fout << -1 << "\n";
26         return 0;
27     }
28     for (int i = 0; i < m; i++) {
29         a[i] = lower_bound(v.begin(), v.end(), a[i]) - v.begin();
30         b[i] = lower_bound(v.begin(), v.end(), b[i]) - v.begin();
31     }
32     int n = v.size();
33     vector<int> deg(n);
34     for (int i = 0; i < m; i++) {
35         deg[a[i]] += 1;
36         deg[b[i]] += 1;
37     }
38     DSU dsu(n);
39     vector<int> p;
```

```

40     int comp = n;
41     for (int i = 0; i < n; i++) {
42         if (deg[i] % 2) {
43             p.push_back(i);
44         }
45     }
46     for (int i = 0; i < m; i++) {
47         comp -= dsu.merge(a[i], b[i]);
48     }
49     if (p.size() > 4) {
50         fout << -1 << "\n";
51         return 0;
52     }
53     if (comp > 2) {
54         fout << -1 << "\n";
55         return 0;
56     }
57     vector<vector<pair<int, int>>> adj(n);
58     if (comp == 1) {
59         if (p.size() == 4) {
60             adj[p[0]].emplace_back(p[1], m);
61             adj[p[1]].emplace_back(p[0], m);
62             deg[p[0]] += 1, deg[p[1]] += 1;
63         }
64     } else {
65         if (p.size() == 4) {
66             bool ok = false;
67             for (int i = 1; i < 4; i++) {
68                 if (!dsu.same(p[0], p[i])) {
69                     adj[p[0]].emplace_back(p[i], m);
70                     adj[p[i]].emplace_back(p[0], m);
71                     deg[p[0]] += 1, deg[p[i]] += 1;
72                     ok = true;
73                     break;
74                 }
75             }
76             if (!ok) {
77                 fout << -1 << "\n";
78                 return 0;
79             }
80         } else if (p.size() == 2) {
81             for (int i = 0; i < n; i++) {
82                 if (!dsu.same(p[0], i)) {
83                     adj[p[0]].emplace_back(i, m);
84                     adj[i].emplace_back(p[0], m);
85                     deg[p[0]] += 1, deg[i] += 1;
86                     break;
87                 }
88             }
89         } else {
90             for (int i = 0; i < n; i++) {
91                 if (!dsu.same(0, i)) {
92                     adj[0].emplace_back(i, m);
93                     adj[i].emplace_back(0, m);
94                     deg[0] += 1, deg[i] += 1;
95                     break;
96                 }
97             }
98         }
99     }
100    int x = 0;
101    while (x < n - 1 && deg[x] % 2 == 0) {
102        x++;
103    }

```

```

104     vector<int> e;
105     vector<bool> vis(m + 1);
106     for (int i = 0; i < m; i++) {
107         adj[a[i]].emplace_back(b[i], i);
108         adj[b[i]].emplace_back(a[i], i);
109     }
110     function<void(int, int)> dfs = [&](int x, int p) {
111         while (!adj[x].empty()) {
112             auto [y, i] = adj[x].back();
113             adj[x].pop_back();
114             if (vis[i]) {
115                 continue;
116             }
117             vis[i] = true;
118             dfs(y, i);
119         }
120         if (p != -1) {
121             e.push_back(p);
122         }
123     };
124     dfs(x, -1);
125     auto it = find(e.begin(), e.end(), m);
126     vector<int> A, B;
127     if (it != e.end()) {
128         A.assign(e.begin(), it);
129         B.assign(it + 1, e.end());
130     } else {
131         A.assign(e.begin(), e.begin() + 1);
132         B.assign(e.begin() + 1, e.end());
133     }
134     fout << A.size() << "\n";
135     for (auto x : A) {
136         fout << x + 1 << " \n"[x == A.back()];
137     }
138     fout << B.size() << "\n";
139     for (auto x : B) {
140         fout << x + 1 << " \n"[x == B.back()];
141     }
142     return 0;
143 }
```

### 673: The Fox and the Complete Tree Traversal

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The fox Yae climbed the tree of the Sacred Sakura. A tree is a connected undirected graph that does not contain cycles.

The fox uses her magical powers to move around the tree. Yae can jump from vertex  $v$  to another vertex  $u$  if and only if the distance between these vertices does not exceed 2. In other words, in one jump Yae can jump from vertex  $v$  to vertex  $u$  if vertices  $v$  and  $u$  are connected by an edge, or if there exists such

vertex  $w$  that vertices  $v$  and  $w$  are connected by an edge, and also vertices  $u$  and  $w$  are connected by an edge.

After Yae was able to get the sakura petal, she wondered if there was a cyclic route in the tree  $v_1, v_2, \dots, v_n$  such that:

the fox can jump from vertex  $v_i$  to vertex  $v_{i+1}$ ,

the fox can jump from vertex  $v_n$  to vertex  $v_1$ ,

all  $v_i$  are pairwise distinct.

Help the fox determine if the required traversal exists.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<vector<int>> adj(n);
10    for (int i = 1; i < n; i++) {
11        int u, v;
12        cin >> u >> v;
13        u--, v--;
14        adj[u].push_back(v);
15        adj[v].push_back(u);
16    }
17    vector<int> parent(n, -1);
18    vector<int> dis(n);
19    auto bfs = [&](int s) {
20        queue<int> q;
21        q.push(s);
22        dis.assign(n, -1);
23        parent.assign(n, -1);
24        dis[s] = 0;
25        while (!q.empty()) {
26            int x = q.front();
27            q.pop();
28            for (auto y : adj[x]) {
29                if (dis[y] == -1) {
30                    dis[y] = dis[x] + 1;
31                    parent[y] = x;
32                    q.push(y);
33                }
34            }
35        }
36        return max_element(dis.begin(), dis.end()) - dis.begin();
37    };
38    int t = bfs(0);
39    int s = bfs(t);
40    vector<int> a;
41    vector<bool> path(n);
42    for (int i = s; i != -1; i = parent[i]) {
43        path[i] = true;

```

```

44         a.push_back(i);
45     }
46     for (int x = 0; x < n; x++) {
47         if (!path[x] && adj[x].size() != 1) {
48             cout << "No\n";
49             return 0;
50         }
51     }
52     cout << "Yes\n";
53     int m = a.size();
54     vector<int> ans;
55     for (int i = 0; i < m; i++) {
56         int x = 2 * i;
57         int y = 2 * i + 2;
58         if (x >= m) {
59             x = 2 * m - 1 - x;
60         }
61         if (y >= m) {
62             y = 2 * m - 1 - y;
63         }
64         ans.push_back(a[x]);
65         if (abs(x - y) == 2) {
66             int z = (x + y) / 2;
67             for (auto v : adj[a[z]]) {
68                 if (!path[v]) {
69                     ans.push_back(v);
70                 }
71             }
72         }
73     }
74     for (auto x : ans) {
75         cout << x + 1 << " \n"[x == ans.back()];
76     }
77     return 0;
78 }
```

**674: Square Table**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

While resting on the ship after the “Russian Code Cup” a boy named Misha invented an interesting game. He promised to give his quadrocopter to whoever will be the first one to make a rectangular table of size  $n \times m$ , consisting of positive integers such that the sum of the squares of numbers for each row and each column was also a square.

Since checking the correctness of the table manually is difficult, Misha asks you to make each number in the table to not exceed 108.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     for (int i = 0; i < n; i++) {
10         for (int j = 0; j < m; j++) {
11             int x = abs((i ? 4 : 4 * n - 5) * (j ? 4 : 4 * m - 5));
12             cout << x << " \n"[j == m - 1];
13         }
14     }
15     return 0;
16 }
```

## 675: Multithreading

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given the following concurrent program. There are N processes and the i-th process has the following pseudocode:

Here y is a shared variable. Everything else is local for the process. All actions on a given row are atomic, i.e. when the process starts executing a row it is never interrupted. Beyond that all interleavings are possible, i.e. every process that has yet work to do can be granted the rights to execute its next row. In the beginning  $y = 0$ . You will be given an integer W and  $n_i$ , for  $i = 1, \dots, N$ . Determine if it is possible that after all processes terminate,  $y = W$ , and if it is possible output an arbitrary schedule that will produce this final value.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int N, W;
8     cin >> N >> W;
9     vector<int> n(N);
10    for (int i = 0; i < N; i++) {
11        cin >> n[i];
12    }
```

```

13     if (N == 1) {
14         if (W != n[0]) {
15             cout << "No\n";
16         } else {
17             cout << "Yes\n";
18             for (int i = 1; i <= 2 * n[0]; i++) {
19                 cout << 1 << " \n"[i == 2 * n[0]];
20             }
21         }
22     }
23     return 0;
24 }
25 if (W <= 0) {
26     cout << "No\n";
27     return 0;
28 }
29 if (W == 1) {
30     int x = find(n.begin(), n.end(), 1) - n.begin();
31     if (x == N) {
32         cout << "No\n";
33     } else {
34         cout << "Yes\n";
35         cout << x + 1 << " ";
36         for (int i = 0; i < N; i++) {
37             if (i != x) {
38                 while (n[i]) {
39                     cout << i + 1 << " ";
40                     cout << i + 1 << " ";
41                     n[i] -= 1;
42                 }
43             }
44         }
45         cout << x + 1 << "\n";
46     }
47     return 0;
48 }
49 int sum = accumulate(n.begin(), n.end(), 0);
50 if (W > sum) {
51     cout << "No\n";
52     return 0;
53 }
54 cout << "Yes\n";
55 vector<int> ans;
56 for (int i = 0; i < N; i++) {
57     while (n[i] > 1 && W > 2) {
58         n[i] -= 1;
59         W -= 1;
60         ans.push_back(i + 1);
61         ans.push_back(i + 1);
62     }
63     for (int i = 0; i < N; i++) {
64         while (n[i] > 0 && W > 2) {
65             n[i] -= 1;
66             W -= 1;
67             ans.push_back(i + 1);
68             ans.push_back(i + 1);
69         }
70     }
71     int x = 0;
72     while (!n[x]) {
73         x++;
74     }
75     int y = x + 1;
76     while (!n[y]) {

```

```

77         y++;
78     }
79     n[x] -= 1;
80     n[y] -= 1;
81     ans.push_back(x + 1);
82     for (int i = 0; i < N; i++) {
83         while (i != x && n[i]) {
84             n[i] -= 1;
85             ans.push_back(i + 1);
86             ans.push_back(i + 1);
87         }
88     }
89     ans.push_back(x + 1);
90     ans.push_back(y + 1);
91     while (n[x]) {
92         n[x] -= 1;
93         ans.push_back(x + 1);
94         ans.push_back(x + 1);
95     }
96     ans.push_back(y + 1);
97     for (int i = 0; i < ans.size(); i++) {
98         cout << ans[i] << " \n"[i == ans.size() - 1];
99     }
100    return 0;
101 }
```

## 676: Greedy Change

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Billy investigates the question of applying greedy algorithm to different spheres of life. At the moment he is studying the application of greedy algorithm to the problem about change. There is an amount of  $n$  coins of different face values, and the coins of each value are not limited in number. The task is to collect the sum  $x$  with the minimum amount of coins. Greedy algorithm with each its step takes the coin of the highest face value, not exceeding  $x$ . Obviously, if among the coins' face values exists the face value 1, any sum  $x$  can be collected with the help of greedy algorithm. However, greedy algorithm does not always give the optimal representation of the sum, i.e. the representation with the minimum amount of coins. For example, if there are face values  $\{1, 3, 4\}$  and it is asked to collect the sum 6, greedy algorithm will represent the sum as  $4 + 1 + 1$ , while the optimal representation is  $3 + 3$ , containing one coin less. By the given set of face values find out if there exist such a sum  $x$  that greedy algorithm will collect in a non-optimal way. If such a sum exists, find out the smallest of these sums.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
```

```

2  using namespace std;
3  using i64 = long long;
4  int main() {
5      ios::sync_with_stdio(false);
6      cin.tie(nullptr);
7      int n;
8      cin >> n;
9      vector<int> a(n);
10     for (int i = 0; i < n; i++) {
11         cin >> a[i];
12     }
13     int ans = 2E9;
14     auto greedy = [&](int x) {
15         int res = 0;
16         for (int i = 0; i < n; i++) {
17             res += x / a[i];
18             x %= a[i];
19         }
20         return res;
21     };
22     for (int i = 0; i < n; i++) {
23         int x = a[i] - 1;
24         int res = 0;
25         for (int j = i + 1; j < n; j++) {
26             res += x / a[j];
27             x %= a[j];
28             if (greedy(a[i] - 1 - x + a[j]) > res + 1) {
29                 ans = min(ans, a[i] - 1 - x + a[j]);
30             }
31         }
32     }
33     if (ans == 2E9) {
34         ans = -1;
35     }
36     cout << ans << "\n";
37     return 0;
38 }
```

### 677: Guess the String

- Time limit: 6 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

This is an interactive problem. You have to use flush operation right after printing each line. For example, in C++ you should use the function `fflush(stdout)`, in Java or Kotlin - `System.out.flush()`, and in Python - `sys.stdout.flush()`.

The jury has a string  $s$  consisting of characters 0 and/or 1. The first character of this string is 0. The length of this string is  $n$ . You have to guess this string. Let's denote  $s[l..r]$  as the substring of  $s$  from  $l$  to  $r$  (i. e.  $s[l..r]$  is the string  $s_ls_{l+1}\dots s_r$ ).

Let the prefix function of the string  $s$  be an array  $[p_1, p_2, \dots, p_n]$ , where  $p_i$  is the greatest integer

$j \in [0, i - 1]$  such that  $s[1..j] = s[i - j + 1..i]$ . Also, let the antiprefix function of the string  $s$  be an array  $[q_1, q_2, \dots, q_n]$ , where  $q_i$  is the greatest integer  $j \in [0, i - 1]$  such that  $s[1..j]$  differs from  $s[i - j + 1..i]$  in every position.

For example, for the string 011001, its prefix function is  $[0, 0, 0, 1, 1, 2]$ , and its antiprefix function is  $[0, 1, 1, 2, 3, 4]$ .

You can ask queries of two types to guess the string  $s$ :

1  $i$  - “what is the value of  $p_i$ ?“;

2  $i$  - “what is the value of  $q_i$ ?“.

You have to guess the string by asking no more than 789 queries. Note that giving the answer does not count as a query.

In every test and in every test case, the string  $s$  is fixed beforehand.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
5 int query(int t, int p) {
6     cout << t << " " << p << endl;
7     int ans;
8     cin >> ans;
9     return ans;
10 }
11 void solve() {
12     int n;
13     cin >> n;
14     vector<int> a(n);
15     if (query(1, 2) == 1) {
16         a[1] = 0;
17     } else {
18         a[1] = 1;
19     }
20     vector<int> prev(n), val(n);
21     for (int l = n; l > 2; ) {
22         if (rng() % 2 == 0) {
23             int x = query(1, l);
24             if (x >= 2) {
25                 for (int j = l - x; j < l; j++) {
26                     prev[j] = j - (l - x);
27                     val[j] = 0;
28                 }
29                 l -= x;
30             } else if (x == 0) {
31                 if (a[1] == 1) {
32                     prev[l - 1] = 0;
33                     val[l - 1] = 1;
34                     prev[l - 2] = 0;
35                     val[l - 2] = 1;
36                     l -= 2;
37                 } else {
38             }
39         }
40     }
41 }
```

```
38                     prev[l - 1] = 0;
39                     val[l - 1] = 1;
40                     l--;
41                 }
42             } else {
43                 if (a[1] == 0) {
44                     prev[l - 1] = 0;
45                     val[l - 1] = 0;
46                     prev[l - 2] = 0;
47                     val[l - 2] = 1;
48                     l -= 2;
49             } else {
50                 prev[l - 1] = 0;
51                 val[l - 1] = 0;
52                 l--;
53             }
54         }
55     } else {
56         int x = query(2, l);
57         if (x >= 2) {
58             for (int j = l - x; j < l; j++) {
59                 prev[j] = j - (l - x);
60                 val[j] = 1;
61             }
62             l -= x;
63         } else if (x == 0) {
64             if (a[1] == 1) {
65                 prev[l - 1] = 0;
66                 val[l - 1] = 0;
67                 prev[l - 2] = 0;
68                 val[l - 2] = 0;
69                 l -= 2;
70             } else {
71                 prev[l - 1] = 0;
72                 val[l - 1] = 0;
73                 l--;
74             }
75         } else {
76             if (a[1] == 0) {
77                 prev[l - 1] = 0;
78                 val[l - 1] = 1;
79                 prev[l - 2] = 0;
80                 val[l - 2] = 0;
81                 l -= 2;
82             } else {
83                 prev[l - 1] = 0;
84                 val[l - 1] = 1;
85                 l--;
86             }
87         }
88     }
89 }
90 for (int i = 2; i < n; i++) {
91     a[i] = a[prev[i]] ^ val[i];
92 }
93 cout << 0 << " ";
94 for (int i = 0; i < n; i++) {
95     cout << a[i];
96 }
97 cout << endl;
98 int ans;
99 cin >> ans;
100}
101int main() {
```

```

102     ios::sync_with_stdio(false);
103     cin.tie(nullptr);
104     int t;
105     cin >> t;
106     while (t--) {
107         solve();
108     }
109     return 0;
110 }
```

## 678: Inverse Transformation

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A permutation scientist is studying a self-transforming permutation  $a$  consisting of  $n$  elements  $a_1, a_2, \dots, a_n$ .

A permutation is a sequence of integers from 1 to  $n$  of length  $n$  containing each number exactly once. For example,  $[1], [4, 3, 5, 1, 2]$  are permutations, while  $[1, 1], [4, 3, 1]$  are not.

The permutation transforms day by day. On each day, each element  $x$  becomes  $a_x$ , that is,  $a_x$  becomes  $a_{a_x}$ . Specifically:

on the first day, the permutation becomes  $b$ , where  $b_x = a_{a_x}$ ;

on the second day, the permutation becomes  $c$ , where  $c_x = b_{b_x}$ ;

...

You're given the permutation  $a'$  on the  $k$ -th day.

Define  $\sigma(x) = a_x$ , and define  $f(x)$  as the minimal positive integer  $m$  such that  $\sigma^m(x) = x$ , where  $\sigma^m(x)$  denotes  $\underbrace{\sigma(\sigma(\dots\sigma(x)\dots))}_{m \text{ times}}$ .

For example, if  $a = [2, 3, 1]$ , then  $\sigma(1) = 2, \sigma^2(1) = \sigma(\sigma(1)) = \sigma(2) = 3, \sigma^3(1) = \sigma(\sigma(\sigma(1))) = \sigma(3) = 1$ , so  $f(1) = 3$ . And if  $a = [4, 2, 1, 3], \sigma(2) = 2$  so  $f(2) = 1; \sigma(3) = 1, \sigma^2(3) = 4, \sigma^3(3) = 3$  so  $f(3) = 3$ .

Find the initial permutation  $a$  such that  $\sum_{i=1}^n \frac{1}{f(i)}$  is minimum possible.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int power(int a, int b, int p) {
5     int res = 1 % p;
6     for (; b > 0; b /= 2, a = 1LL * a * a % p) {
7         if (b % 2 == 1) {
8             res = 1LL * res * a % p;
9         }
10    }
11    return res;
12 }
13 void solve() {
14     int n, k;
15     cin >> n >> k;
16     vector<int> a(n);
17     for (int i = 0; i < n; i++) {
18         cin >> a[i];
19         a[i]--;
20     }
21     vector<bool> vis(n);
22     vector<vector<vector<int>>> cyc(n + 1);
23     vector<int> cnt(n + 1);
24     for (int i = 0; i < n; i++) {
25         if (vis[i]) continue;
26         int j = i;
27         vector<int> c;
28         while (!vis[j]) {
29             c.push_back(j);
30             vis[j] = true;
31             j = a[j];
32         }
33         cyc[c.size()].push_back(c);
34     }
35     for (int i = 1; i <= n; i++) {
36         if (i % 2 == 0 && cyc[i].size() > 0) {
37             if (k > 20 || cyc[i].size() % (1 << k) != 0) {
38                 cout << "NO\n";
39                 return;
40             }
41         }
42     }
43     cout << "YES\n";
44     for (int i = 1; i <= n; i++) {
45         while (!cyc[i].empty()) {
46             int t = __lg(cyc[i].size());
47             t = min(t, k);
48             int pw = power(2, k - t, i);
49             vector<int> cs(cyc[i].end() - (1 << t), cyc[i].end());
50             cyc[i].erase(cyc[i].end() - (1 << t), cyc[i].end());
51             for (auto &c : cs) {
52                 vector<int> nc(i);
53                 for (int j = 0; j < i; j++) {
54                     nc[1LL * pw * j % i] = c[j];
55                 }
56                 c = nc;
57             }
58             vector<int> nc;
59             nc.reserve(i << t);
60             for (int j = 0; j < i; j++) {
61                 for (int k = 0; k < (1 << t); k++) {
62                     nc.push_back(cs[k][j]);
63                 }
64             }
}

```

```

65         for (int j = 0; j < nc.size(); j++) {
66             a[nc[j]] = nc[(j + 1) % nc.size()];
67         }
68     }
69     for (int i = 0; i < n; i++) {
70         cout << a[i] + 1 << " \n"[i == n - 1];
71     }
72 }
73 int main() {
74     ios::sync_with_stdio(false);
75     cin.tie(nullptr);
76     int t;
77     cin >> t;
78     while (t--) {
79         solve();
80     }
81     return 0;
82 }
83 }
```

### 679: Xorcerer's Stones

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Misha had been banned from playing chess for good since he was accused of cheating with an engine. Therefore, he retired and decided to become a xorcerer.

One day, while taking a walk in a park, Misha came across a rooted tree with nodes numbered from 1 to  $n$ . The root of the tree is node 1.

For each  $1 \leq i \leq n$ , node  $i$  contains  $a_i$  stones in it. Misha has recently learned a new spell in his xorcery class and wants to test it out. A spell consists of:

Choose some node  $i$  ( $1 \leq i \leq n$ ).

Calculate the bitwise XOR  $x$  of all  $a_j$  such that node  $j$  is in the subtree of  $i$  ( $i$  belongs to its own subtree).

Set  $a_j$  equal to  $x$  for all nodes  $j$  in the subtree of  $i$ .

Misha can perform at most  $2n$  spells and he wants to remove all stones from the tree. More formally, he wants  $a_i = 0$  to hold for each  $1 \leq i \leq n$ . Can you help him perform the spells?

A tree with  $n$  nodes is a connected acyclic graph which contains  $n - 1$  edges. The subtree of node  $i$  is the set of all nodes  $j$  such that  $i$  lies on the simple path from 1 (the root) to  $j$ . We consider  $i$  to be contained in its own subtree.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    vector<int> p(n);
14    p[0] = -1;
15    vector<vector<int>> adj(n);
16    for (int i = 1; i < n; i++) {
17        cin >> p[i];
18        p[i]--;
19        adj[p[i]].push_back(i);
20    }
21    vector<array<int, 32>> dp(n), pre(n);
22    vector<int> siz(n);
23    auto dfs1 = [&](auto dfs1, int x) -> void {
24        siz[x] = 1;
25        for (auto y : adj[x]) {
26            dfs1(dfs1, y);
27            siz[x] += siz[y];
28        }
29        dp[x][a[x]] = 1;
30        for (auto y : adj[x]) {
31            array<int, 32> g{};
32            for (int i = 0; i < 32; i++) {
33                for (int j = 0; j < 32; j++) {
34                    g[i ^ j] |= dp[x][i] & dp[y][j];
35                }
36            }
37            pre[y] = dp[x];
38            dp[x] = g;
39        }
40        if (siz[x] % 2 == 0) dp[x][0] = 1;
41    };
42    dfs1(dfs1, 0);
43    if (!dp[0][0]) {
44        cout << -1 << "\n";
45        return 0;
46    }
47    vector<int> ans;
48    auto dfs2 = [&](auto dfs2, int x, int val) -> void {
49        if (val == 0 && siz[x] % 2 == 0) {
50            ans.push_back(x);
51            return;
52        }
53        int d = adj[x].size();
54        assert(dp[x][val]);
55        for (int k = d - 1; k >= 0; k--) {
56            int y = adj[x][k];
57            int i = 0;
58            while (!dp[y][i] || !pre[y][val ^ i]) i++;
59            val ^= i;
60            dfs2(dfs2, y, i);
61        }
62        assert(val == a[x]);

```

```

63     };
64     dfs2(dfs2, 0, 0);
65     ans.push_back(0);
66     cout << ans.size() << "\n";
67     for (int i = 0; i < ans.size(); i++) {
68         cout << ans[i] + 1 << " \n"[i == ans.size() - 1];
69     }
70     return 0;
71 }
```

**680: Cactus Wall**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp is playing Minecraft and wants to build a wall of cacti. He wants to build it on a field of sand of the size of  $n \times m$  cells. Initially, there are cacti in some cells of the field. Note that, in Minecraft, cacti cannot grow on cells adjacent to each other by side - and the initial field meets this restriction. Monocarp can plant new cacti (they must also fulfil the aforementioned condition). He can't chop down any of the cacti that are already growing on the field - he doesn't have an axe, and the cacti are too prickly for his hands.

Monocarp believes that the wall is complete if there is no path from the top row of the field to the bottom row, such that:

each two consecutive cells in the path are adjacent by side;

no cell belonging to the path contains a cactus.

Your task is to plant the minimum number of cacti to build a wall (or to report that this is impossible).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<string> s(n);
8     for (int i = 0; i < n; i++) {
9         cin >> s[i];
10    }
11    int ans = -1;
12    vector<i64> good(n, vector<i64>(m, 1));
```

```

13     for (int i = 0; i < n; i++) {
14         for (int j = 0; j < m; j++) {
15             if (s[i][j] == '#') {
16                 if (i) good[i - 1][j] = 0;
17                 if (j) good[i][j - 1] = 0;
18                 if (i < n - 1) good[i + 1][j] = 0;
19                 if (j < m - 1) good[i][j + 1] = 0;
20             }
21         }
22     }
23     vector dp(n, vector<int>(m, -1));
24     vector last(n, vector<pair<int, int>>(m));
25     deque<tuple<int, int, int, int, int>> q;
26     for (int i = 0; i < n; i++) {
27         if (good[i][0]) {
28             if (s[i][0] == '.') q.emplace_back(i, 0, 1, -1, -1);
29             else q.emplace_front(i, 0, 0, -1, -1);
30         }
31     }
32     while (!q.empty()) {
33         auto [x, y, d, lx, ly] = q.front();
34         q.pop_front();
35         if (dp[x][y] != -1) continue;
36         dp[x][y] = d;
37         last[x][y] = {lx, ly};
38         for (auto dx : {-1, 1}) for (auto dy : {-1, 1}) {
39             int nx = x + dx, ny = y + dy;
40             if (nx < 0 || nx >= n || ny < 0 || ny >= m || !good[nx][ny]) continue;
41             if (s[nx][ny] == '.') q.emplace_back(nx, ny, d + 1, x, y);
42             else q.emplace_front(nx, ny, d, x, y);
43         }
44     }
45     int x = -1;
46     for (int i = 0; i < n; i++) {
47         if (dp[i][m - 1] != -1 && (ans == -1 || ans > dp[i][m - 1])) {
48             ans = dp[i][m - 1];
49             x = i;
50         }
51     }
52     if (ans == -1) {
53         cout << "NO\n";
54         return;
55     }
56     cout << "YES\n";
57     int y = m - 1;
58     while (1) {
59         s[x][y] = '#';
60         if (!y) break;
61         tie(x, y) = last[x][y];
62     }
63     for (int i = 0; i < n; i++) {
64         cout << s[i] << "\n";
65     }
66 }
67 int main() {
68     ios::sync_with_stdio(false);
69     cin.tie(nullptr);
70     int t;
71     cin >> t;
72     while (t--) {
73         solve();
74     }
75     return 0;
76 }

```

## 681: Doremy's Experimental Tree

- Time limit: 2.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Doremy has an edge-weighted tree with  $n$  vertices whose weights are integers between 1 and  $10^9$ . She does  $\frac{n(n+1)}{2}$  experiments on it.

In each experiment, Doremy chooses vertices  $i$  and  $j$  such that  $j \leq i$  and connects them directly with an edge with weight 1. Then, there is exactly one cycle (or self-loop when  $i = j$ ) in the graph. Doremy defines  $f(i, j)$  as the sum of lengths of shortest paths from every vertex to the cycle.

Formally, let  $\text{dis}_{i,j}(x, y)$  be the length of the shortest path between vertex  $x$  and  $y$  when the edge  $(i, j)$  of weight 1 is added, and  $S_{i,j}$  be the set of vertices that are on the cycle when edge  $(i, j)$  is added. Then,

$$f(i, j) = \sum_{x=1}^n \left( \min_{y \in S_{i,j}} \text{dis}_{i,j}(x, y) \right).$$

Doremy writes down all values of  $f(i, j)$  such that  $1 \leq j \leq i \leq n$ , then goes to sleep. However, after waking up, she finds that the tree has gone missing. Fortunately, the values of  $f(i, j)$  are still in her notebook, and she knows which  $i$  and  $j$  they belong to. Given the values of  $f(i, j)$ , can you help her restore the tree?

It is guaranteed that at least one suitable tree exists.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n;
9     cin >> n;
10    vector<tuple<i64, int, int>> edges;
11    vector<i64> D(n);
12    for (int i = 0; i < n; i++) {
13        for (int j = 0; j <= i; j++) {
14            i64 w;
15            cin >> w;
16            if (i == j) {
17                D[i] = w;
18            } else {

```

```

19             edges.emplace_back(w, j, i);
20         }
21     }
22 }
23 DSU dsu(n);
24 sort(edges.begin(), edges.end(), greater());
25 for (auto [w, a, b] : edges) {
26     if (dsu.same(a, b)) {
27         continue;
28     }
29     dsu.merge(a, b);
30     cout << a + 1 << " " << b + 1 << " " << (D[a] + D[b] - 2 * w) / n << "\n";
31 }
32 return 0;
33 }
```

## 682: Make It Connected

- Time limit: 1 second
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given a simple undirected graph consisting of  $n$  vertices. The graph doesn't contain self-loops, there is at most one edge between each pair of vertices. Your task is simple: make the graph connected.

You can do the following operation any number of times (possibly zero):

Choose a vertex  $u$  arbitrarily.

For each vertex  $v$  satisfying  $v \neq u$  in the graph individually, if  $v$  is adjacent to  $u$ , remove the edge between  $u$  and  $v$ , otherwise add an edge between  $u$  and  $v$ .

Find the minimum number of operations required to make the graph connected. Also, find any sequence of operations with the minimum length that makes the graph connected.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector g(n, vector<int>(n));
8     vector<int> deg(n);
9     for (int i = 0; i < n; i++) {
10         for (int j = 0; j < n; j++) {
11             char x;
12             cin >> x;
```

```
13         g[i][j] = x - '0';
14         deg[i] += g[i][j];
15     }
16 }
17 vector<vector<int>> comp;
18 vector<int> vis(n);
19 for (int i = 0; i < n; i++) {
20     if (vis[i]) {
21         continue;
22     }
23     vector<int> q;
24     q.push_back(i);
25     vis[i] = 1;
26     for (int i = 0; i < int(q.size()); i++) {
27         int x = q[i];
28         for (int y = 0; y < n; y++) {
29             if (g[x][y] && !vis[y]) {
30                 q.push_back(y);
31                 vis[y] = 1;
32             }
33         }
34     }
35     comp.push_back(q);
36 }
37 if (comp.size() == 1) {
38     cout << 0 << "\n";
39     return;
40 }
41 for (auto s : comp) {
42     if (s.size() == 1) {
43         cout << 1 << "\n";
44         cout << s[0] + 1 << "\n";
45         return;
46     }
47     if (deg[s.back()] != int(s.size()) - 1) {
48         cout << 1 << "\n";
49         cout << s.back() + 1 << "\n";
50         return;
51     }
52     for (auto x : s) {
53         if (deg[x] != int(s.size()) - 1) {
54             cout << 1 << "\n";
55             cout << x + 1 << "\n";
56             return;
57         }
58     }
59 }
60 if (comp.size() > 2) {
61     cout << 2 << "\n";
62     cout << comp[0][0] + 1 << " " << comp[1][0] + 1 << "\n";
63     return;
64 }
65 if (comp[0].size() > comp[1].size()) {
66     swap(comp[0], comp[1]);
67 }
68 cout << comp[0].size() << "\n";
69 for (auto x : comp[0]) {
70     cout << x + 1 << " \n"[x == comp[0].back()];
71 }
72 }
73 int main() {
74     ios::sync_with_stdio(false);
75     cin.tie(nullptr);
76     int t;
```

```
77     cin >> t;
78     while (t--) {
79         solve();
80     }
81     return 0;
82 }
```

### 683: The Beach

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Andrew loves the sea. That's why, at the height of the summer season, he decided to go to the beach, taking a sunbed with him to sunbathe.

The beach is a rectangular field with  $n$  rows and  $m$  columns. Some cells of the beach are free, some have roads, stones, shops and other non-movable objects. Some of two adjacent along the side cells can have sunbeds located either horizontally or vertically.

Andrew hopes to put his sunbed somewhere, but that's a bad luck, there may no longer be free places for him! That's why Andrew asked you to help him to find a free place for his sunbed. Andrew's sunbed also should be places on two adjacent cells.

If there are no two adjacent free cells, then in order to free some place for a sunbed, you will have to disturb other tourists. You can do the following actions:

Come to some sunbed and, after causing  $p$  units of discomfort to its owner, lift the sunbed by one of its sides and rotate it by 90 degrees. One half of the sunbed must remain in the same cell and another half of the sunbed must move to the free cell. At the same time, anything could be on the way of a sunbed during the rotation . Rotation of the sunbed by 90 degrees around cell (1, 2).

Come to some sunbed and, after causing  $q$  units of discomfort to its owner, shift the sunbed along its long side by one cell. One half of the sunbed must move to the place of another, and another - to the free cell. Shift of the sunbed by one cell to the right.

In any moment each sunbed occupies two adjacent free cells. You cannot move more than one sunbed at a time.

Help Andrew to free a space for his sunbed, causing the minimum possible number of units of discomfort to other tourists, or detect that it is impossible.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 const int dx[] = {-1, 1, 0, 0};
5 const int dy[] = {0, 0, -1, 1};
6 constexpr i64 inf = 1E18;
7 int main() {
8     ios::sync_with_stdio(false);
9     cin.tie(nullptr);
10    int n, m, p, q;
11    cin >> n >> m >> p >> q;
12    vector<string> s(n);
13    for (int i = 0; i < n; i++) {
14        cin >> s[i];
15    }
16    const int N = n * m;
17    vector<int> match(N, -1);
18    for (int i = 0; i < n; i++) {
19        for (int j = 0; j < m; j++) {
20            if (s[i][j] == 'L') {
21                match[i * m + j] = i * m + j + 1;
22                match[i * m + j + 1] = i * m + j;
23            } else if (s[i][j] == 'U') {
24                match[i * m + j] = (i + 1) * m + j;
25                match[(i + 1) * m + j] = i * m + j;
26            }
27        }
28    }
29    auto bfs = [&](int t) {
30        vector<i64> dis(N, inf);
31        queue<pair<int, i64>> q1, q2;
32        for (int i = 0; i < n; i++) {
33            for (int j = 0; j < m; j++) {
34                if ((i + j) % 2 == t && s[i][j] == '.') {
35                    q1.emplace(i * m + j, 0);
36                }
37            }
38        }
39        while (!q1.empty() || !q2.empty()) {
40            int u;
41            i64 d;
42            if (!q1.empty() && (q2.empty() || q1.front().second < q2.front().second)) {
43                tie(u, d) = q1.front();
44                q1.pop();
45            } else {
46                tie(u, d) = q2.front();
47                q2.pop();
48            }
49            if (dis[u] != inf) {
50                continue;
51            }
52            dis[u] = d;
53            int x = u / m, y = u % m;
54            for (int i = 0; i < 4; i++) {
55                int nx = x + dx[i];
56                int ny = y + dy[i];
57                if (nx < 0 || nx >= n || ny < 0 || ny >= m || match[nx * m + ny] == -1) {
58                    continue;
59                }
60                int mx = nx + dx[i];
61                int my = ny + dy[i];
62                if (mx < 0 || mx >= n || my < 0 || my >= m || match[nx * m + ny] != mx * m)

```

```

63         + my) {
64     q1.emplace(match[nx * m + ny], d + p);
65 } else {
66     q2.emplace(match[nx * m + ny], d + q);
67 }
68 }
69 return dis;
70 };
71 auto f0 = bfs(0);
72 auto f1 = bfs(1);
73 i64 ans = inf;
74 for (int i = 0; i < n; i++) {
75     for (int j = 0; j < m; j++) {
76         if ((i + j) % 2 == 0) {
77             i64 v = f0[i * m + j];
78             if (i) {
79                 ans = min(ans, v + f1[(i - 1) * m + j]);
80             }
81             if (i + 1 < n) {
82                 ans = min(ans, v + f1[(i + 1) * m + j]);
83             }
84             if (j) {
85                 ans = min(ans, v + f1[i * m + j - 1]);
86             }
87             if (j + 1 < m) {
88                 ans = min(ans, v + f1[i * m + j + 1]);
89             }
90         }
91     }
92 }
93 if (ans == inf) {
94     ans = -1;
95 }
96 cout << ans << "\n";
97 return 0;
98 }

```

### 684: Joking (Easy Version)

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The only difference between this problem and the hard version is the maximum number of questions.

This is an interactive problem.

There is a hidden integer  $1 \leq x \leq n$  which you have to find. In order to find it you can ask at most **82** questions.

In each question you can choose a non-empty integer set  $S$  and ask if  $x$  belongs to  $S$  or not, after each question, if  $x$  belongs to  $S$ , you'll receive "YES", otherwise "NO".

But the problem is that not all answers are necessarily true (some of them are joking), it's just guaranteed that for each two consecutive questions, at least one of them is answered correctly.

Additionally to the questions, you can make at most 2 guesses for the answer  $x$ . Each time you make a guess, if you guess  $x$  correctly, you receive ":" and your program should terminate, otherwise you'll receive ":".

As a part of the joking, we will not fix the value of  $x$  in the beginning. Instead, it can change throughout the interaction as long as all the previous responses are valid as described above.

Note that your answer guesses are always answered correctly. If you ask a question before and after a guess, at least one of these two questions is answered correctly, as normal.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void dfs(int a, int b, int k) {
5     if (k == 0) {
6         if (a + b > 2) {
7             cerr << a << " " << b << "\n";
8         }
9         return;
10    }
11    if (a <= 2 * b) {
12        int x = (a + 1) / 2, y = b / 2;
13        dfs(x + y, a - x, k - 1);
14        dfs(a - x + b - y, x, k - 1);
15    } else {
16        dfs(a, 0, k - 1);
17        dfs(b, a, k - 1);
18    }
19 }
20 int dp[121][121];
21 int gx[121][121], gy[121][121];
22 int main() {
23     ios::sync_with_stdio(false);
24     cin.tie(nullptr);
25     memset(dp, 0x3f, sizeof(dp));
26     for (int s = 0; s <= 120; s++) {
27         for (int i = 0; i <= s; i++) {
28             const int j = s - i;
29             if (s <= 2) {
30                 dp[i][j] = 0;
31             } else {
32                 for (int x = 0; x <= i; x++) {
33                     for (int y = 0; y <= j; y++) {
34                         int t = max(dp[x + y][i - x], dp[i - x + j - y][x]) + 1;
35                         if (t < dp[i][j]) {
36                             dp[i][j] = t;
37                             gx[i][j] = x;
38                             gy[i][j] = y;
39                         }
40                     }
41                 }
42             }
43         }
44     }
45 }
```

```
43         }
44     }
45     int n;
46     cin >> n;
47     vector<int> T, F;
48     for (int i = 1; i <= n; i++) {
49         T.push_back(i);
50     }
51     while (T.size() + F.size() > 2) {
52         int x, y;
53         if (T.size() + F.size() <= 120) {
54             x = gx[T.size()][F.size()];
55             y = gy[T.size()][F.size()];
56         } else {
57             x = T.size() / 2;
58             y = F.size() / 2;
59         }
60         vector T1(T.begin(), T.begin() + x), T0(T.begin() + x, T.end());
61         vector F1(F.begin(), F.begin() + y), F0(F.begin() + y, F.end());
62         cout << "? " << x + y;
63         for (auto x : T1) {
64             cout << " " << x;
65         }
66         for (auto x : F1) {
67             cout << " " << x;
68         }
69         cout << endl;
70         string s;
71         cin >> s;
72         T = F = {};
73         if (s == "YES") {
74             for (auto x : T1) {
75                 T.push_back(x);
76             }
77             for (auto x : F1) {
78                 T.push_back(x);
79             }
80             for (auto x : T0) {
81                 F.push_back(x);
82             }
83         } else {
84             for (auto x : T0) {
85                 T.push_back(x);
86             }
87             for (auto x : F0) {
88                 T.push_back(x);
89             }
90             for (auto x : T1) {
91                 F.push_back(x);
92             }
93         }
94     }
95     for (auto x : T) {
96         cout << "! " << x << endl;
97         string s;
98         cin >> s;
99         if (s == ":)") {
100             return 0;
101         }
102     }
103     for (auto x : F) {
104         cout << "! " << x << endl;
105         string s;
106         cin >> s;
```

```

107         if (s == ":") {
108             return 0;
109         }
110     }
111 }
112 }
```

## data structures

### 685: Field Should Not Be Empty

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a permutation<sup>†</sup>  $p$  of length  $n$ .

We call index  $x$  good if for all  $y < x$  it holds that  $p_y < p_x$  and for all  $y > x$  it holds that  $p_y > p_x$ . We call  $f(p)$  the number of good indices in  $p$ .

You can perform the following operation: pick 2 distinct indices  $i$  and  $j$  and swap elements  $p_i$  and  $p_j$ .

Find the maximum value of  $f(p)$  after applying the aforementioned operation exactly once.

<sup>†</sup>A permutation of length  $n$  is an array consisting of  $n$  distinct integers from 1 to  $n$  in arbitrary order. For example,  $[2, 3, 1, 5, 4]$  is a permutation, but  $[1, 2, 2]$  is not a permutation (2 appears twice in the array), and  $[1, 3, 4]$  is also not a permutation ( $n = 3$  but there is 4 in the array).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> p(n), invp(n);
8     for (int i = 0; i < n; i++) {
9         cin >> p[i];
10        p[i]--;
11        invp[p[i]] = i;
12    }
13    vector<array<int, 2>> pre(n, {-1, -1}), suf(n, {n, n});
14    for (int i = 1; i < n; i++) {
15        pre[i] = pre[i - 1];
16        int x = p[i - 1];
17        for (int j = 0; j < 2; j++) {
18            if (x > pre[i][j]) {
19                swap(x, pre[i][j]);
20            }
21        }
22    }
23    for (int i = 0; i < n; i++) {
24        int x = p[i];
25        for (int j = 0; j < 2; j++) {
26            if (x < suf[i][j]) {
27                swap(x, suf[i][j]);
28            }
29        }
30    }
31    cout << invp[0] << endl;
32 }
```

```

20         }
21     }
22   }
23   for (int i = n - 2; i >= 0; i--) {
24     suf[i] = suf[i + 1];
25     int x = p[i + 1];
26     for (int j = 0; j < 2; j++) {
27       if (x < suf[i][j]) {
28         swap(x, suf[i][j]);
29       }
30     }
31   }
32   vector<int> bad(n);
33   int cnt = 0;
34   int ans = 0;
35   map<pair<int, int>, int> mp;
36   for (int i = 0; i < n; i++) {
37     if (p[i] == i) {
38       if (pre[i][0] < i && suf[i][0] > i) {
39         bad[i] = 1;
40         cnt += 1;
41       } else if (pre[i][1] < i && suf[i][1] > i) {
42         mp[{invp[pre[i][0]], invp[suf[i][0]]}] += 1;
43       }
44     } else if (invp[i] < i) {
45       if (suf[i][0] > i && pre[i][0] == i) {
46         mp[{invp[i], i}] += 1;
47       }
48     } else {
49       if (suf[i][0] == i && pre[i][0] < i) {
50         mp[{i, invp[i]}] += 1;
51       }
52     }
53   }
54   ans = cnt - 2;
55   if (cnt < n) {
56     ans = cnt - 1;
57   }
58   for (int i = 1; i < n; i++) {
59     if (!bad[i] && !bad[i - 1]) {
60       ans = cnt;
61     }
62   }
63   for (int i = 1; i < n; i++) {
64     bad[i] += bad[i - 1];
65   }
66   for (auto [s, c] : mp) {
67     auto [x, y] = s;
68     int v = bad[y];
69     if (x) {
70       v -= bad[x - 1];
71     }
72     ans = max(ans, cnt - v + c);
73   }
74   cout << ans << "\n";
75 }
76 int main() {
77   ios::sync_with_stdio(false);
78   cin.tie(nullptr);
79   int t;
80   cin >> t;
81   while (t--) {
82     solve();
83   }

```

```

84     return 0;
85 }
```

## 686: Babysitting

- Time limit: 7 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Theofanis wants to play video games, however he should also take care of his sister. Since Theofanis is a CS major, he found a way to do both. He will install some cameras in his house in order to make sure his sister is okay.

His house is an undirected graph with  $n$  nodes and  $m$  edges. His sister likes to play at the edges of the graph, so he has to install a camera to at least one endpoint of every edge of the graph. Theofanis wants to find a vertex cover that maximizes the minimum difference between indices of the chosen nodes.

More formally, let  $a_1, a_2, \dots, a_k$  be a vertex cover of the graph. Let the minimum difference between indices of the chosen nodes be the minimum  $|a_i - a_j|$  (where  $i \neq j$ ) out of the nodes that you chose. If  $k = 1$  then we assume that the minimum difference between indices of the chosen nodes is  $n$ .

Can you find the maximum possible minimum difference between indices of the chosen nodes over all vertex covers?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct TwoSat;
5 void solve() {
6     int n, m;
7     cin >> n >> m;
8     vector<int> u(m), v(m);
9     for (int i = 0; i < m; i++) {
10         cin >> u[i] >> v[i];
11         u[i]--, v[i]--;
12     }
13     int ans = *ranges::partition_point(ranges::iota_view(1, n + 1),
14                                         [&](int d) {
15                 TwoSat ts(2 * n);
16                 for (int i = 0; i < m; i++) {
17                     ts.addClause(u[i], 1, v[i], 1);
18                 }
19                 for (int i = 0; i < n; i += d) {
20                     ts.addClause(i, 0, n + i, 1);
21             });
22 }
```

```

21         for (int j = i + 1; j < i + d && j < n; j++) {
22             ts.addClause(n + j - 1, 0, n + j, 1);
23             ts.addClause(j, 0, n + j, 1);
24             ts.addClause(n + j - 1, 0, j, 0);
25             if (i + d < n) {
26                 ts.addClause(j, 0, n + min(n - 1, j - 1 + d), 0);
27             }
28         }
29     }
30     return ts.satisfiable();
31 } - 1;
32 cout << ans << "\n";
33 }
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int t;
38     cin >> t;
39     while (t--) {
40         solve();
41     }
42     return 0;
43 }
```

## 687: Colorful Constructive

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You have  $n$  colored cubes, the  $i$ -th cube has color  $a_i$ .

You need to distribute all the cubes on shelves. There are a total of  $m$  shelves, the  $i$ -th shelf can hold  $s_i$  cubes. Also,  $s_1 + s_2 + \dots + s_m = n$ .

Suppose on a shelf of size  $k$  there are cubes of colors  $c_1, c_2, \dots, c_k$ , in this order. Then we define the colorfulness of the shelf as the minimum distance between two different cubes of the same color on the shelf. If all the cubes on the shelf have different colors, then the colorfulness is considered to be equal to the size of the shelf, that is, the number  $k$ .

More formally, the colorfulness of  $c_1, c_2, \dots, c_k$  is defined as follows:

If all the colors  $c_1, c_2, \dots, c_k$  are different, the colorfulness is considered to be  $k$ .

Otherwise, the colorfulness is considered to be the smallest integer  $x \geq 1$  such that there exists an index  $i$  ( $1 \leq i \leq k - x$ ) such that  $c_i = c_{i+x}$ .

For each shelf, you are given the minimum required colorfulness, that is, you are given numbers  $d_1, d_2, \dots, d_m$ , which mean that shelf  $i$  must have a colorfulness  $\geq d_i$  for all  $i$ .

Distribute the available cubes among the shelves to ensure the required colorfulness, or report that it is impossible.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<int> a(n), cnt(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10        a[i] -= 1;
11        cnt[a[i]] += 1;
12    }
13    vector<int> s(m), d(m);
14    for (int i = 0; i < m; i++) {
15        cin >> s[i];
16    }
17    for (int i = 0; i < m; i++) {
18        cin >> d[i];
19    }
20    priority_queue<pair<int, int>> q;
21    for (int i = 0; i < n; i++) {
22        q.emplace(cnt[i], i);
23    }
24    vector<vector<int>> ans(m);
25    for (int i = 0; i < m; i++) {
26        for (int j = 0; j < s[i]; j += d[i]) {
27            vector<int> res;
28            int C = min(d[i], s[i] - j);
29            for (int k = 0; k < C; k++) {
30                int x = q.top().second;
31                q.pop();
32                if (cnt[x] == 0) {
33                    cout << -1 << "\n";
34                    return;
35                }
36                res.push_back(x);
37            }
38            for (auto x : res) {
39                ans[i].push_back(x);
40                cnt[x] -= 1;
41                q.emplace(cnt[x], x);
42            }
43        }
44    }
45    for (int i = 0; i < m; i++) {
46        for (auto x : ans[i]) {
47            cnt[x] += 1;
48        }
49        int lim = (s[i] - 1 + d[i]) / d[i];
50        sort(ans[i].begin(), ans[i].end(),
51             [&](int x, int y) {
52                 if ((cnt[x] == lim) != (cnt[y] == lim)) {
53                     return cnt[x] == lim;
54                 }
55                 if ((cnt[x] == lim - 1) != (cnt[y] == lim - 1)) {
56                     return cnt[y] == lim - 1;
57                 }
58             });
59    }
60}
```

```

57             }
58             return x < y;
59         });
60         vector<int> a(s[i]);
61         for (int b = 0, j = 0; b < d[i]; b++) {
62             for (int k = b; k < s[i]; k += d[i]) {
63                 a[k] = ans[i][j++];
64             }
65         }
66         for (int j = 0; j < s[i]; j++) {
67             cout << a[j] + 1 << " \n"[j == s[i] - 1];
68         }
69         for (auto x : ans[i]) {
70             cnt[x] -= 1;
71         }
72     }
73 }
74 int main() {
75     ios::sync_with_stdio(false);
76     cin.tie(nullptr);
77     int t;
78     cin >> t;
79     while (t--) {
80         solve();
81     }
82     return 0;
83 }
```

## 688: Minimum Array

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Given an array  $a$  of length  $n$  consisting of integers. Then the following operation is sequentially applied to it  $q$  times:

Choose indices  $l$  and  $r$  ( $1 \leq l \leq r \leq n$ ) and an integer  $x$ ;

Add  $x$  to all elements of the array  $a$  in the segment  $[l, r]$ . More formally, assign  $a_i := a_i + x$  for all  $l \leq i \leq r$ .

Let  $b_j$  be the array  $a$  obtained after applying the first  $j$  operations ( $0 \leq j \leq q$ ). Note that  $b_0$  is the array  $a$  before applying any operations.

You need to find the lexicographically minimum<sup>†</sup> array among all arrays  $b_j$ .

<sup>†</sup>An array  $x$  is lexicographically smaller than array  $y$  if there is an index  $i$  such that  $x_i < y_i$ , and  $x_j = y_j$  for all  $j < i$ . In other words, for the first index  $i$  where the arrays differ,  $x_i < y_i$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class Info, class Tag>
5 # struct LazySegmentTree;
6 constexpr i64 inf = 1E18;
7 # struct Tag;
8 # struct Info;
9 Info operator+(const Info &a, const Info &b) {
10     return {min(a.min, b.min), max(a.max, b.max)};
11 }
12 void solve() {
13     int n;
14     cin >> n;
15     vector<int> a(n);
16     for (int i = 0; i < n; i++) {
17         cin >> a[i];
18     }
19     int q;
20     cin >> q;
21     vector<vector<pair<int, int>>> add(n);
22     for (int i = 1; i <= q; i++) {
23         int l, r, x;
24         cin >> l >> r >> x;
25         l--;
26         add[l].emplace_back(i, x);
27         if (r < n) {
28             add[r].emplace_back(i, -x);
29         }
30     }
31     LazySegmentTree<Info, Tag> seg(q + 1, Info{0, 0});
32     for (int i = 0; i < n; i++) {
33         for (auto [l, x] : add[i]) {
34             seg.rangeApply(l, q + 1, {x});
35         }
36         i64 v = seg.rangeQuery(0, q + 1).min;
37         cout << v + a[i] << " \n"[i == n - 1];
38         while (true) {
39             int j = seg.findFirst(0, q + 1,
40                 [&](const Info &info) {
41                     return info.max > v;
42                 });
43             if (j == -1) {
44                 break;
45             }
46             seg.modify(j, {});
47         }
48     }
49 }
50 int main() {
51     ios::sync_with_stdio(false);
52     cin.tie(nullptr);
53     int t;
54     cin >> t;
55     while (t--) {
56         solve();
57     }
58     return 0;
59 }
```

## 689: Kuzya and Homework

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Kuzya started going to school. He was given math homework in which he was given an array  $a$  of length  $n$  and an array of symbols  $b$  of length  $n$ , consisting of symbols '\*' and '/'.

Let's denote a path of calculations for a segment  $[l; r]$  ( $1 \leq l \leq r \leq n$ ) in the following way:

Let  $x = 1$  initially. For every  $i$  from  $l$  to  $r$  we will consequently do the following: if  $b_i = '*'$ ,  $x = x * a_i$ , and if  $b_i = '/'$ , then  $x = \frac{x}{a_i}$ . Let's call a path of calculations for the segment  $[l; r]$  a list of all  $x$  that we got during the calculations (the number of them is exactly  $r - l + 1$ ).

For example, let  $a = [7, 12, 3, 5, 4, 10, 9]$ ,  $b = [/ , *, /, /, /, *, *]$ ,  $l = 2$ ,  $r = 6$ , then the path of calculations for that segment is  $[12, 4, 0.8, 0.2, 2]$ .

Let's call a segment  $[l; r]$  simple if the path of calculations for it contains only integer numbers.

Kuzya needs to find the number of simple segments  $[l; r]$  ( $1 \leq l \leq r \leq n$ ). Since he obviously has no time and no interest to do the calculations for each option, he asked you to write a program to get to find that number!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 vector<int> minp, primes;
5 void sieve(int n) {
6     minp.assign(n + 1, 0);
7     primes.clear();
8     for (int i = 2; i <= n; i++) {
9         if (minp[i] == 0) {
10             minp[i] = i;
11             primes.push_back(i);
12         }
13         for (auto p : primes) {
14             if (i * p > n) {
15                 break;
16             }
17             minp[i * p] = p;
18             if (p == minp[i]) {
19                 break;
20             }
21         }
22     }
}

```

```

23 }
24 constexpr int V = 1E6;
25 int main() {
26     ios::sync_with_stdio(false);
27     cin.tie(nullptr);
28     int n;
29     cin >> n;
30     vector<int> a(n);
31     for (int i = 0; i < n; i++) {
32         cin >> a[i];
33     }
34     sieve(V);
35     string b;
36     cin >> b;
37     i64 ans = 0;
38     vector<vector<int>> stk(V + 1);
39     vector<int> m(n);
40     iota(m.begin(), m.end(), 0);
41     for (int i = n - 1; i >= 0; i--) {
42         int x = a[i];
43         while (x > 1) {
44             int p = minp[x];
45             x /= p;
46             if (b[i] == '/') {
47                 stk[p].push_back(i);
48             } else if (!stk[p].empty()) {
49                 m[stk[p].back()] = min(m[stk[p].back()], i);
50                 stk[p].pop_back();
51             }
52         }
53     }
54     for (int p = 1; p <= V; p++) {
55         for (auto i : stk[p]) {
56             m[i] = -1;
57         }
58     }
59     vector<int> s;
60     for (int i = n - 1; i >= 0; i--) {
61         while (!s.empty() && m[i] <= m[s.back()]) {
62             s.pop_back();
63         }
64         if (m[i] == i) {
65             ans += (s.empty() ? n : s.back()) - i;
66         }
67         s.push_back(i);
68     }
69     cout << ans << "\n";
70     return 0;
71 }

```

## 690: Frequency Queries

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Petya has a rooted tree with an integer written on each vertex. The vertex 1 is the root. You are to

answer some questions about the tree.

A tree is a connected graph without cycles. A rooted tree has a special vertex called the root. The parent of a node  $v$  is the next vertex on the shortest path from  $v$  to the root.

Each question is defined by three integers  $v$ ,  $l$ , and  $k$ . To get the answer to the question, you need to perform the following steps:

First, write down the sequence of all integers written on the shortest path from the vertex  $v$  to the root (including those written in the  $v$  and the root).

Count the number of times each integer occurs. Remove all integers with less than  $l$  occurrences.

Replace the sequence, removing all duplicates and ordering the elements by the number of occurrences in the original list in increasing order. In case of a tie, you can choose the order of these elements arbitrary.

The answer to the question is the  $k$ -th number in the remaining sequence. Note that the answer is not always uniquely determined, because there could be several orderings. Also, it is possible that the length of the sequence on this step is less than  $k$ , in this case the answer is  $-1$ .

For example, if the sequence of integers on the path from  $v$  to the root is  $[2, 2, 1, 7, 1, 1, 4, 4, 4, 4]$ ,  $l = 2$  and  $k = 2$ , then the answer is  $1$ .

Please answer all questions about the tree.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, q;
6     cin >> n >> q;
7     vector<int> a(n), cnt(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10        a[i]--;
11    }
12    vector<int> p(n), invp(n);
13    iota(p.begin(), p.end(), 0);
14    iota(inv.p.begin(), invp.end(), 0);
15    vector<int> pre(n + 1, n);
16    pre[0] = 0;
17    vector<vector<int>> adj(n);
18    for (int i = 1; i < n; i++) {
19        int p;
20        cin >> p;
21        p--;
22        adj[p].push_back(i);
23    }
```

```

24     vector<int> ans(q);
25     vector<vector<array<int, 3>>> qry(n);
26     for (int i = 0; i < q; i++) {
27         int v, l, k;
28         cin >> v >> l >> k;
29         v--, k--;
30         qry[v].push_back({l, k, i});
31     }
32     auto dfs = [&](auto self, int x) -> void {
33         int c = a[x];
34         int q = invp[c];
35         swap(invp[p[pre[cnt[c] + 1] - 1]], invp[c]);
36         swap(p[pre[cnt[c] + 1] - 1], p[q]);
37         pre[cnt[c] + 1]--;
38         cnt[c]++;
39         for (auto [l, k, i] : qry[x]) {
40             if (pre[l] + k >= n) {
41                 ans[i] = -1;
42             } else {
43                 ans[i] = p[pre[l] + k] + 1;
44             }
45         }
46         for (auto y : adj[x]) {
47             self(self, y);
48         }
49         cnt[c]--;
50         pre[cnt[c] + 1]++;
51         swap(p[pre[cnt[c] + 1] - 1], p[q]);
52         swap(invp[p[pre[cnt[c] + 1] - 1]], invp[c]);
53     };
54     dfs(dfs, 0);
55     for (int i = 0; i < q; i++) {
56         cout << ans[i] << " \n"[i == q - 1];
57     }
58 }
59 int main() {
60     ios::sync_with_stdio(false);
61     cin.tie(nullptr);
62     int t;
63     cin >> t;
64     while (t--) {
65         solve();
66     }
67     return 0;
68 }
```

## 691: Madoka and the Sixth-graders

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

After the most stunning success with the fifth-graders, Madoka has been trusted with teaching the sixth-graders.

There's  $n$  single-place desks in her classroom. At the very beginning Madoka decided that the student

number  $b_i$  ( $1 \leq b_i \leq n$ ) will sit at the desk number  $i$ . Also there's an infinite line of students with numbers  $n+1, n+2, n+3, \dots$  waiting at the door with the hope of being able to learn something from the Madoka herself. Pay attention that each student has his unique number.

After each lesson, the following happens in sequence.

The student sitting at the desk  $i$  moves to the desk  $p_i$ . All students move simultaneously.

If there is more than one student at a desk, the student with the lowest number keeps the place, and the others are removed from the class forever.

For all empty desks in ascending order, the student from the lowest number from the outside line occupies the desk.

Note that in the end there is exactly one student at each desk again. It is guaranteed that the numbers  $p$  are such that at least one student is removed after each lesson. Check out the explanation to the first example for a better understanding.

After several (possibly, zero) lessons the desk  $i$  is occupied by student  $a_i$ . Given the values  $a_1, a_2, \dots, a_n$  and  $p_1, p_2, \dots, p_n$ , find the lexicographically smallest suitable initial seating permutation  $b_1, b_2, \dots, b_n$ .

The permutation is an array of  $n$  different integers from 1 up to  $n$  in any order. For example, [2, 3, 1, 5, 4] is a permutation, but [1, 2, 2] is not (2 occurs twice). [1, 3, 4] is not a permutation either ( $n = 3$  but there's 4 in the array).

For two different permutations  $a$  and  $b$  of the same length,  $a$  is lexicographically less than  $b$  if in the first position where  $a$  and  $b$  differ, the permutation  $a$  has a smaller element than the corresponding element in  $b$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n;
9     cin >> n;
10    vector<int> p(n);
11    for (int i = 0; i < n; i++) {
12        cin >> p[i];
13        p[i]--;
14    }
15    vector<int> a(n);
16    for (int i = 0; i < n; i++) {

```

```

17         cin >> a[i];
18     }
19     vector<int> b(n, -1);
20     int change = 0;
21     vector<int> deg(n);
22     for (auto x : p) {
23         deg[x]++;
24     }
25     change = count(deg.begin(), deg.end(), 0);
26     int k = (*max_element(a.begin(), a.end()) - n) / change;
27     const int logk = __lg(k);
28     vector<int> pk(n), pw = p;
29     iota(pk.begin(), pk.end(), 0);
30     while (k > 0) {
31         if (k % 2 == 1) {
32             auto tmp = pk;
33             for (int i = 0; i < n; i++) {
34                 pk[i] = tmp[pw[i]];
35             }
36         }
37         k /= 2;
38         auto tmp = pw;
39         for (int i = 0; i < n; i++) {
40             pw[i] = tmp[tmp[i]];
41         }
42     }
43     // for (int i = 0; i < n; i++) {
44     //     cerr << pk[i] << " \n"[i == n - 1];
45     // }
46     vector<bool> vis(n);
47     DSU dsu(n + 1);
48     for (int i = 0; i < n; i++) {
49         if (!vis[pk[i]]) {
50             vis[pk[i]] = true;
51             b[i] = a[pk[i]];
52             dsu.merge(b[i], b[i] - 1);
53         }
54     }
55     for (int i = 0; i < n; i++) {
56         if (b[i] != -1) {
57             continue;
58         }
59         int x = dsu.find(a[pk[i]] - 1);
60         b[i] = x + 1;
61         dsu.merge(x + 1, x);
62     }
63     for (int i = 0; i < n; i++) {
64         cout << b[i] << " \n"[i == n - 1];
65     }
66     return 0;
67 }
```

## 692: Willem, Chtholly and Seniorious

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output
- Willem...
- What's the matter?
- It seems that there's something wrong with Seniorious...
- I'll have a look...

Seniorious is made by linking special talismans in particular order.

After over 500 years, the carillon is now in bad condition, so Willem decides to examine it thoroughly.

Seniorious has  $n$  pieces of talisman. Willem puts them in a line, the  $i$ -th of which is an integer  $a_i$ .

In order to maintain it, Willem needs to perform  $m$  operations.

There are four types of operations:

1  $l \ r \ x$ : For each  $i$  such that  $l \leq i \leq r$ , assign  $a_i + x$  to  $a_i$ .

2  $l \ r \ x$ : For each  $i$  such that  $l \leq i \leq r$ , assign  $x$  to  $a_i$ .

3  $l \ r \ x$ : Print the  $x$ -th smallest number in the index range  $[l, r]$ , i.e. the element at the  $x$ -th position if all the elements  $a_i$  such that  $l \leq i \leq r$  are taken and sorted into an array of non-decreasing integers. It's guaranteed that  $1 \leq x \leq r - l + 1$ .

4  $l \ r \ x \ y$ : Print the sum of the  $x$ -th power of  $a_i$  such that  $l \leq i \leq r$ , modulo  $y$ , i.e. .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int power(int a, int b, int p) {
5     int res = 1 % p;
6     for (; b; b /= 2, a = 1LL * a * a % p) {
7         if (b % 2) {
8             res = 1LL * res * a % p;
9         }
10    }
11    return res;
12 }
13 int main() {
14     ios::sync_with_stdio(false);
15     cin.tie(nullptr);
16     int n, m, seed, vmax;
17     cin >> n >> m >> seed >> vmax;
18     auto rnd = [&]() {
19         int ret = seed;
20         seed = (seed * 7LL + 13) % 1000000007;
21         return ret;
22     };
23     vector<int> a(n);
24     map<int, i64> f;
```

```

25     for (int i = 0; i < n; i++) {
26         a[i] = rnd() % vmax + 1;
27         f[i] = a[i];
28     }
29     f[n] = -1;
30     auto split = [&](int i) {
31         auto it = prev(f.upper_bound(i));
32         if (it->first != i) {
33             f[i] = it->second;
34         }
35     };
36     for (int i = 0; i < m; i++) {
37         int op = rnd() % 4 + 1;
38         int l = rnd() % n + 1;
39         int r = rnd() % n + 1;
40         if (l > r) {
41             swap(l, r);
42         }
43         l--;
44         int x;
45         if (op == 3) {
46             x = rnd() % (r - l);
47         } else {
48             x = rnd() % vmax + 1;
49         }
50         split(l);
51         split(r);
52         if (op == 1) {
53             for (auto it = f.find(l); it->first != r; it++) {
54                 it->second += x;
55             }
56         } else if (op == 2) {
57             for (auto it = f.find(l); it->first != r; it = f.erase(it))
58                 ;
59             f[l] = x;
60         } else if (op == 3) {
61             vector<array<i64, 2>> v;
62             for (auto it = f.find(l); it->first != r; it++) {
63                 v.push_back({it->second, next(it)->first - it->first});
64             }
65             sort(v.begin(), v.end());
66             for (auto [a, b] : v) {
67                 if (x < b) {
68                     cout << a << "\n";
69                     break;
70                 }
71                 x -= b;
72             }
73         } else {
74             int y = rnd() % vmax + 1;
75             int ans = 0;
76             for (auto it = f.find(l); it->first != r; it++) {
77                 ans = (ans + 1LL * (next(it)->first - it->first) * power(it->second % y, x
78                                         , y)) % y;
79             }
80             cout << ans << "\n";
81         }
82     }
83     return 0;
84 }
```

## 693: Fast Travel Text Editor

- Time limit: 5 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given a string  $s$ , consisting of lowercase Latin letters.

There's a cursor, initially placed between two adjacent letters of the string. The cursor can't be placed before the first or after the last letter.

In one move, you can perform one of three actions:

move a cursor one position to the left (if that doesn't place it before the first letter);

move a cursor one position to the right (if that doesn't place it after the last letter);

let  $x$  be the letter immediately to the left of the cursor, and  $y$  be the letter immediately to the right of the cursor. Choose any pair of adjacent letters in the string such that the left of them is  $x$  and the right of them is  $y$ , and move the cursor to the position between these two letters.

You are asked  $m$  queries. In each query, you are given two integers  $f$  and  $t$ , which correspond to two valid positions of the cursor in the string. In response to the query, you have to calculate the minimum number of operations required to move the cursor from position  $f$  to position  $t$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     string s;
8     cin >> s;
9     int n = s.size();
10    vector<int> p[26][26];
11    for (int i = 0; i < n - 1; i++) {
12        p[s[i] - 'a'][s[i + 1] - 'a'].push_back(i);
13    }
14    vector<vector<int>> dis(26, vector<int>(n - 1, -1));
15    for (int a = 0; a < 26; a++) {
16        for (int b = 0; b < 26; b++) {
17            if (!p[a][b].empty()) {
18                queue<int> q;
19                bool vis[26][26] = {};
20                vis[a][b] = true;
21                for (auto i : p[a][b]) {
22                    dis[a][b][i] = 0;

```

```

23             q.push(i);
24         }
25     while (!q.empty()) {
26         int x = q.front();
27         q.pop();
28         if (x > 0 && dis[a][b][x - 1] == -1) {
29             dis[a][b][x - 1] = dis[a][b][x] + 1;
30             q.push(x - 1);
31         }
32         if (x < n - 2 && dis[a][b][x + 1] == -1) {
33             dis[a][b][x + 1] = dis[a][b][x] + 1;
34             q.push(x + 1);
35         }
36         int l = s[x] - 'a', r = s[x + 1] - 'a';
37         if (!vis[l][r]) {
38             vis[l][r] = true;
39             for (auto i : p[l][r]) {
40                 if (dis[a][b][i] == -1) {
41                     dis[a][b][i] = dis[a][b][x] + 1;
42                     q.push(i);
43                 }
44             }
45         }
46     }
47 }
48 }
49 int m;
50 cin >> m;
51 for (int i = 0; i < m; i++) {
52     int x, y;
53     cin >> x >> y;
54     x--, y--;
55     int ans = abs(x - y);
56     for (int a = 0; a < 26; a++) {
57         for (int b = 0; b < 26; b++) {
58             if (!p[a][b].empty()) {
59                 ans = min(ans, dis[a][b][x] + dis[a][b][y] + 1);
60             }
61         }
62     }
63     cout << ans << "\n";
64 }
65 return 0;
66 }
```

## 694: Rollbacks (Hard Version)

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is a hard version of this problem. The only difference between the versions is that you have to solve the hard version in online mode. You can make hacks only if both versions of the problem are solved.

You have an array  $a$ , which is initially empty. You need to process queries of the following types:

- $x$  - add the integer  $x$  to the end of the array  $a$ .
- $k$  - remove the last  $k$  numbers from the array  $a$ .

! - roll back the last active change (i.e., make the array  $a$  the way it was before the change). In this problem, only queries of the first two types (+ and -) are considered as changes.

? - find the number of distinct numbers in the array  $a$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int N = 1 << 20;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int q;
9     cin >> q;
10    vector<int> pos(N, q);
11    int n = 0;
12    vector<array<int, 5>> a;
13    vector<int> b(q), c(q + 1);
14    c[q] = n;
15    vector<int> s(q + 1);
16    for (int i = 0; i < q; i++) {
17        char o;
18        cin >> o;
19        if (o == '+') {
20            int x;
21            cin >> x;
22            a.push_back({x, pos[x], b[n], pos[b[n]], s[n + 1]});
23            if (pos[b[n]] == n) {
24                c[pos[b[n]]] -= 1;
25                pos[b[n]] = q;
26                c[pos[b[n]]] += 1;
27            }
28            if (pos[x] > n) {
29                c[pos[x]] -= 1;
30                pos[x] = n;
31                c[pos[x]] += 1;
32            }
33            s[n + 1] = s[n] + c[n];
34            b[n] = x;
35            n += 1;
36        } else if (o == '-') {
37            int k;
38            cin >> k;
39            n -= k;
40            a.push_back({-1, k});
41        } else if (o == '?') {
42            int ans = s[n];
43            cout << ans << endl;
44        } else {
45            auto [x, y, z, w, t] = a.back();
46            a.pop_back();

```

```

47         if (x == -1) {
48             n += y;
49         } else {
50             n -= 1;
51             b[n] = z;
52             s[n + 1] = t;
53             if (pos[x] != y) {
54                 c[pos[x]] -= 1;
55                 pos[x] = y;
56                 c[pos[x]] += 1;
57             }
58             if (pos[z] != w) {
59                 c[pos[z]] -= 1;
60                 pos[z] = w;
61                 c[pos[z]] += 1;
62             }
63         }
64     }
65 }
66 return 0;
67 }
```

### 695: Rollbacks (Easy Version)

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an easy version of this problem. The only difference between the versions is that you have to solve the hard version in online mode. You can make hacks only if both versions of the problem are solved.

You have an array  $a$ , which is initially empty. You need to process queries of the following types:

- $x$  - add the integer  $x$  to the end of the array  $a$ .
- $k$  - remove the last  $k$  numbers from the array  $a$ .

$!$  - roll back the last active change (i.e., make the array  $a$  the way it was before the change). In this problem, only queries of the first two types (+ and -) are considered as changes.

$?$  - find the number of distinct numbers in the array  $a$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template <typename T>
5 # struct Fenwick;
```

```
6  constexpr int N = 1 << 20;
7  int main() {
8      ios::sync_with_stdio(false);
9      cin.tie(nullptr);
10     int q;
11     cin >> q;
12     Fenwick<int> fen(q);
13     vector<int> pos(N, q);
14     int n = 0;
15     vector<array<int, 4>> a;
16     vector<int> b(q);
17     for (int i = 0; i < q; i++) {
18         char o;
19         cin >> o;
20         if (o == '+') {
21             int x;
22             cin >> x;
23             a.push_back({x, pos[x], b[n], pos[b[n]]});
24             if (pos[b[n]] == n) {
25                 fen.add(pos[b[n]], -1);
26                 pos[b[n]] = q;
27                 fen.add(pos[b[n]], 1);
28             }
29             if (pos[x] > n) {
30                 fen.add(pos[x], -1);
31                 pos[x] = n;
32                 fen.add(pos[x], 1);
33             }
34             b[n] = x;
35             n += 1;
36         } else if (o == '-') {
37             int k;
38             cin >> k;
39             n -= k;
40             a.push_back({-1, k});
41         } else if (o == '?') {
42             int ans = fen.sum(n);
43             cout << ans << endl;
44         } else {
45             auto [x, y, z, w] = a.back();
46             a.pop_back();
47             if (x == -1) {
48                 n += y;
49             } else {
50                 n -= 1;
51                 b[n] = z;
52                 if (pos[x] != y) {
53                     fen.add(pos[x], -1);
54                     pos[x] = y;
55                     fen.add(pos[x], 1);
56                 }
57                 if (pos[z] != w) {
58                     fen.add(pos[z], -1);
59                     pos[z] = w;
60                     fen.add(pos[z], 1);
61                 }
62             }
63         }
64     }
65     return 0;
66 }
```

## 696: More Queries to Array...

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You've got an array, consisting of  $n$  integers:  $a_1, a_2, \dots, a_n$ . Your task is to quickly run the queries of two types:

Assign value  $x$  to all elements from  $l$  to  $r$  inclusive. After such query the values of the elements of array  $a_l, a_{l+1}, \dots, a_r$  become equal to  $x$ .

Calculate and print sum , where  $k$  doesn't exceed 5. As the value of the sum can be rather large, you should print it modulo  $1000000007$  ( $10^9 + 7$ ).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 template<class Info, class Tag>
35 # struct LazySegmentTree;

```

```

36 # struct Comb comb;
37 # struct Tag;
38 # struct Info;
39 Info operator+(Info a, Info b) {
40     Info c;
41     c.n = a.n + b.n;
42     for (int i = 0; i <= 5; i++) {
43         c.sum[i] += a.sum[i];
44         for (int j = 0; i + j <= 5; j++) {
45             c.sum[i + j] += b.sum[i] * comb.binom(a.n, j);
46         }
47     }
48     return c;
49 }
50 Z S[6][6];
51 int main() {
52     ios::sync_with_stdio(false);
53     cin.tie(nullptr);
54     for (int i = 0; i <= 5; i++) {
55         S[i][0] = (i == 0);
56         for (int j = 1; j <= i; j++) {
57             S[i][j] = S[i - 1][j] * j + S[i - 1][j - 1];
58         }
59     }
60     int n, m;
61     cin >> n >> m;
62     vector<int> a(n);
63     for (int i = 0; i < n; i++) {
64         cin >> a[i];
65     }
66     LazySegmentTree<Info, Tag> seg(n, Info{1});
67     for (int i = 0; i < n; i++) {
68         seg.rangeApply(i, i + 1, {a[i]});
69     }
70     while (m--) {
71         char op;
72         cin >> op;
73         int l, r, k;
74         cin >> l >> r >> k;
75         l--;
76         if (op == '+') {
77             seg.rangeApply(l, r, {k});
78         } else {
79             auto info = seg.rangeQuery(l, r);
80             Z ans = 0;
81             for (int i = 0; i <= k; i++) {
82                 ans += info.sum[i] * comb.fac(i) * S[k][i];
83             }
84             cout << ans << "\n";
85         }
86     }
87     return 0;
88 }

```

**697: Editorial for Two**

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

Berland Intercollegiate Contest has just finished. Monocarp and Polycarp, as the jury, are going to conduct an editorial. Unfortunately, the time is limited, since they have to finish before the closing ceremony.

There were  $n$  problems in the contest. The problems are numbered from 1 to  $n$ . The editorial for the  $i$ -th problem takes  $a_i$  minutes. Monocarp and Polycarp are going to conduct an editorial for exactly  $k$  of the problems.

The editorial goes as follows. They have a full problemset of  $n$  problems before them, in order. They remove  $n - k$  problems without changing the order of the remaining  $k$  problems. Then, Monocarp takes some prefix of these  $k$  problems (possibly, an empty one or all problems). Polycarp takes the remaining suffix of them. After that, they go to different rooms and conduct editorials for their problems in parallel. So, the editorial takes as much time as the longer of these two does.

Please, help Monocarp and Polycarp to choose the problems and the split in such a way that the editorial finishes as early as possible. Print the duration of the editorial.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    multiset<int> pl, pr;
12    multiset<int> sl, sr;
13    i64 sp = 0, ss = 0;
14    for (int i = 0; i < n; i++) {
15        sr.insert(a[i]);
16    }
17    i64 ans = 1E18;
18    for (int i = 0; i <= n; i++) {
19        if (i) {
20            if (sr.contains(a[i - 1])) {
21                pr.insert(sr.extract(a[i - 1]));
22            } else {
23                ss -= a[i - 1];
24                pr.insert(sl.extract(a[i - 1]));
25            }
26        }
27        while (!pl.empty() && !pr.empty() && *pl.rbegin() > *pr.begin()) {
28            int x = *pl.rbegin();
29            int y = *pr.begin();
30            pr.insert(pl.extract(x));
31        }
32    }
33}
```

```

31         pl.insert(pr.extract(y));
32         sp += y - x;
33     }
34     while (!sl.empty() && !sr.empty() && *sl.rbegin() > *sr.begin()) {
35         int x = *sl.rbegin();
36         int y = *sr.begin();
37         sr.insert(sl.extract(x));
38         sl.insert(sr.extract(y));
39         ss += y - x;
40     }
41     while (pl.size() + sl.size() < k) {
42         if (!pr.empty()) {
43             int x = *pr.begin();
44             pl.insert(pr.extract(x));
45             sp += x;
46         } else {
47             int x = *sr.begin();
48             sl.insert(sr.extract(x));
49             ss += x;
50         }
51     }
52     ans = min(ans, max(sp, ss));
53     while (sp < ss && !pr.empty() && !sl.empty()) {
54         int x = *pr.begin();
55         pl.insert(pr.extract(x));
56         sp += x;
57         int y = *sl.rbegin();
58         sr.insert(sl.extract(y));
59         ss -= y;
60         ans = min(ans, max(sp, ss));
61     }
62     while (ss < sp && !sr.empty() && !pl.empty()) {
63         int x = *sr.begin();
64         sl.insert(sr.extract(x));
65         ss += x;
66         int y = *pl.rbegin();
67         pr.insert(pl.extract(y));
68         sp -= y;
69         ans = min(ans, max(sp, ss));
70     }
71 }
72 cout << ans << "\n";
73 }
74 int main() {
75     ios::sync_with_stdio(false);
76     cin.tie(nullptr);
77     int t;
78     cin >> t;
79     while (t--) {
80         solve();
81     }
82     return 0;
83 }
```

**698: Palindrome Partition**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

A substring is a continuous and non-empty segment of letters from a given string, without any re-orders.

An even palindrome is a string that reads the same backward as forward and has an even length. For example, strings “zz”, “abba”, “abccba” are even palindromes, but strings “codeforces”, “reality”, “aba”, “c” are not.

A beautiful string is an even palindrome or a string that can be partitioned into some smaller even palindromes.

You are given a string  $s$ , consisting of  $n$  lowercase Latin letters. Count the number of beautiful substrings of  $s$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 vector<int> manacher(string s) {
5     string t = "#";
6     for (auto c : s) {
7         t += c;
8         t += '#';
9     }
10    int n = t.size();
11    vector<int> r(n);
12    for (int i = 0, j = 0; i < n; i++) {
13        if (2 * j - i >= 0 && j + r[j] > i) {
14            r[i] = min(r[2 * j - i], j + r[j] - i);
15        }
16        while (i - r[i] >= 0 && i + r[i] < n && t[i - r[i]] == t[i + r[i]]) {
17            r[i] += 1;
18        }
19        if (i + r[i] > j + r[j]) {
20            j = i;
21        }
22    }
23    return r;
24 }
# struct DSU;
25 void solve() {
26     int n;
27     cin >> n;
28     string s;
29     cin >> s;
30     auto r = manacher(s);
31     DSU dsu(n + 1);
32     vector<int> f(n, -1);
33     for (int i = 0; i <= n; i++) {
34         int d = r[2 * i] / 2;
35         for (int l = dsu.find(i - d); l < i; l = dsu.find(l)) {
36             dsu.merge(l + 1, l);
37             f[l] = 2 * i - l;
38         }
39     }
40 }
```

```

41     vector<int> dp(n + 1);
42     i64 ans = 0;
43     for (int i = n - 1; i >= 0; i--) {
44         if (f[i] != -1) {
45             dp[i] = 1 + dp[f[i]];
46             ans += dp[i];
47         }
48     }
49     cout << ans << "\n";
50 }
51 int main() {
52     ios::sync_with_stdio(false);
53     cin.tie(nullptr);
54     int t;
55     cin >> t;
56     while (t--) {
57         solve();
58     }
59     return 0;
60 }
```

## 699: Range Sorting (Hard Version)

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The only difference between this problem and the easy version is the constraints on  $t$  and  $n$ .

You are given an array  $a$ , consisting of  $n$  distinct integers  $a_1, a_2, \dots, a_n$ .

Define the beauty of an array  $p_1, p_2, \dots, p_k$  as the minimum amount of time needed to sort this array using an arbitrary number of range-sort operations. In each range-sort operation, you will do the following:

Choose two integers  $l$  and  $r$  ( $1 \leq l < r \leq k$ ).

Sort the subarray  $p_l, p_{l+1}, \dots, p_r$  in  $r - l$  seconds.

Please calculate the sum of beauty over all subarrays of array  $a$ .

A subarray of an array is defined as a sequence of consecutive elements of the array.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template <typename T>
5 # struct Fenwick;
6 void solve() {
```

```

7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    auto va = a;
14    sort(va.begin(), va.end());
15    for (auto &x : a) {
16        x = lower_bound(va.begin(), va.end(), x) - va.begin();
17    }
18    i64 ans = 0;
19    for (int i = 1; i <= n; i++) {
20        ans += 1LL * (i - 1) * (n - i + 1);
21    }
22    vector<vector<pair<int, int>>> R(n);
23    vector<int> stk{n};
24    Fenwick<int> f1(n), f2(n);
25    for (int i = n - 1; i >= 0; i--) {
26        while (stk.size() > 1 && a[i] <= a[stk.back()]) {
27            int x = stk.back();
28            stk.pop_back();
29            R[i].emplace_back(a[x], stk.back() - x);
30            f2.add(a[x], x - stk.back());
31        }
32        R[i].emplace_back(a[i], i - stk.back());
33        f2.add(a[i], stk.back() - i);
34        stk.push_back(i);
35    }
36    stk = {-1};
37    i64 res = 0;
38    for (int i = 0; i < n; i++) {
39        while (stk.size() > 1 && a[i] >= a[stk.back()]) {
40            int x = stk.back();
41            stk.pop_back();
42            res += 1LL * f2.rangeSum(a[x], n) * (stk.back() - x);
43            f1.add(a[x], stk.back() - x);
44        }
45        res += 1LL * f2.rangeSum(a[i], n) * (i - stk.back());
46        f1.add(a[i], i - stk.back());
47        stk.push_back(i);
48        for (auto [v, x] : R[i]) {
49            res += 1LL * f1.sum(v + 1) * x;
50            f2.add(v, x);
51        }
52        ans -= res;
53    }
54    cout << ans << "\n";
55 }
56 int main() {
57     ios::sync_with_stdio(false);
58     cin.tie(nullptr);
59     int t;
60     cin >> t;
61     while (t--) {
62         solve();
63     }
64     return 0;
65 }
```

## 700: Walk the Runway

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A fashion tour consists of  $m$  identical runway shows in different cities. There are  $n$  models willing to participate in the tour, numbered from 1 to  $n$ . People in different cities have different views on the fashion industry, so they rate each model differently. In particular, people in city  $i$  rate model  $j$  with rating  $r_{i,j}$ .

You are to choose some number of  $k$  models, and their order, let the chosen models have indices  $j_1, j_2, \dots, j_k$  in the chosen order. In each city, these  $k$  models will walk the runway one after another in this order. To make the show exciting, in each city, the ratings of models should be strictly increasing in the order of their performance. More formally, for any city  $i$  and index  $t$  ( $2 \leq t \leq k$ ), the ratings must satisfy  $r_{i,j_{t-1}} < r_{i,j_t}$ .

After all, the fashion industry is all about money, so choosing model  $j$  to participate in the tour profits you  $p_j$  money. Compute the maximum total profit you can make by choosing the models and their order while satisfying all the requirements.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int N = 5E3;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int m, n;
9     cin >> m >> n;
10    vector<int> p(n);
11    for (int i = 0; i < n; i++) {
12        cin >> p[i];
13    }
14    vector r(m, vector<int>(n));
15    vector<bitset<N>> f(n, ~bitset<N>{});
16    vector<int> o;
17    for (int i = 0; i < m; i++) {
18        for (int j = 0; j < n; j++) {
19            cin >> r[i][j];
20        }
21        vector<int> ord(n);
22        iota(ord.begin(), ord.end(), 0);
23        sort(ord.begin(), ord.end(), [&](int x, int y) {
24            return r[i][x] < r[i][y];
25        });
26        for (int j = 0; j < n; j++) {
27            if (r[i][j] > 0) {
28                f[j].set(i);
29            }
30        }
31    }
32    int ans = 0;
33    for (int i = 0; i < n; i++) {
34        if (f[i].count() == m) {
35            ans += p[i];
36        }
37    }
38    cout << ans;
39 }
```

```

11     }
12     o = ord;
13     bitset<N> s{};
14     for (int j = 0, k = 0; j < n; j++) {
15         while (k < j && r[i][ord[k]] < r[i][ord[j]]) {
16             s.set(ord[k]);
17             k++;
18         }
19         f[ord[j]] &= s;
20     }
21 }
22 vector<i64> dp(n);
23 for (auto x : o) {
24     dp[x] = p[x];
25     for (int y = 0; y < n; y++) {
26         if (f[x][y]) {
27             dp[x] = max(dp[x], dp[y] + p[x]);
28         }
29     }
30 }
31 i64 ans = *max_element(dp.begin(), dp.end());
32 cout << ans << "\n";
33 return 0;
34 }
```

## 701: LuoTianyi and XOR-Tree

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

LuoTianyi gives you a tree with values in its vertices, and the root of the tree is vertex 1.

In one operation, you can change the value in one vertex to any non-negative integer.

Now you need to find the minimum number of operations you need to perform to make each path from the root to leaf<sup>†</sup> has a bitwise XOR value of zero.

<sup>†</sup>A leaf in a rooted tree is a vertex that has exactly one neighbor and is not a root.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
```

```
11         cin >> a[i];
12     }
13     vector<vector<int>> adj(n);
14     for (int i = 1; i < n; i++) {
15         int u, v;
16         cin >> u >> v;
17         u--;
18         adj[u].push_back(v);
19         adj[v].push_back(u);
20     }
21     vector<int> dp(n), t(n);
22     vector<set<int>> s(n);
23     function<void(int, int)> dfs = [&](int x, int p) {
24         map<int, int> cnt;
25         if (adj[x].size() == 1 && x) {
26             s[x].insert(0);
27             dp[x] = 1;
28         }
29         for (auto y : adj[x]) {
30             if (y == p) {
31                 continue;
32             }
33             dfs(y, x);
34             dp[x] += dp[y] + 1;
35             if (s[x].size() < s[y].size()) {
36                 swap(s[x], s[y]);
37                 swap(t[x], t[y]);
38             }
39             for (auto z : s[y]) {
40                 z ^= t[x] ^ t[y];
41                 if (s[x].count(z)) {
42                     cnt[z ^ t[x]]++;
43                 } else {
44                     s[x].insert(z);
45                 }
46             }
47         }
48         if (!cnt.empty()) {
49             int mx = 0;
50             for (auto [_, v] : cnt) {
51                 mx = max(mx, v);
52             }
53             dp[x] -= mx + 1;
54             s[x].clear();
55             t[x] = a[x];
56             for (auto [z, v] : cnt) {
57                 if (v == mx) {
58                     s[x].insert(z);
59                 }
60             }
61         } else {
62             dp[x] -= 1;
63             t[x] ^= a[x];
64         }
65     };
66     dfs(0, -1);
67     int ans = dp[0] + !s[0].count(t[0]);
68     cout << ans << "\n";
69     return 0;
70 }
```

## 702: Frogs and mosquitoes

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  frogs sitting on the coordinate axis  $Ox$ . For each frog two values  $x_i, t_i$  are known - the position and the initial length of the tongue of the  $i$ -th frog (it is guaranteed that all positions  $x_i$  are different).  $m$  mosquitoes one by one are landing to the coordinate axis. For each mosquito two values are known  $p_j$  - the coordinate of the position where the  $j$ -th mosquito lands and  $b_j$  - the size of the  $j$ -th mosquito. Frogs and mosquitoes are represented as points on the coordinate axis.

The frog can eat mosquito if mosquito is in the same position with the frog or to the right, and the distance between them is not greater than the length of the tongue of the frog.

If at some moment several frogs can eat a mosquito the leftmost frog will eat it (with minimal  $x_i$ ). After eating a mosquito the length of the tongue of a frog increases with the value of the size of eaten mosquito. It's possible that after it the frog will be able to eat some other mosquitoes (the frog should eat them in this case).

For each frog print two values - the number of eaten mosquitoes and the length of the tongue after landing all mosquitoes and after eating all possible mosquitoes by frogs.

Each mosquito is landing to the coordinate axis only after frogs eat all possible mosquitoes landed before. Mosquitoes are given in order of their landing to the coordinate axis.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class Info, class Tag>
5 # struct LazySegmentTree;
6 constexpr int inf = 1E9+1;
7 # struct Min;
8 Min operator+(Min a, Min b) {
9     return {min(a.x, b.x)};
10 }
11 int main() {
12     ios::sync_with_stdio(false);
13     cin.tie(nullptr);
14     int n, m;
15     cin >> n >> m;
16     vector<int> x(n);
17     vector<i64> t(n);
18     for (int i = 0; i < n; i++) {
19         cin >> x[i] >> t[i];
20     }

```

```

21     vector<int> o(n);
22     iota(o.begin(), o.end(), 0);
23     sort(o.begin(), o.end(), [&](int i, int j) {
24         return x[i] < x[j];
25     });
26     vector<int> p(m), s(m);
27     for (int i = 0; i < m; i++) {
28         cin >> p[i] >> s[i];
29     }
30     auto vp = p;
31     sort(vp.begin(), vp.end());
32     LazySegmentTree<Min, Min> seg(m);
33     for (int i = 0; i < n; i++) {
34         int j = o[i];
35         int l = lower_bound(vp.begin(), vp.end(), x[j]) - vp.begin();
36         int r = lower_bound(vp.begin(), vp.end(), x[j]+t[j]+1) - vp.begin();
37         seg.rangeApply(l, r, {i});
38     }
39     vector<int> cnt(n);
40     multiset<pair<int, int>> S;
41     for (int i = 0; i < m; i++) {
42         int q = lower_bound(vp.begin(), vp.end(), p[i]) - vp.begin();
43         int k = seg.rangeQuery(q, q+1).x;
44         if (k == inf) {
45             S.emplace(p[i], s[i]);
46             continue;
47         }
48         int j = o[k];
49         cnt[j] += 1;
50         t[j] += s[i];
51         auto it = S.lower_bound({x[j], 0});
52         while (it != S.end() && it->first <= x[j] + t[j]) {
53             cnt[j]++;
54             t[j] += it->second;
55             it = S.erase(it);
56         }
57         int l = lower_bound(vp.begin(), vp.end(), x[j]) - vp.begin();
58         int r = lower_bound(vp.begin(), vp.end(), x[j]+t[j]+1) - vp.begin();
59         seg.rangeApply(l, r, {k});
60     }
61     for (int i = 0; i < n; i++) {
62         cout << cnt[i] << " " << t[i] << "\n";
63     }
64     return 0;
65 }
```

### 703: Shooting Gallery

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Berland amusement park shooting gallery is rightly acknowledged as one of the best in the world. Every day the country's best shooters master their skills there and the many visitors compete in clay pigeon shooting to win decent prizes. And the head of the park has recently decided to make an online

version of the shooting gallery. During the elaboration process it turned out that the program that imitates the process of shooting effectively, is needed. To formulate the requirements to the program, the shooting gallery was formally described. A 3D Cartesian system of coordinates was introduced, where the X axis ran across the gallery floor along the line, along which the shooters are located, the Y axis ran vertically along the gallery wall and the positive direction of the Z axis matched the shooting direction. Let's call the XOY plane a shooting plane and let's assume that all the bullets are out of the muzzles at the points of this area and fly parallel to the Z axis. Every clay pigeon can be represented as a rectangle whose sides are parallel to X and Y axes, and it has a positive z-coordinate. The distance between a clay pigeon and the shooting plane is always different for every target. The bullet hits the target if it goes through the inner area or border of the rectangle corresponding to it. When the bullet hits the target, the target falls down vertically into the crawl-space of the shooting gallery and cannot be shot at any more. The targets are tough enough, that's why a bullet can not pierce a target all the way through and if a bullet hits a target it can't fly on. In input the simulator program is given the arrangement of all the targets and also of all the shots in the order of their appearance. The program should determine which target was hit by which shot. If you haven't guessed it yet, you are the one who is to write such a program.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int inf = 1E9;
5 # struct Node;
6 map<pair<int, int>, vector<int>> shoots;
7 void pull(Node *t) {
8     t->minz = min(t->l->minz, t->r->minz);
9 }
10 Node *build(vector<array<int, 2>> pts, int lvl = 0) {
11     Node *t = new Node;
12     for (auto [x, y] : pts) {
13         t->xl = min(t->xl, x);
14         t->xr = max(t->xr, x);
15         t->yl = min(t->yl, y);
16         t->yr = max(t->yr, y);
17     }
18     if (pts.size() == 1) {
19         auto [x, y] = pts[0];
20         t->minz = shoots[{x, y}].back();
21         return t;
22     }
23     int n = pts.size();
24     nth_element(pts.begin(), pts.begin() + n/2, pts.end(), [&](auto a, auto b) {
25         return a[lvl] < b[lvl];
26     });
27     t->l = build(vector(pts.begin(), pts.begin() + n/2), lvl^1);
28     t->r = build(vector(pts.begin() + n/2, pts.end()), lvl^1);
29     pull(t);
30     return t;

```

```

31 }
32 int query(Node *t, int xl, int xr, int yl, int yr) {
33     if (xl > t->xr || xr < t->xl || yl > t->yr || yr < t->yl) {
34         return inf;
35     }
36     if (xl <= t->xl && xr >= t->xr && yl <= t->yl && yr >= t->yr) {
37         return t->minz;
38     }
39     return min(query(t->l, xl, xr, yl, yr), query(t->r, xl, xr, yl, yr));
40 }
41 void del(Node *t, int x, int y) {
42     if (x > t->xr || x < t->xl || y > t->yr || y < t->yl) {
43         return;
44     }
45     if (x <= t->xl && x >= t->xr && y <= t->yl && y >= t->yr) {
46         auto &v = shoots[{x, y}];
47         v.pop_back();
48         t->minz = v.empty() ? inf : v.back();
49         return;
50     }
51     del(t->l, x, y);
52     del(t->r, x, y);
53     pull(t);
54 }
55 int main() {
56     ios::sync_with_stdio(false);
57     cin.tie(nullptr);
58     int n;
59     cin >> n;
60     vector<int> xl(n), xr(n), yl(n), yr(n), z(n);
61     for (int i = 0; i < n; i++) {
62         cin >> xl[i] >> xr[i] >> yl[i] >> yr[i] >> z[i];
63     }
64     int m;
65     cin >> m;
66     vector<array<int, 2>> pts(m);
67     vector<int> x(m), y(m);
68     for (int i = 0; i < m; i++) {
69         cin >> x[i] >> y[i];
70         pts[i] = {x[i], y[i]};
71     }
72     sort(pts.begin(), pts.end());
73     pts.erase(unique(pts.begin(), pts.end()), pts.end());
74     for (int i = m-1; i >= 0; i--) {
75         shoots[{x[i], y[i]}].push_back(i);
76     }
77     auto root = build(pts);
78     vector<int> invz(n);
79     iota(invz.begin(), invz.end(), 0);
80     sort(invz.begin(), invz.end(), [&](int i, int j) {
81         return z[i] < z[j];
82     });
83     vector<int> ans(m);
84     for (int i = 0; i < n; i++) {
85         int j = invz[i];
86         int k = query(root, xl[j], xr[j], yl[j], yr[j]);
87         if (k != inf) {
88             ans[k] = j+1;
89             del(root, x[k], y[k]);
90         }
91     }
92     for (auto x : ans) {
93         cout << x << "\n";
94     }

```

```

95     return 0;
96 }
```

## 704: Don't fear, DravDe is kind

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A motorcade of  $n$  trucks, driving from city Z to city , has approached a tunnel, known as Tunnel of Horror. Among truck drivers there were rumours about monster DravDe, who hunts for drivers in that tunnel. Some drivers fear to go first, others - to be the last, but let's consider the general case. Each truck is described with four numbers:

$v$  - value of the truck, of its passengers and cargo

$c$  - amount of passenger on the truck, the driver included

$l$  - total amount of people that should go into the tunnel before this truck, so that the driver can overcome his fear (if the monster appears in front of the motorcade, he'll eat them first)

$r$  - total amount of people that should follow this truck, so that the driver can overcome his fear (if the monster appears behind the motorcade, he'll eat them first).

Since the road is narrow, it's impossible to escape DravDe, if he appears from one side. Moreover, the motorcade can't be rearranged. The order of the trucks can't be changed, but it's possible to take any truck out of the motorcade, and leave it near the tunnel for an indefinite period. You, as the head of the motorcade, should remove some of the trucks so, that the rest of the motorcade can move into the tunnel and the total amount of the left trucks' values is maximal.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> dp(n);
10    int ans = 0;
11    int t = -1;
12    map<pair<int, int>, pair<int, int>> f;
```

```

13     vector<int> prev(n, -1);
14     for (int i = 0; i < n; i++) {
15         int v, c, l, r;
16         cin >> v >> c >> l >> r;
17         if (l == 0) {
18             dp[i] = v;
19         } else if (f.count({l, r + c})) {
20             dp[i] = v + f[{l, r + c}].first;
21             prev[i] = f[{l, r + c}].second;
22         } else {
23             dp[i] = -1E9 - 1;
24         }
25         if (dp[i] >= 0) {
26             f[{l + c, r}] = max(f[{l + c, r}], pair{dp[i], i});
27         }
28         if (dp[i] > ans && r == 0) {
29             ans = dp[i];
30             t = i;
31         }
32     }
33     vector<int> a;
34     for (int i = t; i != -1; i = prev[i]) {
35         a.push_back(i);
36     }
37     reverse(a.begin(), a.end());
38     cout << a.size() << "\n";
39     for (auto x : a) {
40         cout << x + 1 << " \n"[x == a.back()];
41     }
42     return 0;
43 }
```

## dp

### 705: Construct Tree

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array of integers  $l_1, l_2, \dots, l_n$  and an integer  $d$ . Is it possible to construct a tree satisfying the following three conditions?

The tree contains  $n + 1$  nodes.

The length of the  $i$ -th edge is equal to  $l_i$ .

The (weighted) diameter of the tree is equal to  $d$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int N = 2001;
5 void solve() {
6     int n, d;
7     cin >> n >> d;
8     vector<int> l(n);
9     for (int i = 0; i < n; i++) {
10         cin >> l[i];
11     }
12     sort(l.begin(), l.end());
13     vector<bitset<N>> suf(n + 1);
14     suf[n].set(0);
15     for (int i = n - 1; i >= 0; i--) {
16         suf[i] = suf[i + 1] | suf[i + 1] << l[i];
17     }
18     int sum = accumulate(l.begin(), l.end(), 0);
19     if (sum == d) {
20         cout << "Yes\n";
21         return;
22     }
23     vector<bitset<N>> dp(d + 1);
24     dp[0].set(0);
25     for (int i = 0; i < n; i++) {
26         for (int j = d; j >= 0; j--) {
27             dp[j] |= dp[j] << l[i];
28             if (j >= l[i]) {
29                 dp[j] |= dp[j - l[i]];
30             }
31         }
32         sum -= l[i];
33         if (sum <= d) {
34             for (int x = 0; x <= d - sum; x++) {
35                 int y = d - sum - x;
36                 if (dp[x][y]) {
37                     int j = 0;
38                     if (x < l[i]) {
39                         j = suf[i + 1]._Find_next(l[i] - x - 1);
40                         if (j == N) {
41                             continue;
42                         }
43                     }
44                     if (y + sum - j >= l[i]) {
45                         cout << "Yes\n";
46                         return;
47                     }
48                 }
49             }
50         }
51     }
52     cout << "No\n";
53 }
54 int main() {
55     ios::sync_with_stdio(false);
56     cin.tie(nullptr);
57     int t;
58     cin >> t;
59     while (t--) {
60         solve();
61     }
62     return 0;
63 }
```

## 706: Small Permutation Problem (Hard Version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In the easy version, the  $a_i$  are in the range  $[0, n]$ ; in the hard version, the  $a_i$  are in the range  $[-1, n]$  and the definition of good permutation is slightly different. You can make hacks only if all versions of the problem are solved.

You are given an integer  $n$  and an array  $a_1, a_2, \dots, a_n$  of integers in the range  $[-1, n]$ .

A permutation  $p_1, p_2, \dots, p_n$  of  $[1, 2, \dots, n]$  is good if, for each  $i$ , the following condition is true:

if  $a_i \neq -1$ , the number of values  $\leq i$  in  $[p_1, p_2, \dots, p_i]$  is exactly  $a_i$ .

Count the good permutations of  $[1, 2, \dots, n]$ , modulo 998 244 353.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 # struct Comb comb;
```

```

35 void solve() {
36     int n;
37     cin >> n;
38     vector<int> a(n);
39     for (int i = 0; i < n; i++) {
40         cin >> a[i];
41     }
42     int lst = 0, lstd = 0;
43     Z ans = 1;
44     auto work = [&](int x, int y) {
45         assert(x >= lst);
46         if (y < lstd || ans == 0 || x < y) {
47             ans = 0;
48             return;
49         }
50         Z res = 0;
51         for (int i = 0; i <= y - lstd; i++) {
52             res += comb.binom(x - lst, i) * comb.binom(lst - lstd, i) * comb.fac(i)
53             * comb.binom(x - lst, y - lstd - i) * comb.binom(x - lstd - i, y - lstd -
54                 i) * comb.fac(y - lstd - i);
55         }
56         ans *= res;
57         lst = x;
58         lstd = y;
59     };
60     for (int i = 0; i < n; i++) {
61         if (a[i] != -1) {
62             work(i + 1, a[i]);
63         }
64     }
65     work(n, n);
66     cout << ans << "\n";
67 }
68 int main() {
69     ios::sync_with_stdio(false);
70     cin.tie(nullptr);
71     int t;
72     cin >> t;
73     while (t--) {
74         solve();
75     }
76     return 0;
77 }
```

## 707: One-X

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

In this sad world full of imperfections, ugly segment trees exist.

A segment tree is a tree where each node represents a segment and has its number. A segment tree for an array of  $n$  elements can be built in a recursive manner. Let's say function  $\text{build}(v, l, r)$  builds the segment tree rooted in the node with number  $v$  and it corresponds to the segment  $[l, r]$ .

Now let's define  $\text{build}(v, l, r)$ :

If  $l = r$ , this node  $v$  is a leaf so we stop adding more edges

Else, we add the edges  $(v, 2v)$  and  $(v, 2v + 1)$ . Let  $m = \lfloor \frac{l+r}{2} \rfloor$ . Then we call  $\text{build}(2v, l, m)$  and  $\text{build}(2v + 1, m + 1, r)$ .

So, the whole tree is built by calling  $\text{build}(1, 1, n)$ .

Now Ibt will construct a segment tree for an array with  $n$  elements. He wants to find the sum of  $\text{lca}^\dagger(S)$ , where  $S$  is a non-empty subset of leaves. Notice that there are exactly  $2^n - 1$  possible subsets. Since this sum can be very large, output it modulo 998 244 353.

$\dagger \text{lca}(S)$  is the number of the least common ancestor for the nodes that are in  $S$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 map<i64, array<Z, 3>> f;
35 array<Z, 3> get(i64 n) {
36     if (f.count(n)) {
37         return f[n];
38     }
39     if (n == 1) {
40         return {1, 0, 1};
41     }

```

```

42     array<Z, 3> &res = f[n];
43     auto [kl, bl, cl] = get((n + 1) / 2);
44     auto [kr, br, cr] = get(n / 2);
45     res[0] += kl * 2;
46     res[1] += bl;
47     res[0] += kr * 2;
48     res[1] += br + kr;
49     res[0] += cl * cr;
50     res[2] += cl * cr + cl + cr;
51     return res;
52 }
53 void solve() {
54     i64 n;
55     cin >> n;
56     auto [k, b, c] = get(n);
57     Z ans = k + b;
58     cout << ans << "\n";
59 }
60 int main() {
61     ios::sync_with_stdio(false);
62     cin.tie(nullptr);
63     int t;
64     cin >> t;
65     while (t--) {
66         solve();
67     }
68     return 0;
69 }
```

## 708: Maximum And Queries (hard version)

- Time limit: 7 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. The only difference between the two versions is the constraint on  $n$  and  $q$ , the memory and time limits. You can make hacks only if all versions of the problem are solved.

Theofanis really likes to play with the bits of numbers. He has an array  $a$  of size  $n$  and an integer  $k$ . He can make at most  $k$  operations in the array. In each operation, he picks a single element and increases it by 1.

He found the maximum bitwise AND that array  $a$  can have after at most  $k$  operations.

Theofanis has put a lot of work into finding this value and was very happy with his result. Unfortunately, Ada, being the evil person that he is, decided to bully him by repeatedly changing the value of  $k$ .

Help Theofanis by calculating the maximum possible bitwise AND for  $q$  different values of  $k$ . Note that queries are independent.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, q;
8     cin >> n >> q;
9     vector<i64> f(1 << 20);
10    vector<int> a(n);
11    for (int i = 0; i < n; i++) {
12        cin >> a[i];
13    }
14    for (int k = 0; k < 20; k++) {
15        vector<i64> s(1 << (19 - k)), c(1 << (19 - k));
16        for (int i = 0; i < n; i++) {
17            if (~a[i] >> k & 1) {
18                s[a[i] >> (k + 1)] += a[i] & ((1 << k) - 1);
19                c[a[i] >> (k + 1)] += 1;
20            }
21        }
22        for (int i = 1; i < (1 << (19 - k)); i *= 2) {
23            for (int j = 0; j < (1 << (19 - k)); j += 2 * i) {
24                for (int l = 0; l < i; l++) {
25                    s[j + l] += s[i + j + l];
26                    c[j + l] += c[i + j + l];
27                }
28            }
29        }
30        for (int x = 0; x < (1 << 20); x++) {
31            if (x >> k & 1) {
32                int y = x >> (k + 1);
33                f[x] += c[y] * (x & ((1 << (k + 1)) - 1)) - s[y];
34            }
35        }
36    }
37    for (int i = (1 << 20) - 2; i >= 0; i--) {
38        f[i] = min(f[i], f[i + 1]);
39    }
40    while (q--) {
41        i64 k;
42        cin >> k;
43        i64 ans = upper_bound(f.begin(), f.end(), k) - f.begin() - 1;
44        if (k >= f.back()) {
45            ans += (k - f.back()) / n;
46        }
47        cout << ans << "\n";
48    }
49    return 0;
50 }
```

## 709: Fancy Arrays

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Let's call an array  $a$  of  $n$  non-negative integers fancy if the following conditions hold:

at least one from the numbers  $x, x + 1, \dots, x + k - 1$  appears in the array;

consecutive elements of the array differ by at most  $k$  (i.e.  $|a_i - a_{i-1}| \leq k$  for each  $i \in [2, n]$ ).

You are given  $n, x$  and  $k$ . Your task is to calculate the number of fancy arrays of length  $n$ . Since the answer can be large, print it modulo  $10^9 + 7$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 constexpr int N = 40;
35 using Matrix = array<array<Z, N>, N>;
36 Matrix operator*(const Matrix &a, const Matrix &b) {
37     Matrix c;
38     for (int i = 0; i < N; i++) {
39         for (int j = 0; j < N; j++) {
40             for (int k = 0; k < N; k++) {
41                 c[i][k] += a[i][j] * b[j][k];
42             }
43         }
44     }
45     return c;
46 }
47 void solve() {

```

```

48     int n, x, k;
49     cin >> n >> x >> k;
50     Z ans = power(Z(2 * k + 1), n - 1) * (x + k);
51     Matrix m;
52     for (int i = 0; i < x; i++) {
53         for (int j = 0; j < x; j++) {
54             m[i][j] = abs(i - j) <= k;
55         }
56     }
57     n -= 1;
58     Matrix f;
59     for (int i = 0; i < x; i++) {
60         f[i][i] = 1;
61     }
62     for (; n; n /= 2, m = m * m) {
63         if (n % 2) {
64             f = f * m;
65         }
66         for (int i = 0; i < x; i++) {
67             for (int j = 0; j < x; j++) {
68                 ans -= f[i][j];
69             }
70         }
71     }
72     cout << ans << "\n";
73 }
74 int main() {
75     ios::sync_with_stdio(false);
76     cin.tie(nullptr);
77     int t;
78     cin >> t;
79     while (t--) {
80         solve();
81     }
82     return 0;
83 }
```

## 710: Vasya and Maximum Profit

- Time limit: 3.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Vasya got really tired of these credits (from problem F) and now wants to earn the money himself! He decided to make a contest to gain a profit.

Vasya has  $n$  problems to choose from. They are numbered from 1 to  $n$ . The difficulty of the  $i$ -th problem is  $d_i$ . Moreover, the problems are given in the increasing order by their difficulties. The difficulties of all tasks are pairwise distinct. In order to add the  $i$ -th problem to the contest you need to pay  $c_i$  burles to its author. For each problem in the contest Vasya gets  $a$  burles.

In order to create a contest he needs to choose a consecutive subsegment of tasks.

So the total earnings for the contest are calculated as follows:

if Vasya takes problem  $i$  to the contest, he needs to pay  $c_i$  to its author;

for each problem in the contest Vasya gets  $a$  burles;

let  $gap(l, r) = \max_{l \leq i < r} (d_{i+1} - d_i)^2$ . If Vasya takes all the tasks with indices from  $l$  to  $r$  to the contest, he also needs to pay  $gap(l, r)$ . If  $l = r$  then  $gap(l, r) = 0$ .

Calculate the maximum profit that Vasya can earn by taking a consecutive segment of tasks.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, a;
8     cin >> n >> a;
9     vector<int> d(n), c(n);
10    vector<int> b(n - 1);
11    for (int i = 0; i < n; i++) {
12        cin >> d[i] >> c[i];
13        c[i] = a - c[i];
14    }
15    for (int i = 0; i < n - 1; i++) {
16        b[i] = d[i + 1] - d[i];
17    }
18    vector<array<int, 2>> ch(n - 1, {-1, -1});
19    vector<int> stk;
20    for (int i = 0; i < n - 1; i++) {
21        while (!stk.empty() && b[i] > b[stk.back()]) {
22            ch[stk.back()][1] = ch[i][0];
23            ch[i][0] = stk.back();
24            stk.pop_back();
25        }
26        stk.push_back(i);
27    }
28    while (stk.size() > 1) {
29        int x = stk.back();
30        stk.pop_back();
31        ch[stk.back()][1] = x;
32    }
33    i64 ans = 0;
34    for (int i = 0; i < n; i++) {
35        ans = max(ans, 1LL * c[i]);
36    }
37    auto dfs = [&](auto self, int x, int l, int r) -> array<i64, 3> {
38        if (x == -1) {
39            return {c[l], c[l], c[l]};
40        }
41        auto [a1, b1, c1] = self(self, ch[x][0], l, x);
42        auto [a2, b2, c2] = self(self, ch[x][1], x + 1, r);
43        ans = max(ans, c1 + b2 - 1LL * b[x] * b[x]);
44        return {a1 + a2, max(b1, a1 + b2), max(c2, c1 + a2)};
45    };
46    if (n > 1) {

```

```

47         dfs(dfs, stk[0], 0, n - 1);
48     }
49     cout << ans << "\n";
50     return 0;
51 }
```

## 711: I Wanna be the Team Leader

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Monocarp is a team leader in a massive IT company.

There are  $m$  projects his team of programmers has to complete, numbered from 1 to  $m$ . The  $i$ -th project has a difficulty level  $b_i$ .

There are  $n$  programmers in the team, numbered from 1 to  $n$ . The  $j$ -th programmer has a stress tolerance level  $a_j$ .

Monocarp wants to assign the programmers to the projects in such a way that:

each programmer is assigned to no more than one project;

each project has at least one programmer assigned to it;

let  $k$  programmers be assigned to the  $i$ -th project; then all the assigned programmers have to have a stress tolerance level greater than or equal to  $\frac{b_i}{k}$ .

Help Monocarp to find a valid assignment. If there are multiple answers, print any of them.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<array<int, 2>> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i][0];
12        a[i][1] = i;
13    }
14    sort(a.begin(), a.end(), greater());
```

```
15     vector<int> b(m);
16     for (int i = 0; i < m; i++) {
17         cin >> b[i];
18     }
19     vector f(m, vector<int>(n, n + 1));
20     for (int i = 0; i < m; i++) {
21         for (int j = 0; j < n; j++) {
22             int l = j + 1 - ((b[i] - 1) / a[j][0] + 1);
23             if (l >= 0) {
24                 f[i][l] = min(f[i][l], j + 1);
25             }
26         }
27         for (int j = n - 2; j >= 0; j--) {
28             f[i][j] = min(f[i][j], f[i][j + 1]);
29         }
30     }
31     vector<int> dp(1 << m, n + 1);
32     dp[0] = 0;
33     for (int s = 1; s < (1 << m); s++) {
34         for (int i = 0; i < m; i++) {
35             if (s >> i & 1) {
36                 int t = s ^ (1 << i);
37                 if (dp[t] < n) {
38                     dp[s] = min(dp[s], f[i][dp[t]]);
39                 }
40             }
41         }
42     }
43     if (dp.back() <= n) {
44         cout << "YES\n";
45         vector<vector<int>> ans(m);
46         int s = (1 << m) - 1;
47         while (s > 0) {
48             for (int i = 0; i < m; i++) {
49                 if (s >> i & 1) {
50                     int t = s ^ (1 << i);
51                     if (dp[t] < n && f[i][dp[t]] == dp[s]) {
52                         for (int j = dp[t]; j < dp[s]; j++) {
53                             ans[i].push_back(a[j][1] + 1);
54                         }
55                         s = t;
56                         break;
57                     }
58                 }
59             }
60         }
61         for (auto x : ans) {
62             cout << x.size();
63             for (auto y : x) {
64                 cout << " " << y;
65             }
66             cout << "\n";
67         }
68     } else {
69         cout << "NO\n";
70     }
71     return 0;
72 }
```

## 712: Axel and Marston in Bitland

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A couple of friends, Axel and Marston are travelling across the country of Bitland. There are  $n$  towns in Bitland, with some pairs of towns connected by one-directional roads. Each road in Bitland is either a pedestrian road or a bike road. There can be multiple roads between any pair of towns, and may even be a road from a town to itself. However, no pair of roads shares the starting and the destination towns along with their types simultaneously.

The friends are now located in the town 1 and are planning the travel route. Axel enjoys walking, while Marston prefers biking. In order to choose a route diverse and equally interesting for both friends, they have agreed upon the following procedure for choosing the road types during the travel:

The route starts with a pedestrian route.

Suppose that a beginning of the route is written in a string  $s$  of letters P (pedestrian road) and B (biking road). Then, the string is appended to  $s$ , where  $\bar{s}$  stands for the string  $s$  with each character changed to opposite (that is, all pedestrian roads changed to bike roads, and vice versa).

In the first few steps the route will look as follows: P, PB, PBBP, PBBPBPPB, PBBPBPPBBPPBPBBP, and so on.

After that the friends start travelling from the town 1 via Bitlandian roads, choosing the next road according to the next character of their route type each time. If it is impossible to choose the next road, the friends terminate their travel and fly home instead.

Help the friends to find the longest possible route that can be travelled along roads of Bitland according to the road types choosing procedure described above. If there is such a route with more than 1018 roads in it, print -1 instead.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int N = 500;
5 using Matrix = array<bitset<N>, N>;
6 Matrix operator+(const Matrix &a, const Matrix &b) {
7     Matrix c{};
8     for (int i = 0; i < N; i++) {
```

```

9         c[i] = a[i] | b[i];
10    }
11    return c;
12 }
13 Matrix operator*(const Matrix &a, const Matrix &b) {
14     Matrix c{};
15     for (int i = 0; i < N; i++) {
16         for (int j = 0; j < N; j++) {
17             if (a[i][j]) {
18                 c[i] |= b[j];
19             }
20         }
21     }
22     return c;
23 }
24 int main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
27     int n, m;
28     cin >> n >> m;
29     Matrix mat[60][2];
30     for (int i = 0; i < m; i++) {
31         int u, v, t;
32         cin >> u >> v >> t;
33         u--, v--;
34         mat[0][t][u][v] = 1;
35     }
36     i64 ans = 0;
37     constexpr i64 inf = 1E18;
38     for (int i = 1; i < 60; i++) {
39         mat[i][0] = mat[i - 1][0] * mat[i - 1][1];
40         mat[i][1] = mat[i - 1][1] * mat[i - 1][0];
41     }
42     int c = 0;
43     Matrix f{};
44     f[0][0] = 1;
45     for (int i = 59; i >= 0; i--) {
46         if ((f * mat[i][c])[0].any()) {
47             ans += 1LL << i;
48             f = f * mat[i][c];
49             c ^= 1;
50         }
51     }
52     if (ans > inf) {
53         ans = -1;
54     }
55     cout << ans << "\n";
56     return 0;
57 }

```

### 713: Korney Korneevich and XOR (hard version)

- Time limit: 1.5 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

This is a harder version of the problem with bigger constraints.

Korney Korneevich dag up an array  $a$  of length  $n$ . Korney Korneevich has recently read about the operation bitwise XOR, so he wished to experiment with it. For this purpose, he decided to find all integers  $x \geq 0$  such that there exists an increasing subsequence of the array  $a$ , in which the bitwise XOR of numbers is equal to  $x$ .

It didn't take a long time for Korney Korneevich to find all such  $x$ , and he wants to check his result. That's why he asked you to solve this problem!

A sequence  $s$  is a subsequence of a sequence  $b$  if  $s$  can be obtained from  $b$  by deletion of several (possibly, zero or all) elements.

A sequence  $s_1, s_2, \dots, s_m$  is called increasing if  $s_1 < s_2 < \dots < s_m$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    constexpr int V = 8192;
14    vector<bitset<V>> dp(V + 1, bitset<V>{});
15    vector<vector<int>> f(V + 1);
16    for (int i = 0; i <= V; i++) {
17        dp[i][0] = 1;
18        f[i].push_back(0);
19    }
20    for (auto x : a) {
21        auto b = move(f[x]);
22        for (auto y : b) {
23            if (!dp[x][x ^ y]) {
24                for (int j = x; j <= V && !dp[j][x ^ y]; j++) {
25                    dp[j][x ^ y] = 1;
26                    f[j].push_back(x ^ y);
27                }
28            }
29        }
30    }
31    vector<int> ans;
32    for (int i = 0; i < V; i++) {
33        if (dp[V][i]) {
34            ans.push_back(i);
35        }
36    }
37    cout << ans.size() << "\n";
38    for (auto x : ans) {
39        cout << x << " \n"[x == ans.back()];
40    }

```

```

41     return 0;
42 }
```

## 714: Travel Plan

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

During the summer vacation after Zhongkao examination, Tom and Daniel are planning to go traveling.

There are  $n$  cities in their country, numbered from 1 to  $n$ . And the traffic system in the country is very special. For each city  $i$  ( $1 \leq i \leq n$ ), there is

a road between city  $i$  and  $2i$ , if  $2i \leq n$ ;

a road between city  $i$  and  $2i + 1$ , if  $2i + 1 \leq n$ .

Making a travel plan, Daniel chooses some integer value between 1 and  $m$  for each city, for the  $i$ -th city we denote it by  $a_i$ .

Let  $s_{i,j}$  be the maximum value of cities in the simple<sup>†</sup> path between cities  $i$  and  $j$ . The score of the travel plan is  $\sum_{i=1}^n \sum_{j=i}^n s_{i,j}$ .

Tom wants to know the sum of scores of all possible travel plans. Daniel asks you to help him find it. You just need to tell him the answer modulo 998 244 353.

<sup>†</sup> A simple path between cities  $x$  and  $y$  is a path between them that passes through each city at most once.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
```

```
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 map<i64, array<vector<Z>, 2>> mp;
35 array<vector<Z>, 2> dp(i64 n) {
36     if (n == 0) {
37         return {};
38     }
39     if (mp.count(n)) {
40         return mp[n];
41     }
42     int k = __lg(n + 1);
43     i64 l = ((1LL << (k - 1)) - 1) + min(1LL << (k - 1), n - (1LL << k) + 1);
44     i64 r = n - 1 - l;
45     auto dpl = dp(l);
46     auto dpr = dp(r);
47     vector<Z> a(k + 1), b(2 * k + 1);
48     a[0] += 1, b[0] += 1;
49     for (int i = 0; i < dpl[0].size(); i++) {
50         if (dpl[0][i] != 0) {
51             a[i + 1] += dpl[0][i];
52             b[i + 1] += dpl[0][i];
53         }
54     }
55     for (int i = 0; i < dpl[1].size(); i++) {
56         if (dpl[1][i] != 0) {
57             b[i] += dpl[1][i];
58         }
59     }
60     for (int i = 0; i < dpr[0].size(); i++) {
61         if (dpr[0][i] != 0) {
62             a[i + 1] += dpr[0][i];
63             b[i + 1] += dpr[0][i];
64         }
65     }
66     for (int i = 0; i < dpr[1].size(); i++) {
67         if (dpr[1][i] != 0) {
68             b[i] += dpr[1][i];
69         }
70     }
71     for (int i = 0; i < dpl[0].size(); i++) {
72         if (dpl[0][i] != 0) {
73             for (int j = 0; j < dpr[0].size(); j++) {
74                 if (dpr[0][j] != 0) {
75                     b[i + j + 2] += dpl[0][i] * dpr[0][j];
76                 }
77             }
78         }
79     }
}
```

```

80     return mp[n] = {a, b};
81 }
82 void solve() {
83     i64 n;
84     int m;
85     cin >> n >> m;
86     Z ans = 0;
87     auto a = dp(n)[1];
88     vector<Z> pw(a.size() + 1);
89     for (int i = 0; i <= a.size(); i++) {
90         pw[i] = power(Z(m), n - i);
91     }
92     ans += pw[0] * m * accumulate(a.begin(), a.end(), Z(0));
93     for (int v = 1; v < m; v++) {
94         Z p = 1;
95         for (int i = 0; i < a.size(); i++) {
96             p *= v;
97             ans -= a[i] * p * pw[i + 1];
98         }
99     }
100    cout << ans << "\n";
101 }
102 int main() {
103     ios::sync_with_stdio(false);
104     cin.tie(nullptr);
105     int t;
106     cin >> t;
107     while (t--) {
108         solve();
109     }
110     return 0;
111 }
```

## 715: Poachers

- Time limit: 1.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Alice and Bob are two poachers who cut trees in a forest.

A forest is a set of zero or more trees. A tree is a connected graph without cycles. A rooted tree has a special vertex called the root. The parent of a node  $v$  is the next vertex on the shortest path from  $v$  to the root. Children of vertex  $v$  are all nodes for which  $v$  is the parent. A vertex is a leaf if it has no children.

In this problem we define the depth of vertex as number of vertices on the simple path from this vertex to the root. The rank of a tree is the minimum depth among its leaves.

Initially there is a forest of rooted trees. Alice and Bob play a game on this forest. They play alternating turns with Alice going first. At the beginning of their turn, the player chooses a tree from the forest. Then the player chooses a positive cutting depth, which should not exceed the rank of the chosen tree.

Then the player removes all vertices of that tree whose depth is less than or equal to the cutting depth. All other vertices of the tree form a set of rooted trees with root being the vertex with the smallest depth before the cut. All these trees are included in the game forest and the game continues.

A player loses if the forest is empty at the beginning of his move.

You are to determine whether Alice wins the game if both players play optimally.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> p(n);
8     int ans = 0;
9     vector<vector<int>> adj(n);
10    for (int i = 0; i < n; i++) {
11        cin >> p[i];
12        p[i]--;
13        if (p[i] != -1) {
14            adj[p[i]].push_back(i);
15        }
16    }
17    vector<int> cnt(n + 1);
18    int cur = 0;
19    auto dfs = [&](auto self, int x) -> vector<int> {
20        vector<int> a{0};
21        cnt[0]++;
22        for (auto y : adj[x]) {
23            if (a.size() == 1) {
24                cnt[0]--;
25                a = self(self, y);
26            } else {
27                for (auto v : a) {
28                    if (v <= n) {
29                        cnt[v]--;
30                    }
31                }
32                cur = 0;
33                auto b = self(self, y);
34                for (auto v : b) {
35                    if (v <= n) {
36                        cnt[v]--;
37                    }
38                }
39                cur = 0;
40                if (a.size() < b.size()) {
41                    swap(a, b);
42                }
43                a.erase(a.begin(), a.end() - b.size());
44                for (int i = 0; i < a.size(); i++) {
45                    a[i] ^= b[i];
46                }
47                for (auto v : a) {

```

```

48             if (v <= n) {
49                 cnt[v]++;
50             }
51         }
52     }
53     while (cnt[cur] > 0) {
54         cur++;
55     }
56     a.push_back(cur);
57     cnt[cur]++;
58     return move(a);
59 };
60 for (int i = 0; i < n; i++) {
61     if (p[i] == -1) {
62         auto res = dfs(dfs, i);
63         ans ^= res.back();
64         for (auto v : res) {
65             if (v <= n) {
66                 cnt[v]--;
67             }
68         }
69         cur = 0;
70     }
71 }
72 cout << (ans ? "YES" : "NO") << "\n";
73 }
74 int main() {
75     ios::sync_with_stdio(false);
76     cin.tie(nullptr);
77     int t;
78     cin >> t;
79     while (t--) {
80         solve();
81     }
82     return 0;
83 }
84 }
```

## 716: Non-equal Neighbours

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given an array of  $n$  positive integers  $a_1, a_2, \dots, a_n$ . Your task is to calculate the number of arrays of  $n$  positive integers  $b_1, b_2, \dots, b_n$  such that:

$1 \leq b_i \leq a_i$  for every  $i$  ( $1 \leq i \leq n$ ), and

$b_i \neq b_{i+1}$  for every  $i$  ( $1 \leq i \leq n-1$ ).

The number of such arrays can be very large, so print it modulo 998 244 353.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int n;
38     cin >> n;
39     vector<int> a(n);
40     for (int i = 0; i < n; i++) {
41         cin >> a[i];
42     }
43     vector<Z> dp(n + 1);
44     vector<Z> s(n + 1);
45     dp[0] = s[0] = 1;
46     dp[1] = -1;
47     vector<array<int, 2>> ch(n, {-1, -1});
48     vector<int> stk;
49     for (int i = 0; i < n; i++) {
50         while (!stk.empty() && a[i] < a[stk.back()]) {
51             ch[stk.back()][1] = ch[i][0];
52             ch[i][0] = stk.back();
53             stk.pop_back();
54         }
55         stk.push_back(i);
56     }
57     while (stk.size() > 1) {
58         ch[stk[stk.size() - 2]][1] = stk.back();
59         stk.pop_back();
60     }
}
```

```

61     auto solve = [&](auto self, int x, int l, int r) -> void {
62         if (x == -1) {
63             return;
64         }
65         self(self, ch[x][0], l, x);
66         Z v = s[x] - (l ? s[l - 1] : 0);
67         v *= a[x];
68         dp[x + 1] -= v;
69         if (r < n) {
70             dp[r + 1] += v;
71         }
72         dp[x + 1] += dp[x];
73         s[x + 1] = s[x] + dp[x + 1];
74         self(self, ch[x][1], x + 1, r);
75     };
76     solve(solve, stk[0], 0, n);
77     cout << dp[n] * (n % 2 ? -1 : 1) << "\n";
78     return 0;
79 }
```

## 717: Keen Tree Calculation

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

There is a tree of  $N$  vertices and  $N - 1$  edges. The  $i$ -th edge connects vertices  $U_i$  and  $V_i$  and has a length of  $W_i$ .

Chaneka, the owner of the tree, asks you  $Q$  times. For the  $j$ -th question, the following is the question format:

$X_j K_j$  - If each edge that contains vertex  $X_j$  has its length multiplied by  $K_j$ , what is the diameter of the tree?

Notes:

Each of Chaneka's question is independent, which means the changes in edge length do not influence the next questions.

The diameter of a tree is the maximum possible distance between two different vertices in the tree.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 # struct Point;
6 template<class T>
```

```

7  T dot(Point<T> a, Point<T> b) {
8      return a.x * b.x + a.y * b.y;
9  }
10 template<class T>
11 T cross(Point<T> a, Point<T> b) {
12     return a.x * b.y - a.y * b.x;
13 }
14 template<class T>
15 T square(Point<T> p) {
16     return dot(p, p);
17 }
18 template<class T>
19 double length(Point<T> p) {
20     return sqrt(double(square(p)));
21 }
22 long double length(Point<long double> p) {
23     return sqrt(square(p));
24 }
25 template<class T>
26 Point<T> normalize(Point<T> p) {
27     return p / length(p);
28 }
29 template<class T>
30 # struct Line;
31 template<class T>
32 Point<T> rotate(Point<T> a) {
33     return Point(-a.y, a.x);
34 }
35 template<class T>
36 int sgn(Point<T> a) {
37     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
38 }
39 template<class T>
40 bool pointOnLineLeft(Point<T> p, Line<T> l) {
41     return cross(l.b - l.a, p - l.a) > 0;
42 }
43 template<class T>
44 Point<T> lineIntersection(Line<T> l1, Line<T> l2) {
45     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1
        .a - l1.b));
46 }
47 template<class T>
48 bool pointOnSegment(Point<T> p, Line<T> l) {
49     return cross(p - l.a, l.b - l.a) == 0 && min(l.a.x, l.b.x) <= p.x && p.x <= max(l.a.x,
        l.b.x)
50     && min(l.a.y, l.b.y) <= p.y && p.y <= max(l.a.y, l.b.y);
51 }
52 template<class T>
53 bool pointInPolygon(Point<T> a, vector<Point<T>> p) {
54     int n = p.size();
55     for (int i = 0; i < n; i++) {
56         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
57             return true;
58         }
59     }
60     int t = 0;
61     for (int i = 0; i < n; i++) {
62         auto u = p[i];
63         auto v = p[(i + 1) % n];
64         if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
65             t ^= 1;
66         }
67         if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
68             t ^= 1;
69         }
70     }
71     return t == 1;
72 }
```

```

69         }
70     }
71     return t == 1;
72 }
73 // 0 : not intersect
74 // 1 : strictly intersect
75 // 2 : overlap
76 // 3 : intersect at endpoint
77 template<class T>
78 tuple<int, Point<T>, Point<T>> segmentIntersection(Line<T> l1, Line<T> l2) {
79     if (max(l1.a.x, l1.b.x) < min(l2.a.x, l2.b.x)) {
80         return {0, Point<T>(), Point<T>()};
81     }
82     if (min(l1.a.x, l1.b.x) > max(l2.a.x, l2.b.x)) {
83         return {0, Point<T>(), Point<T>()};
84     }
85     if (max(l1.a.y, l1.b.y) < min(l2.a.y, l2.b.y)) {
86         return {0, Point<T>(), Point<T>()};
87     }
88     if (min(l1.a.y, l1.b.y) > max(l2.a.y, l2.b.y)) {
89         return {0, Point<T>(), Point<T>()};
90     }
91     if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
92         if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
93             return {0, Point<T>(), Point<T>()};
94         } else {
95             auto maxx1 = max(l1.a.x, l1.b.x);
96             auto minx1 = min(l1.a.x, l1.b.x);
97             auto maxy1 = max(l1.a.y, l1.b.y);
98             auto miny1 = min(l1.a.y, l1.b.y);
99             auto maxx2 = max(l2.a.x, l2.b.x);
100            auto minx2 = min(l2.a.x, l2.b.x);
101            auto maxy2 = max(l2.a.y, l2.b.y);
102            auto miny2 = min(l2.a.y, l2.b.y);
103            Point<T> p1(max(minx1, minx2), max(miny1, miny2));
104            Point<T> p2(min(maxx1, maxx2), min(maxy1, maxy2));
105            if (!pointOnSegment(p1, l1)) {
106                swap(p1.y, p2.y);
107            }
108            if (p1 == p2) {
109                return {3, p1, p2};
110            } else {
111                return {2, p1, p2};
112            }
113        }
114    }
115    auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
116    auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
117    auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
118    auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
119    if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) || (cp3 < 0
120      && cp4 < 0)) {
121        return {0, Point<T>(), Point<T>()};
122    }
123    Point p = lineIntersection(l1, l2);
124    if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
125        return {1, p, p};
126    } else {
127        return {3, p, p};
128    }
129 template<class T>
130 bool segmentInPolygon(Line<T> l, vector<Point<T>> p) {
131     int n = p.size();

```

```
132     if (!pointInPolygon(l.a, p)) {
133         return false;
134     }
135     if (!pointInPolygon(l.b, p)) {
136         return false;
137     }
138     for (int i = 0; i < n; i++) {
139         auto u = p[i];
140         auto v = p[(i + 1) % n];
141         auto w = p[(i + 2) % n];
142         auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
143         if (t == 1) {
144             return false;
145         }
146         if (t == 0) {
147             continue;
148         }
149         if (t == 2) {
150             if (pointOnSegment(v, l) && v != l.a && v != l.b) {
151                 if (cross(v - u, w - v) > 0) {
152                     return false;
153                 }
154             }
155         } else {
156             if (p1 != u && p1 != v) {
157                 if (pointOnLineLeft(l.a, Line(v, u))
158                     || pointOnLineLeft(l.b, Line(v, u))) {
159                     return false;
160                 }
161             } else if (p1 == v) {
162                 if (l.a == v) {
163                     if (pointOnLineLeft(u, l)) {
164                         if (pointOnLineLeft(w, l)
165                             && pointOnLineLeft(w, Line(u, v))) {
166                             return false;
167                         }
168                     } else {
169                         if (pointOnLineLeft(w, l)
170                             || pointOnLineLeft(w, Line(u, v))) {
171                             return false;
172                         }
173                     }
174                 } else if (l.b == v) {
175                     if (pointOnLineLeft(u, Line(l.b, l.a))) {
176                         if (pointOnLineLeft(w, Line(l.b, l.a))
177                             && pointOnLineLeft(w, Line(u, v))) {
178                             return false;
179                         }
180                     } else {
181                         if (pointOnLineLeft(w, Line(l.b, l.a))
182                             || pointOnLineLeft(w, Line(u, v))) {
183                             return false;
184                         }
185                     }
186                 } else {
187                     if (pointOnLineLeft(u, l)) {
188                         if (pointOnLineLeft(w, Line(l.b, l.a))
189                             || pointOnLineLeft(w, Line(u, v))) {
190                             return false;
191                         }
192                     } else {
193                         if (pointOnLineLeft(w, l)
194                             || pointOnLineLeft(w, Line(u, v))) {
195                             return false;
196                         }
197                     }
198                 }
199             }
200         }
201     }
202 }
```

```
196             }
197         }
198     }
199 }
200 }
201 return true;
202 }
203 template<class T>
204 vector<Point<T>> hp(vector<Line<T>> lines) {
205     sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
206         auto d1 = l1.b - l1.a;
207         auto d2 = l2.b - l2.a;
208         if (sgn(d1) != sgn(d2)) {
209             return sgn(d1) == 1;
210         }
211         return cross(d1, d2) > 0;
212     });
213     deque<Line<T>> ls;
214     deque<Point<T>> ps;
215     for (auto l : lines) {
216         if (ls.empty()) {
217             ls.push_back(l);
218             continue;
219         }
220         while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
221             ps.pop_back();
222             ls.pop_back();
223         }
224         while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
225             ps.pop_front();
226             ls.pop_front();
227         }
228         if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
229             if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
230                 if (!pointOnLineLeft(ls.back().a, l)) {
231                     assert(ls.size() == 1);
232                     ls[0] = l;
233                 }
234                 continue;
235             }
236             return {};
237         }
238         ps.push_back(lineIntersection(ls.back(), l));
239         ls.push_back(l);
240     }
241     while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
242         ps.pop_back();
243         ls.pop_back();
244     }
245     if (ls.size() <= 2) {
246         return {};
247     }
248     ps.push_back(lineIntersection(ls[0], ls.back()));
249     return vector(ps.begin(), ps.end());
250 }
251 using i128 = __int128;
252 using P = Point<i128>;
253 int main() {
254     ios::sync_with_stdio(false);
255     cin.tie(nullptr);
256     int N;
257     cin >> N;
258     vector<vector<pair<int, int>>> adj(N);
```

```

260     for (int i = 1; i < N; i++) {
261         int U, V, W;
262         cin >> U >> V >> W;
263         U--, V--;
264         adj[U].push_back({V, W});
265         adj[V].push_back({U, W});
266     }
267     i64 diameter = 0;
268     vector<int> parent(N, -1);
269     vector<i64> dp(N), up(N);
270     auto dfs1 = [&](auto self, int x) -> void {
271         for (auto [y, w] : adj[x]) {
272             if (y == parent[x]) {
273                 continue;
274             }
275             parent[y] = x;
276             self(self, y);
277             diameter = max(diameter, dp[x] + dp[y] + w);
278             dp[x] = max(dp[x], dp[y] + w);
279         }
280     };
281     dfs1(dfs1, 0);
282     auto dfs2 = [&](auto self, int x) -> void {
283         int deg = adj[x].size();
284         vector<i64> pre(deg), suf(deg);
285         for (int i = 0; i < deg - 1; i++) {
286             auto [y, w] = adj[x][i];
287             pre[i + 1] = max(pre[i], (y == parent[x] ? up[x] : dp[y]) + w);
288         }
289         for (int i = deg - 1; i > 0; i--) {
290             auto [y, w] = adj[x][i];
291             suf[i - 1] = max(suf[i], (y == parent[x] ? up[x] : dp[y]) + w);
292         }
293         for (int i = 0; i < deg; i++) {
294             auto [y, w] = adj[x][i];
295             if (y != parent[x]) {
296                 up[y] = max(pre[i], suf[i]);
297                 self(self, y);
298             }
299         }
300     };
301     dfs2(dfs2, 0);
302     vector<vector<P>> h1(N), h2(N);
303     auto add = [&](auto &h, P p, auto &tmp) {
304         while (h.size() > 1 && cross(h.back() - h[h.size() - 2], p - h.back()) >= 0) {
305             tmp.push_back(h.back());
306             h.pop_back();
307         }
308         h.push_back(p);
309     };
310     auto cmp = [&](const P &a, const P &b) {
311         return a.x < b.x || (a.x == b.x && a.y < b.y);
312     };
313     for (int x = 0; x < N; x++) {
314         vector<P> a;
315         for (auto [y, w] : adj[x]) {
316             a.push_back(P(w, y == parent[x] ? up[x] : dp[y]));
317         }
318         sort(a.begin(), a.end(), cmp);
319         vector<P> b;
320         for (auto p : a) {
321             add(h1[x], p, b);
322         }
323         sort(b.begin(), b.end(), cmp);

```

```

324         for (auto p : b) {
325             add(h2[x], p, a);
326         }
327     }
328     auto get = [&](auto &h, P q) {
329         if (h.empty()) {
330             return -1;
331         }
332         int lo = 0, hi = h.size() - 1;
333         while (lo < hi) {
334             int x = (lo + hi) / 2;
335             if (dot(h[x], q) < dot(h[x + 1], q)) {
336                 lo = x + 1;
337             } else {
338                 hi = x;
339             }
340         }
341         return lo;
342     };
343     int Q;
344     cin >> Q;
345     for (int _ = 0; _ < Q; _++) {
346         int X, K;
347         cin >> X >> K;
348         X--;
349         i64 ans = diameter;
350         vector<i64> a;
351         P q(K, 1);
352         auto k = get(h1[X], q);
353         if (k != -1) {
354             a.push_back(dot(h1[X][k], q));
355             if (k > 0) {
356                 a.push_back(dot(h1[X][k - 1], q));
357             }
358             if (k + 1 < h1[X].size()) {
359                 a.push_back(dot(h1[X][k + 1], q));
360             }
361         }
362         k = get(h2[X], q);
363         if (k != -1) {
364             a.push_back(dot(h2[X][k], q));
365         }
366         sort(a.begin(), a.end(), greater());
367         a.resize(2);
368         ans = max(ans, a[0] + a[1]);
369         cout << ans << "\n";
370     }
371     return 0;
372 }

```

### 718: Mighty Rock Tower

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Pak Chanek comes up with an idea in the midst of boredom to play a game. The game is a rock tower

game. There is a big rock that is used as a base. There are also  $N$  small identical rocks that Pak Chanek will use to build a rock tower with a height of  $N$  above the base rock.

Initially, there are no small rocks that are located above the base rock. In other words, the height of the tower above the base rock is initially 0. In one move, Pak Chanek can place one small rock onto the top of the tower which makes the height of the tower above the base rock increase by 1. Each time Pak Chanek place one small rock, the following will happen after the small rock is placed:

Let's say  $x$  is the height of the tower above the base rock right after the small rock is placed.

There is a probability of  $P_x$  percent that the topmost rock falls.

If  $x \geq 2$  and the topmost rock falls, then there is another probability of  $P_x$  percent that the 2-nd topmost rock also falls.

If  $x \geq 3$  and the 2-nd topmost rock falls, then there is another probability of  $P_x$  percent that the 3-rd topmost rock also falls.

If  $x \geq 4$  and the 3-rd topmost rock falls, then there is another probability of  $P_x$  percent that the 4-th topmost rock also falls.

And so on.

If the tower successfully reaches a height of  $N$  without any rocks falling after that, then the game is ended.

If given an integer array  $[P_1, P_2, \dots, P_N]$ , what is the expected value of the number of moves that Pak Chanek needs to do to end the game? It can be proven that the expected value can be represented as an simple fraction  $\frac{P}{Q}$  where  $Q$  is coprime to 998 244 353. Output the value of  $P \times Q^{-1}$  modulo 998 244 353.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;

```

```

19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int N;
39     cin >> N;
40     vector<int> P(N);
41     for (int i = 0; i < N; i++) {
42         cin >> P[i];
43     }
44     vector<Z> dp(N + 1);
45     array<Z, 100> g{};
46     auto inv = Z(100).inv();
47     for (int i = 0; i < N; i++) {
48         Z p = Z(P[i]) * inv;
49         Z res = 1 + dp[i];
50         res -= g[P[i]] * (1 - p);
51         // for (int j = 1; j <= i; j++) {
52         //     res -= power(p, j) * (1 - p) * dp[i + 1 - j];
53         // }
54         res -= power(p, i + 1) * dp[0];
55         res /= (1 - p);
56         dp[i + 1] = res;
57         for (int j = 0; j < 100; j++) {
58             g[j] = (g[j] + res) * j * inv;
59         }
60     }
61     cout << dp[N] << "\n";
62     return 0;
63 }

```

## 719: Divide, XOR, and Conquer

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array of  $n$  integers  $a_1, a_2, \dots, a_n$ .

In one operation you split the array into two parts: a non-empty prefix and a non-empty suffix. The value of each part is the bitwise XOR of all elements in it. Next, discard the part with the smaller value.

If both parts have equal values, you can choose which one to discard. Replace the array with the remaining part.

The operations are being performed until the length of the array becomes 1. For each  $i$  ( $1 \leq i \leq n$ ), determine whether it is possible to achieve the state when only the  $i$ -th element (with respect to the original numbering) remains.

Formally, you have two numbers  $l$  and  $r$ , initially  $l = 1$  and  $r = n$ . The current state of the array is  $[a_l, a_{l+1}, \dots, a_r]$ .

As long as  $l < r$ , you apply the following operation:

Choose an arbitrary  $k$  from the set  $\{l, l+1, \dots, r-1\}$ . Denote  $x = a_l \oplus a_{l+1} \oplus \dots \oplus a_k$  and  $y = a_{k+1} \oplus a_{k+2} \oplus \dots \oplus a_r$ , where  $\oplus$  denotes the bitwise XOR operation.

If  $x < y$ , set  $l = k + 1$ .

If  $x > y$ , set  $r = k$ .

If  $x = y$ , either set  $l = k + 1$ , or set  $r = k$ .

For each  $i$  ( $1 \leq i \leq n$ ), determine whether it is possible to achieve  $l = r = i$ .

## Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<i64> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<i64> s(n + 1);
12    for (int i = 0; i < n; i++) {
13        s[i + 1] = s[i] ^ a[i];
14    }
15    string ans(n, '0');
16    vector<i64> fl0(n + 1), fl1(n + 1), fr0(n + 1), fr1(n + 1);
17    for (int len = n; len >= 1; len--) {
18        for (int l = 0; l <= n - len; l++) {
19            int r = l + len;
20            int dp;
21            if (len == n) {
22                dp = 1;
23            } else {
24                dp = (fl0[l] & ~s[r]) || (fl1[l] & s[r]) || (fr0[r] & ~s[l]) || (fr1[r] &
25                s[l]);
26            }
27            if (!dp) {
28                continue;
29            }
30            if (len == 1) {
```

```

30         ans[l] = '1';
31     }
32     if (s[l] == s[r]) {
33         fl0[l] |= 1;
34         fl1[l] |= 1;
35         fr0[r] |= 1;
36         fr1[r] |= 1;
37     } else {
38         int k = __lg(s[l] ^ s[r]);
39         if (s[l] >> k & 1) {
40             fl0[l] |= 1LL << k;
41             fr1[r] |= 1LL << k;
42         } else {
43             fl1[l] |= 1LL << k;
44             fr0[r] |= 1LL << k;
45         }
46     }
47 }
48 cout << ans << "\n";
49 }
50 int main() {
51     ios::sync_with_stdio(false);
52     cin.tie(nullptr);
53     int t;
54     cin >> t;
55     while (t--) {
56         solve();
57     }
58     return 0;
59 }
60 }
```

## 720: Strange Calculation and Cats

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Gosha's universe is a table consisting of  $n$  rows and  $m$  columns. Both the rows and columns are numbered with consecutive integers starting with 1. We will use  $(r, c)$  to denote a cell located in the row  $r$  and column  $c$ .

Gosha is often invited somewhere. Every time he gets an invitation, he first calculates the number of ways to get to this place, and only then he goes. Gosha's house is located in the cell  $(1, 1)$ .

At any moment of time, Gosha moves from the cell he is currently located in to a cell adjacent to it (two cells are adjacent if they share a common side). Of course, the movement is possible only if such a cell exists, i.e. Gosha will not go beyond the boundaries of the table. Thus, from the cell  $(r, c)$  he is able to make a move to one of the cells  $(r - 1, c)$ ,  $(r, c - 1)$ ,  $(r + 1, c)$ ,  $(r, c + 1)$ . Also, Gosha can skip a move and stay in the current cell  $(r, c)$ .

Besides the love of strange calculations, Gosha is allergic to cats, so he never goes to the cell that has a cat in it. Gosha knows exactly where and when he will be invited and the schedule of cats travelling along the table. Formally, he has  $q$  records, the  $i$ -th of them has one of the following forms:

1,  $x_i, y_i, t_i$  - Gosha is invited to come to cell  $(x_i, y_i)$  at the moment of time  $t_i$ . It is guaranteed that there is no cat inside cell  $(x_i, y_i)$  at this moment of time.

2,  $x_i, y_i, t_i$  - at the moment  $t_i$  a cat appears in cell  $(x_i, y_i)$ . It is guaranteed that no other cat is located in this cell  $(x_i, y_i)$  at that moment of time.

3,  $x_i, y_i, t_i$  - at the moment  $t_i$  a cat leaves cell  $(x_i, y_i)$ . It is guaranteed that there is cat located in the cell  $(x_i, y_i)$ .

Gosha plans to accept only one invitation, but he has not yet decided, which particular one. In order to make this decision, he asks you to calculate for each of the invitations  $i$  the number of ways to get to the cell  $(x_i, y_i)$  at the moment  $t_i$ . For every invitation, assume that Gosha starts moving from cell  $(1, 1)$  at the moment 1.

Moving between two neighboring cells takes Gosha exactly one unit of time. In particular, this means that Gosha can come into the cell only if a cat sitting in it leaves the moment when Gosha begins his movement from the neighboring cell, and if none of the cats comes to the cell at the time when Gosha is in it.

Two ways to go from cell  $(1, 1)$  to cell  $(x, y)$  at time  $t$  are considered distinct if for at least one moment of time from 1 to  $t$  Gosha's positions are distinct for the two ways at this moment. Note, that during this travel Gosha is allowed to visit both  $(1, 1)$  and  $(x, y)$  multiple times. Since the number of ways can be quite large, print it modulo  $10^9 + 7$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 1000000007;
5 constexpr int N = 20;
6 using Matrix = array<array<int, N>, N>;
7 Matrix operator*(const Matrix &a, const Matrix &b) {
8     Matrix c{};
9     for (int i = 0; i < N; i++) {
10         for (int j = 0; j < N; j++) {
11             for (int k = 0; k < N; k++) {
12                 c[i][j] = (c[i][j] + 1LL * a[i][k] * b[k][j]) % P;
13             }
14         }
15     }
16     return c;
17 }
18 constexpr int dx[] = {0, 0, 0, -1, 1};
19 constexpr int dy[] = {0, -1, 1, 0, 0};
20 int main() {

```

```

21     ios::sync_with_stdio(false);
22     cin.tie(nullptr);
23     int n, m, q;
24     cin >> n >> m >> q;
25     int cur = 1;
26     vector<vector<int>> cat(n, vector<int>(m));
27     Matrix mat{};
28     mat[0][0] = 1;
29     for (int _ = 0; _ < q; _++) {
30         int tp, x, y, t;
31         cin >> tp >> x >> y >> t;
32         x--, y--;
33         Matrix g{};
34         for (int i = 0; i < n; i++) {
35             for (int j = 0; j < m; j++) {
36                 if (!cat[i][j]) {
37                     for (int k = 0; k < 5; k++) {
38                         int nx = i + dx[k];
39                         int ny = j + dy[k];
40                         if (0 <= nx && nx < n && 0 <= ny && ny < m && !cat[nx][ny]) {
41                             g[i * m + j][nx * m + ny] = 1;
42                         }
43                     }
44                 }
45             }
46         }
47         int diff = t - cur;
48         for (; diff; diff /= 2, g = g * g) {
49             if (diff & 1) {
50                 mat = mat * g;
51             }
52         }
53         cur = t;
54         if (tp == 1) {
55             cout << mat[0][x * m + y] << "\n";
56         } else if (tp == 2) {
57             cat[x][y] = 1;
58         } else if (tp == 3) {
59             cat[x][y] = 0;
60         }
61     }
62     return 0;
63 }
```

## 721: Maximum Element

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

One day Petya was solving a very interesting problem. But although he used many optimization techniques, his solution still got Time limit exceeded verdict. Petya conducted a thorough analysis of his program and found out that his function for finding maximum element in an array of  $n$  positive integers was too slow. Desperate, Petya decided to use a somewhat unexpected optimization using

parameter k, so now his function contains the following code:

That way the function iteratively checks array elements, storing the intermediate maximum, and if after k consecutive iterations that maximum has not changed, it is returned as the answer.

Now Petya is interested in fault rate of his function. He asked you to find the number of permutations of integers from 1 to n such that the return value of his function on those permutations is not equal to n. Since this number could be very big, output the answer modulo 109 + 7.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int n, k;
39     cin >> n >> k;
40     vector<Z> dp(n + 1);
41     dp[0] = 1;
42     for (int i = 1; i <= n; i++) {
43         dp[i] = dp[i - 1] * i;
44         if (i >= k + 1) {
45             dp[i] -= dp[i - k - 1] * comb.fact(i - 1) * comb.invfact(i - 1 - k);
46         }
47     }
48     Z ans = 0;

```

```

49     for (int i = k + 1; i <= n; i++) {
50         ans += dp[i - k - 1] * comb.fac(i - 1) * comb.invfac(i - 1 - k) * (comb.fac(n) *
51             comb.invfac(i) - comb.fac(n - 1) * comb.invfac(i - 1));
52     }
53     cout << ans << "\n";
54 }
```

## 722: Connecting Vertices

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  points marked on the plane. The points are situated in such a way that they form a regular polygon (marked points are its vertices, and they are numbered in counter-clockwise order). You can draw  $n - 1$  segments, each connecting any two marked points, in such a way that all points have to be connected with each other (directly or indirectly).

But there are some restrictions. Firstly, some pairs of points cannot be connected directly and have to be connected indirectly. Secondly, the segments you draw must not intersect in any point apart from the marked points (that is, if any two segments intersect and their intersection is not a marked point, then the picture you have drawn is invalid).

How many ways are there to connect all vertices with  $n - 1$  segments? Two ways are considered different iff there exist some pair of points such that a segment is drawn between them in the first way of connection, but it is not drawn between these points in the second one. Since the answer might be large, output it modulo  $10^9 + 7$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
```

```

15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int n;
38     cin >> n;
39     vector a(n, vector<int>(n));
40     for (int i = 0; i < n; i++) {
41         for (int j = 0; j < n; j++) {
42             cin >> a[i][j];
43         }
44     }
45     vector dp(n, vector<Z>(n)), g(n, vector<Z>(n));
46     for (int i = 0; i < n; i++) {
47         dp[i][i] = 1;
48     }
49     for (int l = n - 1; l >= 0; l--) {
50         for (int r = l + 1; r < n; r++) {
51             for (int i = l + 1; i <= r; i++) {
52                 g[l][r] += dp[l][i - 1] * dp[i][r];
53             }
54             for (int j = l + 1; j <= r; j++) {
55                 if (a[l][j]) {
56                     dp[l][r] += g[l][j] * dp[j][r];
57                 }
58             }
59         }
60     }
61     cout << dp[0][n - 1] << "\n";
62     return 0;
63 }

```

### 723: Maximum Monogonosity

- Time limit: 2.5 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  of length  $n$  and an array  $b$  of length  $n$ . The cost of a segment  $[l, r]$ ,  $1 \leq l \leq r \leq n$ ,

is defined as  $|b_l - a_r| + |b_r - a_l|$ .

Recall that two segments  $[l_1, r_1]$ ,  $1 \leq l_1 \leq r_1 \leq n$ , and  $[l_2, r_2]$ ,  $1 \leq l_2 \leq r_2 \leq n$ , are non-intersecting if one of the following conditions is satisfied:  $r_1 < l_2$  or  $r_2 < l_1$ .

The length of a segment  $[l, r]$ ,  $1 \leq l \leq r \leq n$ , is defined as  $r - l + 1$ .

Find the maximum possible sum of costs of non-intersecting segments  $[l_j, r_j]$ ,  $1 \leq l_j \leq r_j \leq n$ , whose total length is equal to  $k$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 1E18;
5 void solve() {
6     int n, k;
7     cin >> n >> k;
8     vector<int> a(n), b(n);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11     }
12     for (int i = 0; i < n; i++) {
13         cin >> b[i];
14     }
15     vector<vector<vector<i64>>> f(2, vector<vector<i64>>(k + 1, {-inf}));
16     vector<i64> g(k + 1, -inf);
17     g[0] = 0;
18     for (int i = 0; i < n; i++) {
19         for (int x = 0; x < 2; x++) {
20             for (int y = 0; y < 2; y++) {
21                 for (int l = k; l >= 0; l--) {
22                     f[x][y][l] = l > 0 ? f[x][y][l - 1] : -inf;
23                 }
24             }
25         }
26         for (int x = 0; x < 2; x++) {
27             for (int y = 0; y < 2; y++) {
28                 for (int l = 0; l < k; l++) {
29                     f[x][y][l + 1] = max(f[x][y][l + 1], g[l] + (x ? -1 : 1) * b[i] + (y ? 1 : -1) * a[i]);
30                 }
31             }
32         }
33         for (int x = 0; x < 2; x++) {
34             for (int y = 0; y < 2; y++) {
35                 for (int l = 0; l <= k; l++) {
36                     g[l] = max(g[l], f[x][y][l] + (x ? 1 : -1) * a[i] + (y ? -1 : 1) * b[i]);
37                 }
38             }
39         }
40     }
41     cout << g[k] << "\n";
42 }
43 int main() {
44     ios::sync_with_stdio(false);
45     cin.tie(nullptr);

```

```

46     int t;
47     cin >> t;
48     while (t--) {
49         solve();
50     }
51     return 0;
52 }
```

## 724: Expected Destruction

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have a set  $S$  of  $n$  distinct integers between 1 and  $m$ .

Each second you do the following steps:

Pick an element  $x$  in  $S$  uniformly at random.

Remove  $x$  from  $S$ .

If  $x + 1 \leq m$  and  $x + 1$  is not in  $S$ , add  $x + 1$  to  $S$ .

What is the expected number of seconds until  $S$  is empty?

Output the answer modulo 1 000 000 007.

Formally, let  $P = 1\,000\,000\,007$ . It can be shown that the answer can be expressed as an irreducible fraction  $\frac{a}{b}$ , where  $a$  and  $b$  are integers and  $b \not\equiv 0 \pmod{P}$ . Output the integer equal to  $a \cdot b^{-1} \pmod{P}$ . In other words, output an integer  $z$  such that  $0 \leq z < P$  and  $z \cdot b \equiv a \pmod{P}$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
```

```

17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int n, m;
38     cin >> n >> m;
39     Z ans = 0;
40     vector<int> a(n);
41     for (int i = 0; i < n; i++) {
42         cin >> a[i];
43         ans += m + 1 - a[i];
44     }
45     vector<Z> dp(m + 1, vector<Z>(m + 1));
46     for (int i = m; i >= 1; i--) {
47         for (int j = m; j >= i; j--) {
48             if (i == j) {
49                 dp[i][j] = m + 1 - i;
50             } else {
51                 dp[i][j] += dp[i + 1][j] * CInv<2, P>;
52                 if (j + 1 <= m) {
53                     dp[i][j] += dp[i][j + 1] * CInv<2, P>;
54                 }
55             }
56         }
57     }
58     for (int i = 1; i < n; i++) {
59         ans -= dp[a[i - 1]][a[i]];
60     }
61     cout << ans << "\n";
62     return 0;
63 }
```

**greedy****725: Alice and Recoloring 1**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The difference between the versions is in the costs of operations. Solution for one version won't work for another!

Alice has a grid of size  $n \times m$ , initially all its cells are colored white. The cell on the intersection of  $i$ -th row and  $j$ -th column is denoted as  $(i, j)$ . Alice can do the following operations with this grid:

Choose any subrectangle containing cell  $(1, 1)$ , and flip the colors of all its cells. (Flipping means changing its color from white to black or from black to white). This operation costs 1 coin.

Choose any subrectangle containing cell  $(1, 1)$ , and flip the colors of all its cells. (Flipping means changing its color from white to black or from black to white).

This operation costs 1 coin.

Choose any subrectangle containing cell  $(n, 1)$ , and flip the colors of all its cells. This operation costs 2 coins.

Choose any subrectangle containing cell  $(n, 1)$ , and flip the colors of all its cells.

This operation costs 2 coins.

Choose any subrectangle containing cell  $(1, m)$ , and flip the colors of all its cells. This operation costs 4 coins.

Choose any subrectangle containing cell  $(1, m)$ , and flip the colors of all its cells.

This operation costs 4 coins.

Choose any subrectangle containing cell  $(n, m)$ , and flip the colors of all its cells. This operation costs 3 coins.

Choose any subrectangle containing cell  $(n, m)$ , and flip the colors of all its cells.

This operation costs 3 coins.

As a reminder, subrectangle is a set of all cells  $(x, y)$  with  $x_1 \leq x \leq x_2, y_1 \leq y \leq y_2$  for some  $1 \leq x_1 \leq x_2 \leq n, 1 \leq y_1 \leq y_2 \leq m$ .

Alice wants to obtain her favorite coloring with these operations. What's the smallest number of coins that she would have to spend? It can be shown that it's always possible to transform the initial grid into any other.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
```

```

6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<string> s(n);
10    for (int i = 0; i < n; i++) {
11        cin >> s[i];
12    }
13    vector a(n, vector<int>(m));
14    for (int i = 0; i < n; i++) {
15        for (int j = 0; j < m; j++) {
16            a[i][j] = (s[i][j] == 'B');
17        }
18    }
19    for (int i = 0; i < n; i++) {
20        for (int j = 0; j < m - 1; j++) {
21            a[i][j] ^= a[i][j + 1];
22        }
23    }
24    for (int i = 0; i < n - 1; i++) {
25        for (int j = 0; j < m; j++) {
26            a[i][j] ^= a[i + 1][j];
27        }
28    }
29    int ans = 0;
30    for (int i = 0; i < n; i++) {
31        for (int j = 0; j < m; j++) {
32            ans += a[i][j];
33        }
34    }
35    int minus = 0;
36    for (int i = 0; i < n - 1; i++) {
37        for (int j = 0; j < m - 1; j++) {
38            if (a[i][j] && a[i][m - 1] && a[n - 1][j] && a[n - 1][m - 1]) {
39                minus = 1;
40            }
41        }
42    }
43    ans -= minus;
44    cout << ans << "\n";
45    return 0;
46 }

```

## 726: Royal Questions

- Time limit: 1.5 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

In a medieval kingdom, the economic crisis is raging. Milk drops fall, Economic indicators are deteriorating every day, money from the treasury disappear. To remedy the situation, King Charles Sunnyface decided make his  $n$  sons-princes marry the brides with as big dowry as possible.

In search of candidates, the king asked neighboring kingdoms, and after a while several delegations arrived with  $m$  unmarried princesses. Receiving guests, Karl learned that the dowry of the  $i$  th princess

is wi of golden coins.

Although the action takes place in the Middle Ages, progressive ideas are widespread in society, according to which no one can force a princess to marry a prince whom she does not like. Therefore, each princess has an opportunity to choose two princes, for each of which she is ready to become a wife. The princes were less fortunate, they will obey the will of their father in the matter of choosing a bride.

Knowing the value of the dowry and the preferences of each princess, Charles wants to play weddings in such a way that the total dowry of the brides of all his sons would be as great as possible. At the same time to marry all the princes or princesses is not necessary. Each prince can marry no more than one princess, and vice versa, each princess can marry no more than one prince.

Help the king to organize the marriage of his sons in the most profitable way for the treasury.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<int> f(n), c(n);
10    iota(f.begin(), f.end(), 0);
11    vector<array<int, 3>> a(m);
12    for (int i = 0; i < m; i++) {
13        int x, y, z;
14        cin >> x >> y >> z;
15        x--, y--;
16        a[i] = {z, x, y};
17    }
18    sort(a.begin(), a.end(), greater());
19    auto find = [&](int x) {
20        while (x != f[x]) {
21            x = f[x] = f[f[x]];
22        }
23        return x;
24    };
25    int ans = 0;
26    for (auto [w, x, y] : a) {
27        x = find(x);
28        y = find(y);
29        if (x == y) {
30            if (c[x] == 0) {
31                c[x] = 1;
32                ans += w;
33            }
34        } else if (c[x] + c[y] < 2) {
35            c[x] += c[y];
36            ans += w;
}

```

```

37         f[y] = x;
38     }
39 }
40 cout << ans << "\n";
41 return 0;
42 }
```

## 727: Two Permutations (Easy Version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the easy version of the problem. The difference between the two versions is that you do not have to minimize the number of operations in this version. You can make hacks only if both versions of the problem are solved.

You have two permutations<sup>†</sup>  $p_1, p_2, \dots, p_n$  (of integers 1 to  $n$ ) and  $q_1, q_2, \dots, q_m$  (of integers 1 to  $m$ ). Initially  $p_i = a_i$  for  $i = 1, 2, \dots, n$ , and  $q_j = b_j$  for  $j = 1, 2, \dots, m$ . You can apply the following operation on the permutations several (possibly, zero) times.

In one operation,  $p$  and  $q$  will change according to the following three steps:

You choose integers  $i, j$  which satisfy  $1 \leq i \leq n$  and  $1 \leq j \leq m$ .

Permutation  $p$  is partitioned into three parts using  $p_i$  as a pivot: the left part is formed by elements  $p_1, p_2, \dots, p_{i-1}$  (this part may be empty), the middle part is the single element  $p_i$ , and the right part is  $p_{i+1}, p_{i+2}, \dots, p_n$  (this part may be empty). To proceed, swap the left and the right parts of this partition. Formally, after this step,  $p$  will become  $p_{i+1}, p_{i+2}, \dots, p_n, p_i, p_1, p_2, \dots, p_{i-1}$ . The elements of the newly formed  $p$  will be reindexed starting from 1.

Perform the same transformation on  $q$  with index  $j$ . Formally, after this step,  $q$  will become  $q_{j+1}, q_{j+2}, \dots, q_m, q_j, q_1, q_2, \dots, q_{j-1}$ . The elements of the newly formed  $q$  will be reindexed starting from 1.

Your goal is to simultaneously make  $p_i = i$  for  $i = 1, 2, \dots, n$ , and  $q_j = j$  for  $j = 1, 2, \dots, m$ .

Find any valid way to achieve the goal using at most 10 000 operations, or say that none exists. Please note that you do not have to minimize the number of operations.

It can be proved that if it is possible to achieve the goal, then there exists a way to do so using at most 10 000 operations.

<sup>†</sup> A permutation of length  $k$  is an array consisting of  $k$  distinct integers from 1 to  $k$  in arbitrary order. For example,  $[2, 3, 1, 5, 4]$  is a permutation, but  $[1, 2, 2]$  is not a permutation (2 appears twice in the

array), and  $[1, 3, 4]$  is also not a permutation ( $k = 3$  but there is 4 in the array).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 array<int, 2> solve(vector<int> a) {
5     int n = a.size();
6     array<int, 2> ans{-1, -1};
7     for (int i = 0; i < n; i++) {
8         int res = 0;
9         auto b = a;
10        while (b[0] != 0) {
11            swap(b[0], b[b[0]]);
12            res++;
13        }
14        for (int j = 0; j < n; j++) {
15            if (b[j] != j) {
16                swap(b[0], b[j]);
17                res++;
18                while (b[0] != 0) {
19                    swap(b[0], b[b[0]]);
20                    res++;
21                }
22            }
23        }
24        if (ans[res % 2] == -1 || ans[res % 2] > res) {
25            ans[res % 2] = res;
26        }
27        for (int j = 0; j < n; j++) {
28            a[j] = (a[j] + 1) % n;
29        }
30    }
31    return ans;
32 }
33 vector<int> get(vector<int> a, int k) {
34     int n = a.size();
35     for (int i = 0; i < n; i++) {
36         int res = 0;
37         auto b = a;
38         vector<int> ans;
39         while (b[0] != 0) {
40             ans.push_back((b[b[0]] - b[0] + n) % n);
41             swap(b[0], b[b[0]]);
42             res++;
43         }
44         for (int j = 0; j < n; j++) {
45             if (b[j] != j) {
46                 ans.push_back((b[j] - b[0] + n) % n);
47                 swap(b[0], b[j]);
48                 res++;
49                 while (b[0] != 0) {
50                     ans.push_back((b[b[0]] - b[0] + n) % n);
51                     swap(b[0], b[b[0]]);
52                     res++;
53                 }
54             }
55         }
56         if (res <= k && (k - res) % 2 == 0) {
57             while (res < k) {
58                 ans.push_back((b[1] - b[0] + n) % n);
59             }
60         }
61     }
62 }
```

```

59             swap(b[0], b[1]);
60             res++;
61         }
62     return ans;
63 }
64 for (int j = 0; j < n; j++) {
65     a[j] = (a[j] + 1) % n;
66 }
67 }
68 assert(false);
69 }
70 int main() {
71     ios::sync_with_stdio(false);
72     cin.tie(nullptr);
73     int n, m;
74     cin >> n >> m;
75     vector<int> a(n + 1), b(m + 1);
76     for (int i = 1; i <= n; i++) {
77         int x;
78         cin >> x;
79         a[x] = i;
80     }
81     for (int i = 1; i <= m; i++) {
82         int x;
83         cin >> x;
84         b[x] = i;
85     }
86     auto [ea, oa] = solve(a);
87     auto [eb, ob] = solve(b);
88     int ans = -1;
89     if (ea != -1 && eb != -1) {
90         ans = max(ea, eb);
91     }
92     if (oa != -1 && ob != -1 && (ans == -1 || ans > max(oa, ob))) {
93         ans = max(oa, ob);
94     }
95     if (ans == -1) {
96         cout << -1 << "\n";
97         return 0;
98     }
99     auto sa = get(a, ans);
100    auto sb = get(b, ans);
101    cout << ans << "\n";
102    for (int i = 0; i < ans; i++) {
103        cout << sa[i] << " " << sb[i] << "\n";
104    }
105    return 0;
106 }

```

## 728: Tenzing and Tree

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Tenzing has an undirected tree of  $n$  vertices.

Define the value of a tree with black and white vertices in the following way. The value of an edge is the absolute difference between the number of black nodes in the two components of the tree after deleting the edge. The value of the tree is the sum of values over all edges.

For all  $k$  such that  $0 \leq k \leq n$ , Tenzing wants to know the maximum value of the tree when  $k$  vertices are painted black and  $n - k$  vertices are painted white.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<vector<int>> adj(n);
10    for (int i = 1; i < n; i++) {
11        int u, v;
12        cin >> u >> v;
13        u--, v--;
14        adj[u].push_back(v);
15        adj[v].push_back(u);
16    }
17    vector<int> ans(n + 1);
18    for (int r = 0; r < n; r++) {
19        vector<int> dep(n), cnt(n);
20        auto dfs = [&](auto self, int x, int p) -> void {
21            cnt[dep[x]]++;
22            for (auto y : adj[x]) {
23                if (y == p) {
24                    continue;
25                }
26                dep[y] = dep[x] + 1;
27                self(self, y, x);
28            }
29        };
30        dfs(dfs, r, -1);
31        int k = 0;
32        int sum = 0;
33        for (int i = 0; i < n; i++) {
34            for (int j = 0; j < cnt[i]; j++) {
35                k++;
36                sum += i;
37                ans[k] = max(ans[k], (n - 1) * k - 2 * sum);
38            }
39        }
40    }
41    for (int i = 0; i <= n; i++) {
42        cout << ans[i] << "\n"[i == n];
43    }
44    return 0;
45 }
```

## 730: Hyperregular Bracket Strings

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an integer  $n$  and  $k$  intervals. The  $i$ -th interval is  $[l_i, r_i]$  where  $1 \leq l_i \leq r_i \leq n$ .

Let us call a regular bracket sequence<sup>†,‡</sup> of length  $n$  hyperregular if for each  $i$  such that  $1 \leq i \leq k$ , the substring  $\overline{s_{l_i} s_{l_i+1} \dots s_{r_i}}$  is also a regular bracket sequence.

Your task is to count the number of hyperregular bracket sequences. Since this number can be really large, you are only required to find it modulo 998 244 353.

<sup>†</sup> A bracket sequence is a string containing only the characters “(” and “)”.

<sup>‡</sup> A bracket sequence is called regular if one can turn it into a valid math expression by adding characters + and 1. For example, sequences  $(())()$ ,  $()$ ,  $((())()$ ) and the empty string are regular, while  $)()$ ,  $((()$ , and  $(())()$  are not.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();

```

```

32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 using u64 = unsigned long long;
36 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
37 void solve() {
38     int n, k;
39     cin >> n >> k;
40     vector<u64> d(n);
41     for (int i = 0; i < k; i++) {
42         int l, r;
43         cin >> l >> r;
44         l--;
45         auto x = rng();
46         d[l] ^= x;
47         if (r < n) {
48             d[r] ^= x;
49         }
50     }
51     for (int i = 1; i < n; i++) {
52         d[i] ^= d[i - 1];
53     }
54     map<u64, int> cnt;
55     for (auto x : d) {
56         cnt[x]++;
57     }
58     Z ans = 1;
59     for (auto [_, x] : cnt) {
60         if (x % 2) {
61             ans = 0;
62         } else {
63             x /= 2;
64             ans *= comb.binom(2 * x, x) - comb.binom(2 * x, x + 1);
65         }
66     }
67     cout << ans << "\n";
68 }
69 int main() {
70     ios::sync_with_stdio(false);
71     cin.tie(nullptr);
72     int t;
73     cin >> t;
74     while (t--) {
75         solve();
76     }
77     return 0;
78 }
```

### 731: Population Size

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Polycarpus develops an interesting theory about the interrelation of arithmetic progressions with just everything in the world. His current idea is that the population of the capital of Berland changes over

time like an arithmetic progression. Well, or like multiple arithmetic progressions.

Polycarpus believes that if he writes out the population of the capital for several consecutive years in the sequence  $a_1, a_2, \dots, a_n$ , then it is convenient to consider the array as several arithmetic progressions, written one after the other. For example, sequence  $(8, 6, 4, 2, 1, 4, 7, 10, 2)$  can be considered as a sequence of three arithmetic progressions  $(8, 6, 4, 2)$ ,  $(1, 4, 7, 10)$  and  $(2)$ , which are written one after another.

Unfortunately, Polycarpus may not have all the data for the  $n$  consecutive years (a census of the population doesn't occur every year, after all). For this reason, some values of  $a_i$  may be unknown. Such values are represented by number  $-1$ .

For a given sequence  $a = (a_1, a_2, \dots, a_n)$ , which consists of positive integers and values  $-1$ , find the minimum number of arithmetic progressions Polycarpus needs to get  $a$ . To get  $a$ , the progressions need to be written down one after the other. Values  $-1$  may correspond to an arbitrary positive integer and the values  $a_i > 0$  must be equal to the corresponding elements of sought consecutive record of the progressions.

Let us remind you that a finite sequence  $c$  is called an arithmetic progression if the difference  $c_{i+1} - c_i$  of any two consecutive elements in it is constant. By definition, any sequence of length 1 is an arithmetic progression.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    vector<int> p;
14    int ans = 1;
15    for (int i = 0, j = 0; i < n; i++) {
16        if (a[i] != -1) {
17            if (p.size() < 2) {
18                p.push_back(i);
19            } else if (a[i] != a[p[0]] + (a[p[1]] - a[p[0]]) / (p[1] - p[0]) * 1LL * (i - p[0])) {
20                ans++;
21                j = i;
22                p = {i};
23            }
24        }
25    }
26}
```

```

25     if (p.size() == 2) {
26         if ((a[p[1]] - a[p[0]]) % (p[1] - p[0]) != 0
27             || a[p[0]] + (a[p[1]] - a[p[0]]) / (p[1] - p[0]) * 1LL * (j - p[0]) <= 0
28             || a[p[0]] + (a[p[1]] - a[p[0]]) / (p[1] - p[0]) * 1LL * (i - p[0]) <= 0)
29             {
30                 ans++;
31                 j = i;
32                 p.clear();
33                 if (a[i] != -1) {
34                     p.push_back(i);
35                 }
36             }
37         cout << ans << "\n";
38     }
39 }
40 }
```

### 732: Red-Blue Operations (Hard Version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The only difference between easy and hard versions is the maximum values of  $n$  and  $q$ .

You are given an array, consisting of  $n$  integers. Initially, all elements are red.

You can apply the following operation to the array multiple times. During the  $i$ -th operation, you select an element of the array; then:

if the element is red, it increases by  $i$  and becomes blue;

if the element is blue, it decreases by  $i$  and becomes red.

The operations are numbered from 1, i. e. during the first operation some element is changed by 1 and so on.

You are asked  $q$  queries of the following form:

given an integer  $k$ , what can the largest minimum in the array be if you apply exactly  $k$  operations to it?

Note that the operations don't affect the array between queries, all queries are asked on the initial array  $a$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 1E18;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, q;
9     cin >> n >> q;
10    vector<i64> a(n);
11    for (int i = 0; i < n; i++) {
12        cin >> a[i];
13    }
14    sort(a.begin(), a.end());
15    vector<i64> f(n + 1, inf);
16    for (int i = 0; i < n; i++) {
17        f[i + 1] = min(f[i] + 1, a[i] + 1);
18    }
19    i64 sum = accumulate(a.begin(), a.end(), 0LL);
20    while (q--) {
21        i64 k;
22        cin >> k;
23        if (k < n) {
24            cout << min(f[k], a[k]) << " ";
25        } else {
26            i64 ans;
27            i64 s = sum;
28            if ((k - n) % 2 == 0) {
29                ans = f[n] + k - n;
30                s += 1LL * n * (k + k - n + 1) / 2 - (k - n) / 2;
31            } else {
32                ans = min(a[n - 1], f[n - 1] + k - (n - 1));
33                s += 1LL * (n - 1) * (k + k - n + 2) / 2 - (k - n + 1) / 2;
34            }
35            ans = min(1LL * ans, s >= 0 ? s / n : (s - n + 1) / n);
36            cout << ans << " ";
37        }
38    }
39    cout << "\n";
40    return 0;
41 }

```

### 733: Trial for Chief

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Having unraveled the Berland Dictionary, the scientists managed to read the notes of the chroniclers of that time. For example, they learned how the chief of the ancient Berland tribe was chosen.

As soon as enough pretenders were picked, the following test took place among them: the chief of the tribe took a slab divided by horizontal and vertical stripes into identical squares (the slab consisted of  $N$  lines and  $M$  columns) and painted every square black or white. Then every pretender was given a

slab of the same size but painted entirely white. Within a day a pretender could paint any side-linked set of the squares of the slab some color. The set is called linked if for any two squares belonging to the set there is a path belonging the set on which any two neighboring squares share a side. The aim of each pretender is to paint his slab in the exactly the same way as the chiefs slab is painted. The one who paints a slab like that first becomes the new chief.

Scientists found the slab painted by the ancient Berland tribe chief. Help them to determine the minimal amount of days needed to find a new chief if he had to paint his slab in the given way.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int dx[] = {0, 0, -1, 1};
5 constexpr int dy[] = {-1, 1, 0, 0};
6 int main() {
7     ios::sync_with_stdio(false);
8     cin.tie(nullptr);
9     int N, M;
10    cin >> N >> M;
11    vector<string> s(N);
12    for (int i = 0; i < N; i++) {
13        cin >> s[i];
14    }
15    int ans = 1E9;
16    for (int x = 0; x < N; x++) {
17        for (int y = 0; y < M; y++) {
18            int res = 0;
19            deque<tuple<int, int, int>> q;
20            q.emplace_back(x, y, 0);
21            vector<vector<i64>> dis(N, vector(M, -1));
22            while (!q.empty()) {
23                auto [x, y, d] = q.front();
24                q.pop_front();
25                if (dis[x][y] != -1) {
26                    continue;
27                }
28                dis[x][y] = d;
29                res = max(res, d + (s[x][y] == 'B'));
30                for (int k = 0; k < 4; k++) {
31                    int nx = dx[k] + x;
32                    int ny = dy[k] + y;
33                    if (0 <= nx && nx < N && 0 <= ny && ny < M) {
34                        if (s[x][y] == s[nx][ny]) {
35                            q.emplace_front(nx, ny, d);
36                        } else {
37                            q.emplace_back(nx, ny, d + 1);
38                        }
39                    }
40                }
41            }
42            ans = min(ans, res);
43        }
44    }
45    cout << ans << "\n";

```

```

46     return 0;
47 }
```

### 734: King's Problem?

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Every true king during his life must conquer the world, hold the Codeforces world finals, win pink panda in the shooting gallery and travel all over his kingdom.

King Copa has already done the first three things. Now he just needs to travel all over the kingdom. The kingdom is an infinite plane with Cartesian coordinate system on it. Every city is a point on this plane. There are  $n$  cities in the kingdom at points with coordinates  $(x_1, 0), (x_2, 0), \dots, (x_n, 0)$ , and there is one city at point  $(x_{n+1}, y_{n+1})$ .

King starts his journey in the city number  $k$ . Your task is to find such route for the king, which visits all cities (in any order) and has minimum possible length. It is allowed to visit a city twice. The king can end his journey in any city. Between any pair of cities there is a direct road with length equal to the distance between the corresponding points. No two cities may be located at the same point.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, k;
8     cin >> n >> k;
9     k--;
10    vector<int> x(n);
11    for (int i = 0; i < n; i++) {
12        cin >> x[i];
13    }
14    int xn, yn;
15    cin >> xn >> yn;
16    int xk = x[k];
17    sort(x.begin(), x.end());
18    if (k < n) {
19        k = lower_bound(x.begin(), x.end(), xk) - x.begin();
20    }
21    double ans = 1E9;
22    if (k == 0 || k == n) {
```

```

23         ans = min(ans, x[n - 1] - x[0] + sqrt(1. * (x[n - 1] - xn) * (x[n - 1] - xn) + 1.
24             * yn * yn));
25     } else {
26         ans = min(ans, x[n - 1] - x[0] + sqrt(1. * (x[n - 1] - xn) * (x[n - 1] - xn) + 1.
27             * yn * yn)
28             + abs(xk - x[0]));
29     }
30     if (k == n - 1 || k == n) {
31         ans = min(ans, x[n - 1] - x[0] + sqrt(1. * (x[0] - xn) * (x[0] - xn) + 1. * yn *
32             yn));
33     } else {
34         ans = min(ans, x[n - 1] - x[0] + sqrt(1. * (x[0] - xn) * (x[0] - xn) + 1. * yn *
35             yn)
36             + abs(xk - x[n - 1]));
37     }
38     for (int i = 1; i < n; i++) {
39         for (auto a : {0, i - 1}) {
40             for (auto b : {i, n - 1}) {
41                 if (k == i - 1 - a || k == i + n - 1 - b) {
42                     ans = min(ans, x[i - 1] - x[0] + x[n - 1] - x[i]
43                         + sqrt(1. * (x[a] - xn) * (x[a] - xn) + 1. * yn * yn)
44                         + sqrt(1. * (x[b] - xn) * (x[b] - xn) + 1. * yn * yn));
45                 } else if (k != n) {
46                     ans = min(ans, x[i - 1] - x[0] + x[n - 1] - x[i]
47                         + sqrt(1. * (x[a] - xn) * (x[a] - xn) + 1. * yn * yn)
48                         + sqrt(1. * (x[b] - xn) * (x[b] - xn) + 1. * yn * yn)
49                         + min(abs(xk - x[i - 1 - a]), abs(xk - x[i + n - 1 - b])));
50                 }
51             }
52         }
53     }
54     cout << fixed << setprecision(10);
55     cout << ans << "\n";
56     return 0;
57 }
```

### 735: Traveling in Berland

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  cities in Berland, arranged in a circle and numbered from 1 to  $n$  in clockwise order.

You want to travel all over Berland, starting in some city, visiting all the other cities and returning to the starting city. Unfortunately, you can only drive along the Berland Ring Highway, which connects all  $n$  cities. The road was designed by a very titled and respectable minister, so it is one-directional - it can only be traversed clockwise, only from the city  $i$  to the city  $(i \bmod n) + 1$  (i.e. from 1 to 2, from 2 to 3, ..., from  $n$  to 1).

The fuel tank of your car holds up to  $k$  liters of fuel. To drive from the  $i$ -th city to the next one,  $a_i$  liters of fuel are needed (and are consumed in the process).

Every city has a fuel station; a liter of fuel in the  $i$ -th city costs  $b_i$  burles. Refueling between cities is not allowed; if fuel has run out between cities, then your journey is considered incomplete.

For each city, calculate the minimum cost of the journey if you start and finish it in that city.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<int> b(n);
12    for (int i = 0; i < n; i++) {
13        cin >> b[i];
14    }
15    i64 sum = accumulate(a.begin(), a.end(), 0LL);
16    vector<i64> f(2 * n + 1);
17    for (int i = 0; i < 2 * n; i++) {
18        f[i + 1] = f[i] + a[i % n];
19    }
20    int last = -1;
21    vector<int> pre(2 * n);
22    for (int i = 0; i < 2 * n; i++) {
23        if (b[i % n] == 1) {
24            last = i;
25        }
26        pre[i] = last;
27    }
28    vector<i64> ans(n);
29    ans[0] += sum;
30    for (int i = 0; i < 2 * n - 1; i++) {
31        if (b[i % n] == 2) {
32            if (pre[i] == -1) {
33                ans[max(0, i - n + 1)] += a[i % n];
34                if (i + 1 < n) {
35                    ans[i + 1] -= a[i % n];
36                }
37            } else {
38                int v = max(0LL, min(1LL * a[i % n], f[i + 1] - f[pre[i]] - k));
39                ans[max(0, i - n + 1)] += v;
40                if (pre[i] + 1 < n) {
41                    ans[pre[i] + 1] += a[i % n] - v;
42                }
43                if (i + 1 < n) {
44                    ans[i + 1] -= a[i % n];
45                }
46            }
47        }
48    }
49    for (int i = 1; i < n; i++) {
50        ans[i] += ans[i - 1];
51    }

```

```

52     for (int i = 0; i < n; i++) {
53         cout << ans[i] << " \n"[i == n - 1];
54     }
55 }
56 int main() {
57     ios::sync_with_stdio(false);
58     cin.tie(nullptr);
59     int t;
60     cin >> t;
61     while (t--) {
62         solve();
63     }
64     return 0;
65 }
```

### 736: Labeling the Tree with Distances

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an unweighted tree of  $n$  vertices numbered from 1 to  $n$  and a list of  $n - 1$  integers  $a_1, a_2, \dots, a_{n-1}$ . A tree is a connected undirected graph without cycles. You will use each element of the list to label one vertex. No vertex should be labeled twice. You can label the only remaining unlabeled vertex with any integer.

A vertex  $x$  is called good if it is possible to do this labeling so that for each vertex  $i$ , its label is the distance between  $x$  and  $i$ . The distance between two vertices  $s$  and  $t$  on a tree is the minimum number of edges on a path that starts at vertex  $s$  and ends at vertex  $t$ .

Find all good vertices.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 P = i64(1E18) + 9;
5 i64 mul(i64 a, i64 b) {
6     return __int128(a) * b % P;
7 }
8 int main() {
9     ios::sync_with_stdio(false);
10    cin.tie(nullptr);
11    int n;
12    cin >> n;
13    mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
14    const int X = rng() % 1000000000 + 123456;
15    vector<i64> p(n);
16    p[0] = 1;
```

```

17     for (int i = 1; i < n; i++) {
18         p[i] = mul(p[i - 1], X);
19     }
20     i64 sum = 0;
21     vector<int> a(n - 1);
22     for (int i = 0; i < n - 1; i++) {
23         cin >> a[i];
24         sum = (sum + p[a[i]]) % P;
25     }
26     vector<i64> cand;
27     for (int i = 0; i < n; i++) {
28         cand.push_back((sum + p[i]) % P);
29     }
30     sort(cand.begin(), cand.end());
31     vector<vector<int>> adj(n);
32     for (int i = 1; i < n; i++) {
33         int u, v;
34         cin >> u >> v;
35         u--, v--;
36         adj[u].push_back(v);
37         adj[v].push_back(u);
38     }
39     vector<i64> dp(n), up(n);
40     vector<int> parent(n, -1);
41     auto dfs1 = [&](auto self, int x) -> void {
42         dp[x] = 1;
43         for (auto y : adj[x]) {
44             if (y == parent[x]) {
45                 continue;
46             }
47             parent[y] = x;
48             self(self, y);
49             dp[x] = (dp[x] + mul(X, dp[y])) % P;
50         }
51     };
52     dfs1(dfs1, 0);
53     auto dfs2 = [&](auto self, int x) -> void {
54         i64 sum = 0;
55         for (auto y : adj[x]) {
56             if (y == parent[x]) {
57                 continue;
58             }
59             sum = (sum + dp[y]) % P;
60         }
61         for (auto y : adj[x]) {
62             if (y == parent[x]) {
63                 continue;
64             }
65             up[y] = (1 + mul((sum - dp[y] + P + up[x]) % P, X)) % P;
66             self(self, y);
67         }
68     };
69     dfs2(dfs2, 0);
70     vector<int> ans;
71     for (int x = 0; x < n; x++) {
72         i64 res = 1;
73         for (auto y : adj[x]) {
74             if (y == parent[x]) {
75                 res = (res + mul(up[x], X)) % P;
76             } else {
77                 res = (res + mul(dp[y], X)) % P;
78             }
79         }
80         if (binary_search(cand.begin(), cand.end(), res)) {

```

```

81         ans.push_back(x);
82     }
83 }
84 cout << ans.size() << "\n";
85 for (auto x : ans) {
86     cout << x + 1 << " \n"[x == ans.back()];
87 }
88 return 0;
89 }
```

### 737: Colored Subgraphs

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp has a tree, consisting of  $n$  vertices.

He is going to select some vertex  $r$  and perform the following operations on each vertex  $v$  from 1 to  $n$ :

set  $d_v$  equal to the distance from  $v$  to  $r$  (the number of edges on the shortest path);

color  $v$  some color.

A nice coloring satisfies two conditions:

for each pair of vertices of the same color  $(v, u)$ , there exists a path from  $v$  to  $u$  that only visits vertices of the same color;

for each pair of vertices of the same color  $(v, u)$ ,  $d_v \neq d_u$ .

Note that Monocarp can choose any amount of different colors he wants to use.

For each used color, he then counts the number of vertices of this color. The cost of the tree is the minimum of these numbers.

What can be the maximum cost of the tree?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int inf = 1E9;
5 # struct Info;
6 Info operator+(Info a, Info b) {
```

```

7     Info c;
8     c.min0 = min(a.min0, b.min0);
9     if (a.min1 < b.min1) {
10         c.min1 = a.min1;
11         c.min0 = min(c.min0, b.min1);
12     } else {
13         c.min1 = b.min1;
14         c.min0 = min(c.min0, a.min1);
15     }
16     return c;
17 }
18 Info toInfo(array<int, 2> a) {
19     return {a[0], a[1]};
20 }
21 array<int, 2> fromSub(Info a) {
22     return {a.min0, a.min1 == inf ? 1 : a.min1 + 1};
23 }
24 void solve() {
25     int n;
26     cin >> n;
27     vector<vector<int>> adj(n);
28     for (int i = 1; i < n; i++) {
29         int u, v;
30         cin >> u >> v;
31         u--, v--;
32         adj[u].push_back(v);
33         adj[v].push_back(u);
34     }
35     vector<array<int, 2>> dp(n), up(n, {inf, inf});
36     vector<int> parent(n, -1);
37     auto dfs1 = [&](auto self, int x) -> void {
38         Info info;
39         for (auto y : adj[x]) {
40             if (y == parent[x]) {
41                 continue;
42             }
43             parent[y] = x;
44             self(self, y);
45             info = info + toInfo(dp[y]);
46         }
47         dp[x] = fromSub(info);
48     };
49     dfs1(dfs1, 0);
50     auto dfs2 = [&](auto self, int x) -> void {
51         vector<Info> pre;
52         for (auto y : adj[x]) {
53             if (y == parent[x]) {
54                 pre.push_back(toInfo(up[x]));
55             } else {
56                 pre.push_back(toInfo(dp[y]));
57             }
58         }
59         auto suf = pre;
60         for (int i = 1; i < pre.size(); i++) {
61             pre[i] = pre[i - 1] + pre[i];
62         }
63         for (int i = suf.size() - 1; i > 0; i--) {
64             suf[i - 1] = suf[i] + suf[i - 1];
65         }
66         for (int i = 0; i < adj[x].size(); i++) {
67             int y = adj[x][i];
68             if (y == parent[x]) {
69                 continue;
70             }
71         }
72     };
73     dfs2(dfs2, 0);
74 }

```

```

71         Info info;
72         if (i > 0) {
73             info = info + pre[i - 1];
74         }
75         if (i + 1 < adj[x].size()) {
76             info = info + suf[i + 1];
77         }
78         up[y] = fromSub(info);
79         self(self, y);
80     }
81 };
82 dfs2(dfs2, 0);
83 int ans = 0;
84 for (int x = 0; x < n; x++) {
85     Info info;
86     for (auto y : adj[x]) {
87         if (y == parent[x]) {
88             info = info + toInfo(up[x]);
89         } else {
90             info = info + toInfo(dp[y]);
91         }
92     }
93     auto res = fromSub(info);
94     ans = max(ans, min(res[0], res[1]));
95 }
96 cout << ans << "\n";
97 }
98 int main() {
99     ios::sync_with_stdio(false);
100    cin.tie(nullptr);
101    int t;
102    cin >> t;
103    while (t--) {
104        solve();
105    }
106    return 0;
107 }
```

### 738: Spinach Pizza

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The two siblings Alberto and Beatrice have to eat a spinach pizza together. However, none of them likes spinach, so they both want to eat as little as possible.

The pizza has the shape of a strictly convex polygon with  $n$  vertices located at integer coordinates  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$  of the plane.

The siblings have decided to eat the pizza in the following way: taking turns, starting with Alberto, each sibling chooses a vertex of the remaining part of the pizza and eats out the triangle determined by its two neighboring edges. In this way, after each of the first  $n - 3$  turns the pizza will have one less vertex.

The game ends after the  $(n - 2)$ -th turn, when all the pizza has been eaten.

Assuming that Alberto and Beatrice choose the slices to eat optimally, which of the siblings manages to eat at most half of the pizza? You should identify a sibling that has a strategy to do so and help them choose the slices appropriately. Note that it is possible that both Alberto and Beatrice end up eating exactly half of the area if they choose their slices optimally.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 using T = i64;
5 # struct Point;
6 T dot(const Point &a, const Point &b) {
7     return a.x * b.x + a.y * b.y;
8 }
9 T cross(const Point &a, const Point &b) {
10    return a.x * b.y - a.y * b.x;
11 }
12 int main() {
13     ios::sync_with_stdio(false);
14     cin.tie(nullptr);
15     int n;
16     cin >> n;
17     vector<Point> p(n);
18     for (int i = 0; i < n; i++) {
19         cin >> p[i].x >> p[i].y;
20     }
21     vector<bool> ok(n, true);
22     if (n % 2 == 0) {
23         cout << "Alberto" << endl;
24     } else {
25         cout << "Beatrice" << endl;
26         int x;
27         cin >> x;
28         x--;
29         ok[x] = false;
30     }
31     while (true) {
32         vector<int> a;
33         for (int i = 0; i < n; i++) {
34             if (ok[i]) {
35                 a.push_back(i);
36             }
37         }
38         if (a.size() == 2) {
39             break;
40         }
41         int t = -1;
42         i64 ans = 1E18;
43         for (int i = 0; i < a.size(); i++) {
44             int q = a[i];
45             int l = a[(i - 1 + a.size()) % a.size()];
46             int r = a[(i + 1 + a.size()) % a.size()];
47             i64 area = cross(p[r] - p[q], p[l] - p[q]);
48             if (area < ans) {
49                 ans = area;
50                 t = a[i];
51             }
52         }
53     }
54 }
```

```
51         }
52     }
53     cout << t + 1 << endl;
54     ok[t] = false;
55     int x;
56     cin >> x;
57     x--;
58     ok[x] = false;
59 }
60
61 }
```

### 739: Library game

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Alessia and Bernardo are discovering the world of competitive programming through the books of their university library.

The library consists of  $m$  sections numbered from 1 to  $m$ . Each section contains only books dedicated to a particular subject and different sections correspond to different subjects. In order to prevent the students from wandering in the library, the university has established a system of passes. Each pass has a length  $y$  associated to it and allows access to an interval of  $y$  consecutive sections in the library. During a visit, the student must choose exactly one book from one of these sections and leave the library. Each pass can be used only once.

At the moment Alessia and Bernardo have  $n$  passes of lengths  $x_1, x_2, \dots, x_n$ . They have different opinions on the best way to improve: Alessia thinks that it is important to study many different topics, while Bernardo believes that it is important to study deeply at least one topic. So, Alessia wants to use the  $n$  passes to get  $n$  books on distinct topics, while Bernardo would like to get at least two books on the same topic.

They have reached the following agreement: for each of the following  $n$  days, Alessia will choose a pass of length  $y$  among those which are still available and an interval of  $y$  sections in the library, and Bernardo will go into the library and will take exactly one book from one of those sections.

Can Bernardo manage to get at least two books on the same subject, or will Alessia be able to avoid it?

You should decide whether you want to be Alessia or Bernardo, and you have to fulfill the goal of your chosen character. The judge will impersonate the other character. Note that, even if at some moment Bernardo has already taken two books on the same subject, the interaction should go on until the end of the  $n$  days.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<int> y(n);
10    for (int i = 0; i < n; i++) {
11        cin >> y[i];
12    }
13    sort(y.begin(), y.end(), greater());
14    int t = -1;
15    for (int i = 0; i < n; i++) {
16        if (y[i] * (i + 1) > m) {
17            t = i;
18            break;
19        }
20    }
21    vector<int> a{0, m + 1};
22    if (t == -1) {
23        cout << "Alessia" << endl;
24        for (int i = 0; i < n; i++) {
25            int len = y[i];
26            for (int j = 0; j < a.size() - 1; j++) {
27                if (a[j + 1] - a[j] > len) {
28                    cout << len << " " << a[j] + 1 << endl;
29                    break;
30                }
31            }
32            int b;
33            cin >> b;
34            a.push_back(b);
35            sort(a.begin(), a.end());
36        }
37    } else {
38        cout << "Bernardo" << endl;
39        for (int i = 0; i < n; i++) {
40            int len, l;
41            cin >> len >> l;
42            int r = l + len - 1;
43            bool ok = false;
44            for (auto x : a) {
45                if (l <= x && x <= r) {
46                    cout << x << endl;
47                    ok = true;
48                    break;
49                }
50            }
51            if (!ok) {
52                for (int i = y[t]; i <= m; i += y[t]) {
53                    if (l <= i && i <= r) {
54                        cout << i << endl;
55                        ok = true;
56                        a.push_back(i);
57                        break;
58                    }
59                }
59            if (!ok) {
60

```

```

61             cout << l << endl;
62         }
63     }
64 }
65 return 0;
66 }
67 }
```

## 740: Blocking Chips

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a tree, consisting of  $n$  vertices. There are  $k$  chips, placed in vertices  $a_1, a_2, \dots, a_k$ . All  $a_i$  are distinct. Vertices  $a_1, a_2, \dots, a_k$  are colored black initially. The remaining vertices are white.

You are going to play a game where you perform some moves (possibly, zero). On the  $i$ -th move (1-indexed) you are going to move the  $((i - 1) \bmod k + 1)$ -st chip from its current vertex to an adjacent white vertex and color that vertex black. So, if  $k = 3$ , you move chip 1 on move 1, chip 2 on move 2, chip 3 on move 3, chip 1 on move 4, chip 2 on move 5 and so on. If there is no adjacent white vertex, then the game ends.

What's the maximum number of moves you can perform?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<vector<int>> adj(n);
8     for (int i = 1; i < n; i++) {
9         int u, v;
10        cin >> u >> v;
11        u--, v--;
12        adj[u].push_back(v);
13        adj[v].push_back(u);
14    }
15    int k;
16    cin >> k;
17    vector<int> a(k);
18    for (int i = 0; i < k; i++) {
19        cin >> a[i];
20        a[i]--;
21    }
```

```

22     int lo = 0, hi = n;
23     while (lo < hi) {
24         int x = (lo + hi + 1) / 2;
25         vector<int> f(n, -1);
26         for (int i = 0; i < k; i++) {
27             f[a[i]] = x / k + (i < x % k);
28         }
29         vector<int> dp(n);
30         bool ok = true;
31         auto dfs = [&](auto self, int x, int p) -> void {
32             int neg = -f[x], max = 0;
33             for (auto y : adj[x]) {
34                 if (y == p) {
35                     continue;
36                 }
37                 self(self, y, x);
38                 if (dp[y] < 0) {
39                     if (neg <= 0) {
40                         ok = false;
41                     }
42                     neg = dp[y] + 1;
43                 } else {
44                     max = max(max, dp[y]);
45                 }
46             }
47             if (neg <= 0) {
48                 if (neg + max >= 0) {
49                     dp[x] = 0;
50                 } else {
51                     dp[x] = neg;
52                 }
53             } else {
54                 dp[x] = max + 1;
55             }
56             // cerr << "dp[" << x << "] = " << dp[x] << "\n";
57         };
58         dfs(dfs, 0, -1);
59         // cerr << "x = " << x << "\n";
60         if (ok && dp[0] >= 0) {
61             lo = x;
62         } else {
63             hi = x - 1;
64         }
65     }
66     cout << lo << "\n";
67 }
68 int main() {
69     ios::sync_with_stdio(false);
70     cin.tie(nullptr);
71     int t;
72     cin >> t;
73     while (t--) {
74         solve();
75     }
76     return 0;
77 }
```

**741: Velepin and Marketing**

- Time limit: 2 seconds

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The famous writer Velepin is very productive. Recently, he signed a contract with a well-known publication and now he needs to write  $k_i$  books for  $i$ -th year. This is not a problem for him at all, he can write as much as he wants about samurai, space, emptiness, insects and werewolves.

He has  $n$  regular readers, each of whom in the  $i$ -th year will read one of the  $k_i$  books published by Velepin. Readers are very fond of discussing books, so the  $j$ -th of them will be satisfied within a year if at least  $a_j$  persons read the same book as him (including himself).

Velepin has obvious problems with marketing, so he turned to you! A well-known book reading service can control what each of Velepin's regular readers will read, but he does not want books to be wasted, so someone should read each book. And so they turned to you with a request to tell you what the maximum number of regular readers can be made satisfied during each of the years, if you can choose each person the book he will read.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    sort(a.begin(), a.end());
14    vector<int> dp(n + 1), pre(n + 1);
15    for (int i = 1; i <= n; i++) {
16        if (a[i - 1] <= i) {
17            dp[i] = 1 + pre[i - a[i - 1]];
18        } else {
19            dp[i] = 1 - a[i - 1] + i;
20        }
21        pre[i] = max(pre[i - 1], dp[i]);
22    }
23    vector<int> ans(n + 1);
24    for (int i = 1; i <= n; i++) {
25        if (dp[i] + n - i > 0) {
26            ans[dp[i] + n - i] = i;
27        }
28    }
29    for (int i = n - 1; i >= 0; i--) {
30        ans[i] = max(ans[i], ans[i + 1]);
31    }

```

```

32     int q;
33     cin >> q;
34     for (int i = 0; i < q; i++) {
35         int k;
36         cin >> k;
37         cout << ans[k] << "\n";
38     }
39     return 0;
40 }
```

## 742: XOR, Tree, and Queries

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a tree of  $n$  vertices. The vertices are numbered from 1 to  $n$ .

You will need to assign a weight to each edge. Let the weight of the  $i$ -th edge be  $a_i$  ( $1 \leq i \leq n - 1$ ). The weight of each edge should be an integer between 0 and  $2^{30} - 1$ , inclusive.

You are given  $q$  conditions. Each condition consists of three integers  $u, v$ , and  $x$ . This means that the bitwise XOR of all edges on the shortest path from  $u$  to  $v$  should be  $x$ .

Find out if there exist  $a_1, a_2, \dots, a_{n-1}$  that satisfy the given conditions. If yes, print a solution such that  $a_1 \oplus a_2 \oplus \dots \oplus a_{n-1}$  is the smallest. Here,  $\oplus$  denotes the bitwise XOR operation.

If there are multiple solutions such that  $a_1 \oplus a_2 \oplus \dots \oplus a_{n-1}$  is the smallest, print any.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, q;
8     cin >> n >> q;
9     vector<int> need(n);
10    vector<pair<int, int>> edges(n - 1);
11    for (int i = 0; i < n - 1; i++) {
12        int u, v;
13        cin >> u >> v;
14        u--, v--;
15        edges[i] = {u, v};
16        need[u] ^= 1;
17        need[v] ^= 1;
18    }
19    vector<vector<pair<int, int>>> adj(n);
20    for (int i = 0; i < q; i++) {
```

```

21     int a, b, c;
22     cin >> a >> b >> c;
23     a--, b--;
24     adj[a].emplace_back(b, c);
25     adj[b].emplace_back(a, c);
26 }
27 vector<int> val(n, -1), belong(n), siz(n);
28 int ans = 0;
29 for (int i = 0; i < n; i++) {
30     if (val[i] != -1) {
31         continue;
32     }
33     queue<int> q;
34     q.push(i);
35     val[i] = 0;
36     while (!q.empty()) {
37         int x = q.front();
38         q.pop();
39         if (need[x] == 1) {
40             ans ^= val[x];
41             siz[i]++;
42         }
43         belong[x] = i;
44         for (auto [y, w] : adj[x]) {
45             if (val[y] == -1) {
46                 val[y] = val[x] ^ w;
47                 q.push(y);
48             }
49             if (val[y] != (val[x] ^ w)) {
50                 cout << "No\n";
51                 return 0;
52             }
53         }
54     }
55 }
56 for (int i = 0; i < n; i++) {
57     if (siz[i] % 2 == 1) {
58         for (int j = 0; j < n; j++) {
59             if (belong[j] == i) {
60                 val[j] ^= ans;
61             }
62         }
63         break;
64     }
65 }
66 cout << "Yes\n";
67 for (int i = 0; i < n - 1; i++) {
68     auto [u, v] = edges[i];
69     cout << (val[u] ^ val[v]) << " \n"[i == n - 2];
70 }
71 return 0;
72 }

```

**743: Laboratory on Pluto**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

As you know, Martian scientists are actively engaged in space research. One of the highest priorities is Pluto. In order to study this planet in more detail, it was decided to build a laboratory on Pluto.

It is known that the lab will be built of  $n$  square blocks of equal size. For convenience, we will assume that Pluto's surface is a plane divided by vertical and horizontal lines into unit squares. Each square is either occupied by a lab block or not, and only  $n$  squares are occupied.

Since each block is square, it has four walls. If a wall is adjacent to another block, it is considered inside, otherwise - outside.

Pluto is famous for its extremely cold temperatures, so the outside walls of the lab must be insulated. One unit of insulation per exterior wall would be required. Thus, the greater the total length of the outside walls of the lab (i. e., its perimeter), the more insulation will be needed.

Consider the lab layout in the figure below. It shows that the lab consists of  $n = 33$  blocks, and all the blocks have a total of 24 outside walls, i. e. 24 units of insulation will be needed.

You should build the lab optimally, i. e., minimize the amount of insulation. On the other hand, there may be many optimal options, so scientists may be interested in the number of ways to build the lab using the minimum amount of insulation, modulo a prime number  $m$ .

Two ways are considered the same if they are the same when overlapping without turning. Thus, if a lab plan is rotated by  $90^\circ$ , such a new plan can be considered a separate way.

To help scientists explore Pluto, you need to write a program that solves these difficult problems.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int P;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
8     if (x < 0) {
9         x += P;
10    }
11    if (x >= P) {
12        x -= P;
13    }
14    return x;
15 }
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24 }
```

```
23     }
24     return res;
25 }
26 # struct Z;
27 void solve1() {
28     int n;
29     cin >> n;
30     int a = 1, b = 1;
31     while (1) {
32         if (a * b >= n) break;
33         a++;
34         if (a * b >= n) break;
35         b++;
36     }
37     cout << a << " " << b << "\n";
38     for (int i = 0; i < a; i++) {
39         for (int j = 0; j < b; j++) {
40             cout << ".#" [i * b + j < n];
41         }
42         cout << "\n";
43     }
44 }
45 constexpr int N = 1000;
46 Z f[N + 1];
47 void solve2() {
48     int n;
49     cin >> n;
50     int a = sqrt(n), b = a;
51     while (1) {
52         if (a * b >= n) break;
53         a++;
54         if (a * b >= n) break;
55         b++;
56     }
57     Z ans = 0;
58     while (a * b >= n) {
59         ans += f[a * b - n] * (1 + (a != b));
60         a++, b--;
61     }
62     cout << 2 * (a + b) << " " << ans << "\n";
63 }
64 int main() {
65     ios::sync_with_stdio(false);
66     cin.tie(nullptr);
67     int t, u;
68     cin >> t >> u;
69     if (u == 2) {
70         cin >> P;
71         f[0] = 1;
72         for (int i = 1; i <= N; i++) {
73             for (int t = 0; t < 4; t++) {
74                 for (int j = i; j <= N; j++) {
75                     f[j] += f[j - i];
76                 }
77             }
78         }
79     }
80     while (t--) {
81         if (u == 1) {
82             solve1();
83         } else {
84             solve2();
85         }
86     }
}
```

```
87     return 0;
88 }
```

## 744: Anya's Simultaneous Exhibition

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an interactive problem.

Anya has gathered  $n$  chess experts numbered from 1 to  $n$  for which the following properties hold:

For any pair of players one of the players wins every game against the other (and no draws ever occur);

Transitivity does not necessarily hold - it might happen that  $A$  always beats  $B$ ,  $B$  always beats  $C$  and  $C$  always beats  $A$ .

To organize a tournament, Anya hosts  $n - 1$  games. In each game, she chooses two players. One of them wins and stays, while the other one is disqualified. After all the games are hosted only one player will remain. A player is said to be a candidate master if they can win a tournament (notice that the winner of a tournament may depend on the players selected by Anya in the  $n - 1$  games).

Since Anya is a curious girl, she is interested in finding the candidate masters. Unfortunately, she does not have much time. To speed up the process, she will organize up to  $2n$  simul (short for “simultaneous exhibition”, in which one player plays against many).

In one simul, Anya chooses exactly one player who will play against some (at least one) of the other players. The chosen player wins all games they would win in a regular game, and the same holds for losses. After the simul finishes, Anya is only told the total number of games won by the chosen player (but not which ones). Nobody is disqualified during a simul.

Can you help Anya host simuls and determine the candidate masters?

The winning players in each pair could be changed between the simuls, but only in a way that preserves the results of all previous simuls. These changes may depend on your queries.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
```

```

5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> deg(n);
10    for (int i = 0; i < n; i++) {
11        string s(n, '1');
12        s[i] = '0';
13        cout << "? " << i + 1 << " " << s << endl;
14        cin >> deg[i];
15    }
16    vector<int> p(n);
17    iota(p.begin(), p.end(), 0);
18    sort(p.begin(), p.end(), [&](int i, int j) {
19        return deg[i] < deg[j];
20    });
21    int sum = 0;
22    int beg = 0;
23    for (int i = 0; i < n - 1; i++) {
24        sum += deg[p[i]];
25        if (sum == i * (i + 1) / 2) beg = i + 1;
26    }
27    string s(n, '0');
28    for (int i = beg; i < n; i++) {
29        s[p[i]] = '1';
30    }
31    cout << "! " << s << endl;
32    return 0;
33 }

```

**math****745: Construct Matrix**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an even integer  $n$  and an integer  $k$ . Your task is to construct a matrix of size  $n \times n$  consisting of numbers 0 and 1 in such a way that the following conditions are true, or report that it is impossible:

the sum of all the numbers in the matrix is exactly  $k$ ;

the bitwise XOR of all the numbers in the row  $i$  is the same for each  $i$ ;

the bitwise XOR of all the numbers in the column  $j$  is the same for each  $j$ .

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     if (k % 2 == 1) {
8         cout << "No\n";
9         return;
10    }
11    vector a(n, vector<int>(n));
12    if (k % 4 == 0) {
13        for (int i = 0; i < n; i += 2) {
14            for (int j = 0; j < n; j += 2) {
15                if (k > 0) {
16                    a[i][j] = a[i][j + 1] = a[i + 1][j] = a[i + 1][j + 1] = 1;
17                    k -= 4;
18                }
19            }
20        }
21    } else if (n == 2) {
22        a[0][0] = a[1][1] = 1;
23    } else if (k == 2 || k == n * n - 2) {
24        cout << "No\n";
25        return;
26    } else {
27        if (k > n * n / 2) {
28            for (int i = 0; i < n; i++) {
29                for (int j = 0; j < n; j++) {
30                    a[i][j] = 1;
31                }
32            }
33            k = n * n - k;
34        }
35        a[0][0] ^= 1;
36        a[0][1] ^= 1;
37        a[1][0] ^= 1;
38        a[1][2] ^= 1;
39        a[2][1] ^= 1;
40        a[2][2] ^= 1;
41        k -= 6;
42        for (int i = 0; i < n; i += 2) {
43            for (int j = 0; j < n; j += 2) {
44                if (i < 4 && j < 4) {
45                    continue;
46                }
47                if (k > 0) {
48                    a[i][j] ^= 1;
49                    a[i][j + 1] ^= 1;
50                    a[i + 1][j] ^= 1;
51                    a[i + 1][j + 1] ^= 1;
52                    k -= 4;
53                }
54            }
55        }
56    }
57    cout << "Yes\n";
58    for (int i = 0; i < n; i++) {
59        for (int j = 0; j < n; j++) {
60            cout << a[i][j] << " \n"[j == n - 1];
61        }
62    }
63 }
64 int main() {
```

```

65     ios::sync_with_stdio(false);
66     cin.tie(nullptr);
67     int t;
68     cin >> t;
69     while (t--) {
70         solve();
71     }
72     return 0;
73 }
```

## 746: Multiple Lamps

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have  $n$  lamps, numbered from 1 to  $n$ . Initially, all the lamps are turned off.

You also have  $n$  buttons. The  $i$ -th button toggles all the lamps whose index is a multiple of  $i$ . When a lamp is toggled, if it was off it turns on, and if it was on it turns off.

You have to press some buttons according to the following rules.

You have to press at least one button.

You cannot press the same button multiple times.

You are given  $m$  pairs  $(u_i, v_i)$ . If you press the button  $u_i$ , you also have to press the button  $v_i$  (at any moment, not necessarily after pressing the button  $u_i$ ). Note that, if you press the button  $v_i$ , you don't need to press the button  $u_i$ .

You don't want to waste too much electricity. Find a way to press buttons such that at the end at most  $\lfloor n/5 \rfloor$  lamps are on, or print  $-1$  if it is impossible.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<int> u(m), v(m);
8     for (int i = 0; i < m; i++) {
9         cin >> u[i] >> v[i];
10    }
11    if (n >= 20) {
12        cout << n << "\n";
13        for (int i = 1; i <= n; i++) {
```

```

14         cout << i << " \n"[i == n];
15     }
16     return;
17 }
18 int ans = -1;
19 auto check = [&](int s) {
20     if (__builtin_popcount(s) > n / 5) {
21         return;
22     }
23     for (int i = 1; i <= n; i++) {
24         if (s >> i & 1) {
25             for (int j = 2 * i; j <= n; j += i) {
26                 s ^= 1 << j;
27             }
28         }
29     }
30     for (int i = 0; i < m; i++) {
31         if (s >> u[i] & ~s >> v[i] & 1) {
32             return;
33         }
34     }
35     ans = s;
36 };
37 for (int x = 1; x <= n; x++) {
38     check(1 << x);
39     for (int y = 1; y < x; y++) {
40         check(1 << x | 1 << y);
41         for (int z = 1; z < y; z++) {
42             check(1 << x | 1 << y | 1 << z);
43         }
44     }
45 }
46 if (ans == -1) {
47     cout << -1 << "\n";
48     return;
49 }
50 cout << __builtin_popcount(ans) << "\n";
51 for (int i = 1; i <= n; i++) {
52     if (ans >> i & 1) {
53         cout << i << " ";
54     }
55 }
56 cout << "\n";
57 }
58 int main() {
59     ios::sync_with_stdio(false);
60     cin.tie(nullptr);
61     int t;
62     cin >> t;
63     while (t--) {
64         solve();
65     }
66     return 0;
67 }

```

## 747: Trees and XOR Queries Again

- Time limit: 6.5 seconds
- Memory limit: 512 megabytes
- Input file: standard input

- Output file: standard output

You are given a tree consisting of  $n$  vertices. There is an integer written on each vertex; the  $i$ -th vertex has integer  $a_i$  written on it.

You have to process  $q$  queries. The  $i$ -th query consists of three integers  $x_i, y_i$  and  $k_i$ . For this query, you have to answer if it is possible to choose a set of vertices  $v_1, v_2, \dots, v_m$  (possibly empty) such that:

every vertex  $v_j$  is on the simple path between  $x_i$  and  $y_i$  (endpoints can be used as well);

$a_{v_1} \oplus a_{v_2} \oplus \dots \oplus a_{v_m} = k_i$ , where  $\oplus$  denotes the bitwise XOR operator.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct HLD;
5 # struct Basis;
6 int main() {
7     ios::sync_with_stdio(false);
8     cin.tie(nullptr);
9     int n;
10    cin >> n;
11    vector<int> a(n);
12    for (int i = 0; i < n; i++) {
13        cin >> a[i];
14    }
15    HLD t(n);
16    for (int i = 1; i < n; i++) {
17        int u, v;
18        cin >> u >> v;
19        u--, v--;
20        t.addEdge(u, v);
21    }
22    t.work();
23    vector<Basis> basis(n);
24    auto dfs = [&](auto self, int x) -> void {
25        basis[x].add(a[x], t.dep[x]);
26        for (auto y : t.adj[x]) {
27            basis[y] = basis[x];
28            self(self, y);
29        }
30    };
31    dfs(dfs, 0);
32    int q;
33    cin >> q;
34    while (q--) {
35        int x, y, k;
36        cin >> x >> y >> k;
37        x--, y--;
38        int l = t.lca(x, y);
39        auto bas = basis[x];
40        for (int i = 0; i < 20; i++) {
41            if (basis[y].t[i] >= t.dep[l]) {
42                bas.add(basis[y].a[i]);

```

```

43         }
44     }
45     cout << (bas.query(k, t.dep[l]) ? "YES" : "NO") << "\n";
46 }
47 return 0;
48 }
```

### 748: Brukhovich and Exams

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The boy Smilo is learning algorithms with a teacher named Brukhovich.

Over the course of the year, Brukhovich will administer  $n$  exams. For each exam, its difficulty  $a_i$  is known, which is a non-negative integer.

Smilo doesn't like when the greatest common divisor of the difficulties of two consecutive exams is equal to 1. Therefore, he considers the sadness of the academic year to be the number of such pairs of exams. More formally, the sadness is the number of indices  $i$  ( $1 \leq i \leq n - 1$ ) such that  $\gcd(a_i, a_{i+1}) = 1$ , where  $\gcd(x, y)$  is the greatest common divisor of integers  $x$  and  $y$ .

Brukhovich wants to minimize the sadness of the year of Smilo. To do this, he can set the difficulty of any exam to 0. However, Brukhovich doesn't want to make his students' lives too easy. Therefore, he will perform this action no more than  $k$  times.

Help Smilo determine the minimum sadness that Brukhovich can achieve if he performs no more than  $k$  operations.

As a reminder, the greatest common divisor (GCD) of two non-negative integers  $x$  and  $y$  is the maximum integer that is a divisor of both  $x$  and  $y$  and is denoted as  $\gcd(x, y)$ . In particular,  $\gcd(x, 0) = \gcd(0, x) = x$  for any non-negative integer  $x$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
```

```

11     int ans = 0;
12     for (int i = 1; i < n; i++) {
13         ans += gcd(a[i - 1], a[i]) == 1;
14     }
15     vector<int> b;
16     for (int l = 0, r = 0; l < n; l = r + 1) {
17         r = l;
18         if (a[l] == 1) {
19             continue;
20         }
21         while (r + 1 < n && a[r + 1] != 1 && gcd(a[r], a[r + 1]) == 1) {
22             r += 1;
23         }
24         int t = (r - l) / 2;
25         for (int i = 0; i < t; i++) {
26             b.push_back(1);
27         }
28     }
29     for (int l = 0, r = 0; l < n; l = r + 1) {
30         r = l;
31         if (a[l] != 1) {
32             continue;
33         }
34         while (r + 1 < n && a[r + 1] == 1) {
35             r += 1;
36         }
37         if (l != 0 && r != n - 1) {
38             b.push_back(r - l + 1);
39         }
40     }
41     if (a == vector(n, 1)) {
42         ans = n - k;
43     } else {
44         ans -= k;
45         sort(b.begin(), b.end());
46         for (auto x : b) {
47             if (k >= x) {
48                 k -= x;
49                 ans -= 1;
50             }
51         }
52         ans = max(0, ans);
53     }
54     cout << ans << "\n";
55 }
56 int main() {
57     ios::sync_with_stdio(false);
58     cin.tie(nullptr);
59     int t;
60     cin >> t;
61     while (t--) {
62         solve();
63     }
64     return 0;
65 }
```

**749: Bored Bakry**

- Time limit: 4 seconds
- Memory limit: 256 megabytes

- Input file: standard input
- Output file: standard output

Bakry got bored of solving problems related to xor, so he asked you to solve this problem for him.

You are given an array  $a$  of  $n$  integers  $[a_1, a_2, \dots, a_n]$ .

Let's call a subarray  $a_l, a_{l+1}, a_{l+2}, \dots, a_r$  good if  $a_l \& a_{l+1} \& a_{l+2} \dots \& a_r > a_l \oplus a_{l+1} \oplus a_{l+2} \dots \oplus a_r$ , where  $\oplus$  denotes the bitwise XOR operation and  $\&$  denotes the bitwise AND operation.

Find the length of the longest good subarray of  $a$ , or determine that no such subarray exists.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    int ans = 0;
14    for (int t = 0; t < 20; t++) {
15        vector<int> s(n + 1);
16        for (int i = 0; i < n; i++) {
17            s[i + 1] = s[i] ^ a[i];
18        }
19        vector<int> r(n + 1);
20        r[n] = n;
21        for (int i = n - 1; i >= 0; i--) {
22            r[i] = a[i] & 1 ? r[i + 1] : i;
23        }
24        vector<int> lst(1 << (20 - t), -1);
25        vector<int> f(n + 1);
26        for (int i = n; i >= 0; i--) {
27            f[i] = i;
28            if (lst[s[i]] != -1) {
29                int j = lst[s[i]];
30                if (r[i] >= j) {
31                    f[i] = f[j];
32                }
33            }
34            ans = max(ans, f[i] - i);
35            lst[s[i]] = i;
36        }
37        for (auto &x : a) {
38            x /= 2;
39        }
40    }
41    cout << ans << "\n";
42    return 0;
43 }
```

## 750: Ideal Farm

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Theofanis decided to visit his uncle's farm. There are  $s$  animals and  $n$  animal pens on the farm. For utility purpose, animal pens are constructed in one row.

Uncle told Theofanis that a farm is lucky if you can distribute all animals in all pens in such a way that there are no empty pens and there is at least one continuous segment of pens that has exactly  $k$  animals in total.

Moreover, a farm is ideal if it's lucky for any distribution without empty pens.

Neither Theofanis nor his uncle knows if their farm is ideal or not. Can you help them to figure it out?

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     i64 s, n, k;
6     cin >> s >> n >> k;
7     if (s == k) {
8         cout << "YES\n";
9         return;
10    }
11    i64 ans = 0;
12    s += 1;
13    ans += (s / k + 2) / 2 * (s % k);
14    ans += (s / k + 1) / 2 * (k - s % k);
15    if (n + 1 <= ans) {
16        cout << "NO\n";
17    } else {
18        cout << "YES\n";
19    }
20 }
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     int t;
25     cin >> t;
26     while (t--) {
27         solve();
28     }
29     return 0;
30 }
```

## 751: Cutting Rectangle

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A rectangle with sides  $A$  and  $B$  is cut into rectangles with cuts parallel to its sides. For example, if  $p$  horizontal and  $q$  vertical cuts were made,  $(p + 1) \cdot (q + 1)$  rectangles were left after the cutting. After the cutting, rectangles were of  $n$  different types. Two rectangles are different if at least one side of one rectangle isn't equal to the corresponding side of the other. Note that the rectangle can't be rotated, this means that rectangles  $a \times b$  and  $b \times a$  are considered different if  $a \neq b$ .

For each type of rectangles, lengths of the sides of rectangles are given along with the amount of the rectangles of this type that were left after cutting the initial rectangle.

Calculate the amount of pairs  $(A; B)$  such as the given rectangles could be created by cutting the rectangle with sides of lengths  $A$  and  $B$ . Note that pairs  $(A; B)$  and  $(B; A)$  are considered different when  $A \neq B$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 using i128 = __int128;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n;
9     cin >> n;
10    map<i64, i64> cntw, cnth;
11    map<pair<i64, i64>, i64> cnt;
12    vector<i64> w(n), h(n), c(n);
13    i64 tot = 0;
14    for (int i = 0; i < n; i++) {
15        cin >> w[i] >> h[i] >> c[i];
16        cntw[w[i]] += c[i];
17        cnth[h[i]] += c[i];
18        cnt[{w[i], h[i]}] += c[i];
19        tot += c[i];
20    }
21    i64 gw = 0, gh = 0;
22    for (auto [w, c] : cntw) {
23        gw = gcd(gw, c);
24    }
25    for (auto [h, c] : cnth) {
26        gh = gcd(gh, c);
27    }
28    for (auto [p, c] : cnt) {

```

```

29         auto [w, h] = p;
30         if (i128(c) * tot != i128(cntw[w]) * cnth[h]) {
31             cout << 0 << "\n";
32             return 0;
33         }
34     }
35     map<i64, int> e;
36     i64 g = gcd(gw, tot);
37     gw /= g, tot /= g;
38     gh /= tot;
39     for (i64 x = 2; x * x <= gw; x++) {
40         while (gw % x == 0) {
41             e[x]++;
42             gw /= x;
43         }
44     }
45     if (gw > 1) {
46         e[gw]++;
47     }
48     for (i64 x = 2; x * x <= gh; x++) {
49         while (gh % x == 0) {
50             e[x]++;
51             gh /= x;
52         }
53     }
54     if (gh > 1) {
55         e[gh]++;
56     }
57     i64 ans = 1;
58     for (auto [_, x] : e) {
59         ans *= x + 1;
60     }
61     cout << ans << "\n";
62     return 0;
63 }
```

## 752: Array Equalizer

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Jeevan has two arrays  $a$  and  $b$  of size  $n$ . He is fond of performing weird operations on arrays. This time, he comes up with two types of operations:

Choose any  $i$  ( $1 \leq i \leq n$ ) and increment  $a_j$  by 1 for every  $j$  which is a multiple of  $i$  and  $1 \leq j \leq n$ .

Choose any  $i$  ( $1 \leq i \leq n$ ) and decrement  $a_j$  by 1 for every  $j$  which is a multiple of  $i$  and  $1 \leq j \leq n$ .

He wants to convert array  $a$  into an array  $b$  using the minimum total number of operations. However, Jeevan seems to have forgotten the value of  $b_1$ . So he makes some guesses. He will ask you  $q$  questions corresponding to his  $q$  guesses, the  $i$ -th of which is of the form:

If  $b_1 = x_i$ , what is the minimum number of operations required to convert  $a$  to  $b$ ?

Help him by answering each question.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<i64> a(n + 1);
10    for (int i = 1; i <= n; i++) {
11        cin >> a[i];
12    }
13    for (int i = 1; i <= n; i++) {
14        int b;
15        cin >> b;
16        if (i > 1) {
17            a[i] = b - a[i];
18        } else {
19            a[i] = -a[i];
20        }
21    }
22    vector<i64> mu(n + 1);
23    mu[1] = 1;
24    for (int i = 1; i <= n; i++) {
25        for (int j = 2 * i; j <= n; j += i) {
26            a[j] -= a[i];
27            mu[j] -= mu[i];
28        }
29    }
30    i64 sum = 0;
31    vector<i64> b;
32    for (int i = 1; i <= n; i++) {
33        if (mu[i] == 0) {
34            sum += abs(a[i]);
35        } else if (mu[i] == -1) {
36            b.push_back(a[i]);
37        } else {
38            b.push_back(-a[i]);
39        }
40    }
41    sort(b.begin(), b.end());
42    int m = b.size();
43    vector<i64> pre(m + 1);
44    for (int i = 0; i < m; i++) {
45        pre[i + 1] = pre[i] + b[i];
46    }
47    int q;
48    cin >> q;
49    for (int _ = 0; _ < q; _++) {
50        int x;
51        cin >> x;
52        i64 ans = sum;
53        int j = lower_bound(b.begin(), b.end(), x) - b.begin();
54        ans += 1LL * j * x - pre[j];
55        ans += pre[m] - pre[j] - 1LL * (m - j) * x;
```

```

56         cout << ans << "\n";
57     }
58 }
59 }
```

### 753: Lihmuf Balling

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

After working at a factory (Lihmuf Sorting), Lihmuf wants to play a game with his good friend, Pak Chanek. There are  $N$  boxes numbered from 1 to  $N$ . The  $i$ -th box contains  $i$  balls. Pak Chanek and Lihmuf will play a game using those boxes.

There will be  $N$  turns numbered from 1 to  $N$ . On each turn, Pak Chanek chooses one box and takes all balls from that box, then Lihmuf does the same thing for one box he chooses (it is possible that the two chosen boxes are the same).

Given an integer  $M$ . Before the game begins, Lihmuf must choose an integer  $K$  satisfying  $1 \leq K \leq M$ . It is known that on the  $j$ -th turn, Pak Chanek will choose the  $j$ -th box, while Lihmuf will choose the  $y$ -th box, with  $y = ((j \times K - 1) \bmod N) + 1$ . Help Lihmuf choose the correct value of  $K$  so that he will get as many balls as possible! If there are more than one possible values of  $K$  that result in the maximum total balls, find the minimum such value of  $K$ .

Keep in mind that each box can be chosen more than once, but after a box is chosen for the first time, the box becomes empty.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int N, M;
8     cin >> N >> M;
9     i64 ans = 0;
10    int k = 1;
11    for (int K = 1; K <= M; K++) {
12        i64 res = 0;
13        int g = gcd(K, N);
14        for (int d = 0; d < K; d++) {
15            i64 lo = (1LL * d * N + K - 1) / K;
16            i64 hi = min(1LL * N / g - 1, (1LL * (d + 1) * N - 1) / K);
17            if (K == 1) {
```

```

18         hi = -1;
19     } else {
20         lo = max(lo, 1LL * d * N / (K - 1) + 1);
21     }
22     if (lo <= hi) {
23         res += 1LL * (lo + hi) * (hi - lo + 1) / 2 * K;
24         res -= 1LL * (hi - lo + 1) * d * N;
25     }
26 }
27 if (g > 1) {
28     res += N;
29 }
30 if (res > ans) {
31     ans = res;
32     k = K;
33 }
34 }
35 cout << k << "\n";
36 return 0;
37 }

```

## 754: Ina of the Mountain

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

To prepare her “Takodachi” dumbo octopuses for world domination, Ninomae Ina’nis, a.k.a. Ina of the Mountain, orders Hoshimachi Suisei to throw boulders at them. Ina asks you, Kiryu Coco, to help choose where the boulders are thrown.

There are  $n$  octopuses on a single-file trail on Ina’s mountain, numbered  $1, 2, \dots, n$ . The  $i$ -th octopus has a certain initial health value  $a_i$ , where  $1 \leq a_i \leq k$ .

Each boulder crushes consecutive octopuses with indexes  $l, l + 1, \dots, r$ , where  $1 \leq l \leq r \leq n$ . You can choose the numbers  $l$  and  $r$  arbitrarily for each boulder.

For each boulder, the health value of each octopus the boulder crushes is reduced by 1. However, as octopuses are immortal, once they reach a health value of 0, they will immediately regenerate to a health value of  $k$ .

Given the octopuses’ initial health values, find the minimum number of boulders that need to be thrown to make the health of all octopuses equal to  $k$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     vector<int> a(n + 2);
8     priority_queue<i64, vector<i64>, greater<i64>> q;
9     i64 add = 0;
10    for (int i = 1; i <= n; i++) {
11        cin >> a[i];
12        if (a[i] == k) {
13            a[i] = 0;
14        }
15    }
16    i64 ans = 0;
17    int p = 0;
18    for (int i = 1; i <= n + 1; i++) {
19        if (a[i] >= a[i - 1]) {
20            q.push(a[i] - a[i - 1]);
21        } else {
22            q.push(0);
23            q.push(k + a[i] - a[i - 1]);
24        }
25        ans += q.top();
26        q.pop();
27    }
28    cout << ans << "\n";
29 }
30 int main() {
31     ios::sync_with_stdio(false);
32     cin.tie(nullptr);
33     int t;
34     cin >> t;
35     while (t--) {
36         solve();
37     }
38     return 0;
39 }
```

## 755: The Boss's Identity

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

While tracking Diavolo's origins, Giorno receives a secret code from Polnareff. The code can be represented as an infinite sequence of positive integers: \$a\_1, a\_2, \dots\$. Giorno immediately sees the pattern behind the code. The first  $n$  numbers  $a_1, a_2, \dots, a_n$  are given. For  $i > n$  the value of  $a_i$  is  $(a_{i-n} \mid a_{i-n+1})$ , where  $\mid$  denotes the bitwise OR operator.

Pieces of information about Diavolo are hidden in  $q$  questions. Each question has a positive integer  $v$  associated with it and its answer is the smallest index  $i$  such that  $a_i > v$ . If no such  $i$  exists, the answer

is –1. Help Giorno in answering the questions!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, q;
6     cin >> n >> q;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<pair<int, i64>> e;
12    for (int i = 0; i < n; i++) {
13        e.emplace_back(a[i], i);
14    }
15    vector<int> f(30, -1);
16    for (int i = 0; i < n; i++) {
17        for (int j = 0; j < 30; j++) {
18            if (a[i] >> j & 1) {
19                f[j] = i;
20            }
21        }
22    }
23    for (int i = 0; i < n; i++) {
24        for (int j = 0; j < 30; j++) {
25            if (a[i] >> j & 1) {
26                f[j] = i;
27            }
28        }
29        if (i > 0) {
30            vector<int> o;
31            for (int j = 0; j < 30; j++) {
32                if (f[j] != -1) {
33                    o.push_back(j);
34                }
35            }
36            sort(o.begin(), o.end(),
37                  [&](int x, int y) {
38                      return (i - f[x] + n) % n < (i - f[y] + n) % n;
39                  });
40            int v = 0;
41            for (auto j : o) {
42                v |= 1 << j;
43                if (f[j] != i) {
44                    i64 t = n;
45                    t += 1LL * ((i - f[j] + n) % n - 1) * (n - 1);
46                    t += (i - 1);
47                    e.emplace_back(v, t);
48                }
49            }
50        }
51    }
52    sort(e.begin(), e.end());
53    int i = 0;
54    for (int j = 0; j < e.size(); j++) {
55        while (i > 0 && e[j].second < e[i - 1].second) {

```

```

56         i--;
57     }
58     e[i++] = e[j];
59 }
60 e.resize(i);
61 while (q--) {
62     int v;
63     cin >> v;
64     auto it = lower_bound(e.begin(), e.end(), make_pair(v + 1, 0LL));
65     if (it == e.end()) {
66         cout << -1 << "\n";
67     } else {
68         cout << it->second + 1 << "\n";
69     }
70 }
71 }
72 int main() {
73     ios::sync_with_stdio(false);
74     cin.tie(nullptr);
75     int t;
76     cin >> t;
77     while (t--) {
78         solve();
79     }
80     return 0;
81 }
```

## 756: Igor and Interesting Numbers

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Igor likes hexadecimal notation and considers positive integer in the hexadecimal notation interesting if each digit and each letter in it appears no more than  $t$  times. For example, if  $t = 3$ , then integers 13a13322, aaa, abcdef0123456789 are interesting, but numbers aaaa, abababab and 1000000 are not interesting.

Your task is to find the  $k$ -th smallest interesting for Igor integer in the hexadecimal notation. The integer should not contain leading zeros.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     i64 k;
8     int t;
```

```
9      cin >> k >> t;
10     k--;
11     i64 binom[20][20] = {};
12     for (int i = 0; i < 20; i++) {
13         binom[i][0] = 1;
14         for (int j = 1; j <= i; j++) {
15             binom[i][j] = min(binom[i - 1][j] + binom[i - 1][j - 1], k + 1);
16         }
17     }
18     auto get = [&](array<int, 16> f, int n) {
19         vector<i64> dp(n + 1);
20         dp[0] = 1;
21         for (int i = 0; i < 16; i++) {
22             for (int x = n; x >= 0; x--) {
23                 i64 v = dp[x];
24                 dp[x] = 0;
25                 for (int y = 0; y + f[i] <= t && x + y <= n; y++) {
26                     dp[x + y] = min(k + 1, dp[x + y] + v * binom[x + y][x]);
27                 }
28             }
29         }
30         return dp[n];
31     };
32     for (int len = 1; ; len++) {
33         for (int i = 1; i < 16; i++) {
34             array<int, 16> f{};
35             f[i] = 1;
36             i64 res = get(f, len - 1);
37             if (res <= k) {
38                 k -= res;
39                 continue;
40             }
41             vector<int> a{i};
42             for (int j = 1; j < len; j++) {
43                 for (int x = 0; x < 16; x++) {
44                     f[x]++;
45                     res = get(f, len - j - 1);
46                     if (k >= res) {
47                         k -= res;
48                         f[x]--;
49                         continue;
50                     }
51                     a.push_back(x);
52                     break;
53                 }
54             }
55             for (auto x : a) {
56                 if (x < 10) {
57                     cout << x;
58                 } else {
59                     cout << char('a' + x - 10);
60                 }
61             }
62             cout << "\n";
63             return 0;
64         }
65     }
66     return 0;
67 }
```

## 757: Vika and Wiki

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Recently, Vika was studying her favorite internet resource - Wikipedia.

On the expanses of Wikipedia, she read about an interesting mathematical operation bitwise XOR, denoted by  $\oplus$ .

Vika began to study the properties of this mysterious operation. To do this, she took an array  $a$  consisting of  $n$  non-negative integers and applied the following operation to all its elements at the same time:  $a_i = a_i \oplus a_{(i+1) \bmod n}$ . Here  $x \bmod y$  denotes the remainder of dividing  $x$  by  $y$ . The elements of the array are numbered starting from 0.

Since it is not enough to perform the above actions once for a complete study, Vika repeats them until the array  $a$  becomes all zeros.

Determine how many of the above actions it will take to make all elements of the array  $a$  zero. If this moment never comes, output  $-1$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    if (a == vector(n, 0)) {
14        cout << 0 << "\n";
15        return 0;
16    }
17    int ans = 1;
18    for (int i = n / 2; i > 0; i /= 2) {
19        vector<int> b(n);
20        for (int j = 0; j < n; j++) {
21            b[j] = a[j] ^ a[(i + j) % n];
22        }
23        if (b != vector(n, 0)) {
24            a = move(b);
25            ans += i;

```

```
26     }
27 }
28 cout << ans << "\n";
29 return 0;
30 }
```

## 758: Vika and Stone Skipping

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In Vika's hometown, Vladivostok, there is a beautiful sea.

Often you can see kids skimming stones. This is the process of throwing a stone into the sea at a small angle, causing it to fly far and bounce several times off the water surface.

Vika has skimmed stones many times and knows that if you throw a stone from the shore perpendicular to the coastline with a force of  $f$ , it will first touch the water at a distance of  $f$  from the shore, then bounce off and touch the water again at a distance of  $f - 1$  from the previous point of contact. The stone will continue to fly in a straight line, reducing the distances between the points where it touches the water, until it falls into the sea.

Formally, the points at which the stone touches the water surface will have the following coordinates:  $f, f + (f - 1), f + (f - 1) + (f - 2), \dots, f + (f - 1) + (f - 2) + \dots + 1$  (assuming that 0 is the coordinate of the shoreline).

Once, while walking along the embankment of Vladivostok in the evening, Vika saw a group of guys skipping stones across the sea, launching them from the same point with different forces.

She became interested in what is the maximum number of guys who can launch a stone with their force  $f_i$ , so that all  $f_i$  are different positive integers, and all  $n$  stones touched the water at the point with the coordinate  $x$  (assuming that 0 is the coordinate of the shoreline).

After thinking a little, Vika answered her question. After that, she began to analyze how the answer to her question would change if she multiplied the coordinate  $x$  by some positive integers  $x_1, x_2, \dots, x_q$ , which she picked for analysis.

Vika finds it difficult to cope with such analysis on her own, so she turned to you for help.

Formally, Vika is interested in the answer to her question for the coordinates  $X_1 = x \cdot x_1, X_2 = X_1 \cdot x_2, \dots, X_q = X_{q-1} \cdot x_q$ . Since the answer for such coordinates can be quite large, find it modulo  $M$ . It is guaranteed that  $M$  is prime.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 using Z = MInt<0>;
33 constexpr int N = 1E6;
34 vector<int> minp, primes;
35 void sieve(int n) {
36     minp.assign(n + 1, 0);
37     primes.clear();
38     for (int i = 2; i <= n; i++) {
39         if (minp[i] == 0) {
40             minp[i] = i;
41             primes.push_back(i);
42         }
43         for (auto p : primes) {
44             if (i * p > n) {
45                 break;
46             }
47             minp[i * p] = p;
48             if (p == minp[i]) {
49                 break;
50             }
51         }
52     }
53 }
54 # struct Comb comb;
55 int main() {
56     ios::sync_with_stdio(false);
57     cin.tie(nullptr);
58     sieve(N);

```

```

59     int x, q, M;
60     cin >> x >> q >> M;
61     Z::setMod(M);
62     Z ans = 1;
63     vector<Z> f(N, 1);
64     x >>= __builtin_ctz(x);
65     for (int i = 2; i * i <= x; i++) {
66         if (x % i == 0) {
67             int t = 0;
68             while (x % i == 0) {
69                 x /= i;
70                 t++;
71             }
72             ans *= (t + 1);
73             if (i < N) {
74                 f[i] = t + 1;
75             }
76         }
77     }
78     if (x > 1) {
79         ans *= 2;
80         if (x < N) {
81             f[x] = 2;
82         }
83     }
84     int zero = 0;
85     while (q--) {
86         cin >> x;
87         x >>= __builtin_ctz(x);
88         while (x > 1) {
89             int p = minp[x];
90             x /= p;
91             if (f[p] == 0) {
92                 zero--;
93             } else {
94                 ans *= comb.inv(f[p].val());
95             }
96             f[p] += 1;
97             if (f[p] == 0) {
98                 zero++;
99             } else {
100                 ans *= f[p];
101             }
102         }
103         cout << (zero > 0 ? 0 : ans) << "\n";
104     }
105     return 0;
106 }
```

**759: Min Cost Permutation (Easy Version)**

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The only difference between this problem and the hard version is the constraints on  $t$  and  $n$ .

You are given an array of  $n$  positive integers  $a_1, \dots, a_n$ , and a (possibly negative) integer  $c$ .

Across all permutations  $b_1, \dots, b_n$  of the array  $a_1, \dots, a_n$ , consider the minimum possible value of

$$\sum_{i=1}^{n-1} |b_{i+1} - b_i - c|.$$

Find the lexicographically smallest permutation  $b$  of the array  $a$  that achieves this minimum.

A sequence  $x$  is lexicographically smaller than a sequence  $y$  if and only if one of the following holds:

$x$  is a prefix of  $y$ , but  $x \neq y$ ;

in the first position where  $x$  and  $y$  differ, the sequence  $x$  has a smaller element than the corresponding element in  $y$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, c;
6     cin >> n >> c;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    if (c >= 0) {
12        sort(a.begin(), a.end());
13        for (int i = 0; i < n; i++) {
14            cout << a[i] << " \n"[i == n - 1];
15        }
16        return;
17    }
18    sort(a.begin(), a.end(), greater());
19    c = -c;
20    vector<int> ans;
21    for (int i = 1, j = 0; i <= n; i++) {
22        if (i == n || a[i - 1] - a[i] > c) {
23            set<pair<int, int>> s, v;
24            for (int k = j; k < i; k++) {
25                s.emplace(a[k], k);
26                v.emplace(a[k], k);
27            }
28            auto x = *s.rbegin();
29            ans.push_back(x.first);
30            s.extract(x);
31            int lst = x.first;
32            v.erase(v.begin());
33            while (!s.empty()) {
34                bool ok = false;
35                while (true) {
36                    auto a = v.lower_bound(pair(lst - c, 0));
37                    if (a == v.end()) {
38                        break;
39                    }

```

```

40             auto x = *a;
41             auto it = s.find(x);
42             v.erase(a);
43             assert(it != s.begin());
44             if (it == s.end()) {
45                 continue;
46             }
47             auto r = next(it);
48             if (r == s.end()) {
49                 continue;
50             }
51             auto l = prev(it);
52             if (r->first - l->first > c) {
53                 continue;
54             }
55             ok = true;
56             ans.push_back(it->first);
57             lst = it->first;
58             s.erase(it);
59             break;
60         }
61     if (!ok) {
62         auto x = *s.rbegin();
63         ans.push_back(x.first);
64         lst = x.first;
65         s.extract(x);
66     }
67 }
68 j = i;
69 }
70 for (int i = 0; i < n; i++) {
71     cout << ans[i] << " \n"[i == n - 1];
72 }
73 }
74 int main() {
75     ios::sync_with_stdio(false);
76     cin.tie(nullptr);
77     int t;
78     cin >> t;
79     while (t--) {
80         solve();
81     }
82     return 0;
83 }
```

## 760: Boxes and Balls

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  boxes placed in a line. The boxes are numbered from 1 to  $n$ . Some boxes contain one ball inside of them, the rest are empty. At least one box contains a ball and at least one box is empty.

In one move, you have to choose a box with a ball inside and an adjacent empty box and move the ball

from one box into another. Boxes  $i$  and  $i + 1$  for all  $i$  from 1 to  $n - 1$  are considered adjacent to each other. Boxes 1 and  $n$  are not adjacent.

How many different arrangements of balls exist after exactly  $k$  moves are performed? Two arrangements are considered different if there is at least one such box that it contains a ball in one of them and doesn't contain a ball in the other one.

Since the answer might be pretty large, print its remainder modulo  $10^9 + 7$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int n, k;
38     cin >> n >> k;
39     vector<int> a(n);
40     for (int i = 0; i < n; i++) {
41         cin >> a[i];
42     }
43     vector<vector<Z>> dp(81, vector<Z>(k + 1));
44     dp[40][0] = 1;
45     for (int i = 0; i < n; i++) {
46         vector<Z> g(81, vector<Z>(k + 1));

```

```

47         for (int j = -40; j <= 40; j++) {
48             for (int x = 0; x < 2; x++) {
49                 int nj = j + x - a[i];
50                 if (abs(nj) > 40) {
51                     continue;
52                 }
53                 for (int s = 0; s + abs(nj) <= k; s++) {
54                     g[nj + 40][s + abs(nj)] += dp[j + 40][s];
55                 }
56             }
57         }
58     swap(dp, g);
59 }
60 Z ans = 0;
61 for (int i = k; i >= 0; i -= 2) {
62     ans += dp[40][i];
63 }
64 cout << ans << "\n";
65 return 0;
66 }
```

## 761: Lottery

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

$n$  people indexed with integers from 1 to  $n$  came to take part in a lottery. Each received a ticket with an integer from 0 to  $m$ .

In a lottery, one integer called target is drawn uniformly from 0 to  $m$ .  $k$  tickets (or less, if there are not enough participants) with the closest numbers to the target are declared the winners. In case of a draw, a ticket belonging to the person with a smaller index is declared a winner.

Bytek decided to take part in the lottery. He knows the values on the tickets of all previous participants. He can pick whatever value he wants on his ticket, but unfortunately, as he is the last one to receive it, he is indexed with an integer  $n + 1$ .

Bytek wants to win the lottery. Thus, he wants to know what he should pick to maximize the chance of winning. He wants to know the smallest integer in case there are many such integers. Your task is to find it and calculate his chance of winning.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
```

```

5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     i64 m;
9     int k;
10    cin >> n >> m >> k;
11    vector<i64> a(n);
12    for (int i = 0; i < n; i++) {
13        cin >> a[i];
14    }
15    if (k > n) {
16        cout << m + 1 << " " << 0 << "\n";
17        return 0;
18    }
19    sort(a.begin(), a.end());
20    auto cand = a;
21    cand.push_back(0);
22    for (int v : {k - 1, k, k + 1}) {
23        for (int i = 0; i + v < n; i++) {
24            cand.push_back((a[i] + a[i + v]) / 2);
25        }
26    }
27    sort(cand.begin(), cand.end());
28    i64 val = 0;
29    i64 ans = 0;
30    i64 y = 0;
31    for (int i = -1, j = 0; auto x : cand) {
32        y = max(y, x - 10);
33        while (y < x + 10 && y <= m) {
34            while (i + 1 < n && a[i + 1] <= y) {
35                i++;
36            }
37            while (j < n && a[j] < y) {
38                j++;
39            }
40            if (i - j + 1 >= k) {
41                y++;
42                continue;
43            }
44            i64 lo = 0, hi = m;
45            if (i - k + 1 >= 0) {
46                lo = max(lo, (y + a[i - k + 1]) / 2 + 1);
47            }
48            if (j + k - 1 < n) {
49                hi = min(hi, (y + a[j + k - 1] - 1) / 2);
50            }
51            if (hi - lo + 1 > ans) {
52                ans = hi - lo + 1;
53                val = y;
54            }
55            y++;
56        }
57    }
58    cout << ans << " " << val << "\n";
59    return 0;
60 }

```

**762: Twin Clusters**

- Time limit: 2 seconds

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Famous worldwide astrophysicist Mleil waGrasse Tysok recently read about the existence of twin galaxy clusters. Before he shares this knowledge with the broader audience in his podcast called S.tarT-ok, he wants to prove their presence on his own. Mleil is aware that the vastness of the universe is astounding (almost as astounding as his observation skills) and decides to try his luck and find some new pair of twin clusters.

To do so, he used his TLEscope to observe a part of the night sky that was not yet examined by humanity in which there are exactly  $2^{k+1}$  galaxies in a row.  $i$ -th of them consist of exactly  $0 \leq g_i < 4^k$  stars.

A galaxy cluster is any non-empty contiguous segment of galaxies. Moreover, its' trait is said to be equal to the bitwise XOR of all values  $g_i$  within this range.

Two galaxy clusters are considered twins if and only if they have the same traits and their corresponding segments are disjoint.

Write a program that, for many scenarios, will read a description of a night sky part observed by Mleil and outputs a location of two intervals belonging to some twin clusters pair, or a single value  $-1$  if no such pair exists.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
5 void solve() {
6     int k;
7     cin >> k;
8     int n = 2 << k;
9     vector<i64> g(n);
10    for (int i = 0; i < n; i++) {
11        cin >> g[i];
12        // g[i] = rng() % (1LL << (2 * k));
13    }
14    vector<i64> s(n + 1);
15    for (int i = 0; i < n; i++) {
16        s[i + 1] = s[i] ^ g[i];
17    }
18    vector<int> p(n + 1);
19    iota(p.begin(), p.end(), 0);
20    sort(p.begin(), p.end(), [&](int i, int j) {
21        return s[i] < s[j] || (s[i] == s[j] && i < j);
22    });
23    for (int i = 0; i + 2 <= n; i++) {
24        if (s[p[i]] == s[p[i + 2]]) {
25            cout << p[i] + 1 << " " << p[i + 1] << " " << p[i + 1] + 1 << " " << p[i + 2]
26            << "\n";
27        }
28    }
29}
```

```

27         }
28     }
29     int j = -1;
30     for (int i = 0; i + 1 <= n; i++) {
31         if (s[p[i]] == s[p[i + 1]]) {
32             if (j != -1) {
33                 vector a{p[j], p[j + 1], p[i], p[i + 1]};
34                 sort(a.begin(), a.end());
35                 cout << a[0] + 1 << " " << a[1] << " " << a[2] + 1 << " " << a[3] << "\n";
36                 return;
37             } else {
38                 j = i;
39             }
40         }
41     }
42     map<i64, array<int, 2>> f;
43     while (true) {
44         int l = rng() % (n + 1);
45         int r = rng() % (n + 1);
46         while ((j != -1 && (l == p[j] || r == p[j])) || l == r) {
47             l = rng() % (n + 1);
48             r = rng() % (n + 1);
49         }
50         if (l > r) {
51             swap(l, r);
52         }
53         if (f.contains(s[l] ^ s[r]) && f[s[l] ^ s[r]] != array{l, r}) {
54             auto [x, y] = f[s[l] ^ s[r]];
55             vector a{l, r, x, y};
56             sort(a.begin(), a.end());
57             cout << a[0] + 1 << " " << a[1] << " " << a[2] + 1 << " " << a[3] << "\n";
58             return;
59         } else {
60             f[s[l] ^ s[r]] = {l, r};
61         }
62     }
63 }
64 int main() {
65     ios::sync_with_stdio(false);
66     cin.tie(nullptr);
67     int t;
68     cin >> t;
69     while (t--) {
70         solve();
71     }
72     return 0;
73 }
```

### 763: Typewriter

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Recently, Polycarp was given an unusual typewriter as a gift! Unfortunately, the typewriter was defective and had a rather strange design.

The typewriter consists of  $n$  cells numbered from left to right from 1 to  $n$ , and a carriage that moves over them. The typewriter cells contain  $n$  distinct integers from 1 to  $n$ , and each cell  $i$  initially contains the integer  $p_i$ . Before all actions, the carriage is at cell number 1 and there is nothing in its buffer storage. The cell on which the carriage is located is called the current cell.

The carriage can perform five types of operations:

Take the integer from the current cell, if it is not empty, and put it in the carriage buffer, if it is empty (this buffer can contain no more than one integer).

Put the integer from the carriage buffer, if it is not empty, into the current cell, if it is empty.

Swap the number in the carriage buffer with the number in the current cell, if both the buffer and the cell contain integers.

Move the carriage from the current cell  $i$  to cell  $i + 1$  (if  $i < n$ ), while the integer in the buffer is preserved.

Reset the carriage, i.e. move it to cell number 1, while the integer in the buffer is preserved.

Polycarp was very interested in this typewriter, so he asks you to help him understand it and will ask you  $q$  queries of three types:

Perform a cyclic shift of the sequence  $p$  to the left by  $k_j$ .

Perform a cyclic shift of the sequence  $p$  to the right by  $k_j$ .

Reverse the sequence  $p$ .

Before and after each query, Polycarp wants to know what minimum number of carriage resets is needed for the current sequence in order to distribute the numbers to their cells (so that the number  $i$  ends up in cell number  $i$ ).

Note that Polycarp only wants to know the minimum number of carriage resets required to arrange the numbers in their places, but he does not actually distribute them.

Help Polycarp find the answers to his queries!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;

```

```

9     vector<int> p(n);
10    for (int i = 0; i < n; i++) {
11        cin >> p[i];
12        p[i]--;
13    }
14    auto solve = [&]() {
15        vector<int> ans(n + 1);
16        for (int i = 0; i < n; i++) {
17            if (p[i] <= i) {
18                ans[0]++;
19            }
20            ans[(i - p[i] + n) % n]--;
21        }
22        ans.resize(n);
23        for (int i = 1; i < n; i++) {
24            ans[i]++;
25            ans[i] += ans[i - 1];
26        }
27        return ans;
28    };
29    vector<int> ans[2];
30    ans[0] = solve();
31    reverse(p.begin() + 1, p.end());
32    ans[1] = solve();
33    reverse(ans[1].begin() + 1, ans[1].end());
34    int k = 1, b = 0;
35    cout << ans[k == 1 ? 0 : 1][b] << "\n";
36    int q;
37    cin >> q;
38    while (q--) {
39        int o;
40        cin >> o;
41        if (o == 1) {
42            int v;
43            cin >> v;
44            b = (b + v * k + n) % n;
45        } else if (o == 2) {
46            int v;
47            cin >> v;
48            b = (b - v * k + n) % n;
49        } else {
50            k *= -1;
51            b = (b + n + k) % n;
52        }
53        cout << ans[k == 1 ? 0 : 1][b] << "\n";
54    }
55    return 0;
56 }

```

## 764: Count Supersequences

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  of  $n$  integers, where all elements  $a_i$  lie in the range  $[1, k]$ . How many different arrays  $b$  of  $m$  integers, where all elements  $b_i$  lie in the range  $[1, k]$ , contain  $a$  as a subsequence? Two

arrays are considered different if they differ in at least one position.

A sequence  $x$  is a subsequence of a sequence  $y$  if  $x$  can be obtained from  $y$  by the deletion of several (possibly, zero or all) elements.

Since the answer may be large, print it modulo  $10^9 + 7$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 void solve() {
36     int n, m, k;
37     cin >> n >> m >> k;
38     vector<int> a(n);
39     for (int i = 0; i < n; i++) {
40         cin >> a[i];
41     }
42     m -= n;
43     Z ans = 0;
44     ans += power(1 - Z(k - 1) / k, P - 1 - n) * power(Z(k), m);
45     vector<Z> f(n);
46     for (int i = 0; i < n; i++) {
47         f[i] = -comb.binom(n, i) * power(-Z(k - 1), i) * power(Z(k), n);
48     }
49     f[0] += 1;
50     for (int i = 1; i < n; i++) {
51         f[i] += f[i - 1] * k;
52     }
53 }
```

```

52     }
53     Z v = 1;
54     for (int i = 1; i < n; i++) {
55         v *= m + i;
56         v /= i;
57     }
58     for (int i = 0; i < n && i <= m; i++) {
59         if (i > 0) {
60             v *= m - i + 1;
61             v /= m - i + n;
62         }
63         ans += f[i] * v * power(Z(k - 1), m - i);
64     }
65     cout << ans << "\n";
66 }
67 int main() {
68     ios::sync_with_stdio(false);
69     cin.tie(nullptr);
70     int t;
71     cin >> t;
72     while (t--) {
73         solve();
74     }
75     return 0;
76 }
```

## misc

### 765: Hypercatapult Commute

- Time limit: 3 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

A revolutionary new transport system is currently operating in Byteland. This system requires neither roads nor sophisticated mechanisms, only giant catapults.

The system works as follows. There are  $n$  cities in Byteland. In every city there is a catapult, right in the city center. People who want to travel are put in a special capsule, and a catapult throws this capsule to the center of some other city. Every catapult is powerful enough to throw the capsule to any other city, with any number of passengers inside the capsule. The only problem is that it takes a long time to charge the catapult, so it is only possible to use it once a day.

The passenger may need to use the catapults multiple times. For example, if the passenger wants to travel from city A to city B, they can first use one catapult to move from A to C, and then transfer to another catapult to move from C to B.

Today there are  $m$  passengers. Passenger  $i$  wants to travel from city  $a_i$  to city  $b_i$ . Your task is to find the way to deliver all the passengers to their destinations in a single day, using the minimal possible

number of catapults, or say that it is impossible.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, m;
9     cin >> n >> m;
10    DSU dsu(n);
11    vector<vector<int>> adj(n);
12    for (int i = 0; i < m; i++) {
13        int u, v;
14        cin >> u >> v;
15        u--, v--;
16        adj[u].push_back(v);
17        dsu.merge(u, v);
18    }
19    vector<array<int, 2>> ans;
20    vector<int> t;
21    auto topo = [&](auto &a, int f = -1) {
22        vector<int> deg(n);
23        for (auto x : a) {
24            for (auto y : adj[x]) {
25                if (y != f) {
26                    deg[y]++;
27                }
28            }
29        }
30        t.clear();
31        for (auto x : a) {
32            if (deg[x] == 0) {
33                t.push_back(x);
34            }
35        }
36        for (int i = 0; i < t.size(); i++) {
37            int x = t[i];
38            for (auto y : adj[x]) {
39                if (y != f) {
40                    deg[y]--;
41                    if (deg[y] == 0) {
42                        t.push_back(y);
43                    }
44                }
45            }
46        }
47        return t.size() == a.size();
48    };
49    for (int i = 0; i < n; i++) {
50        if (dsu.find(i) == i) {
51            vector<int> a;
52            for (int j = 0; j < n; j++) {
53                if (dsu.find(j) == i) {
54                    a.push_back(j);
55                }
56            }
57            ans.push_back({a, i});
58        }
59    }
60}
```

```

56         }
57         int f = -1;
58         if (!topo(a)) {
59             bool ok = false;
60             for (auto x : a) {
61                 if (topo(a, x)) {
62                     ok = true;
63                     f = x;
64                     break;
65                 }
66             }
67             if (!ok) {
68                 cout << -1 << "\n";
69                 return 0;
70             }
71         }
72         for (int j = 1; j < t.size(); j++) {
73             ans.push_back({t[j - 1], t[j]});
74         }
75         if (f != -1) {
76             ans.push_back({t.back(), f});
77         }
78     }
79 }
80 cout << ans.size() << "\n";
81 for (auto [x, y] : ans) {
82     cout << x + 1 << " " << y + 1 << "\n";
83 }
84 return 0;
85 }
```

## 766: Matrix Problem

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a matrix  $a$ , consisting of  $n$  rows by  $m$  columns. Each element of the matrix is equal to 0 or 1.

You can perform the following operation any number of times (possibly zero): choose an element of the matrix and replace it with either 0 or 1.

You are also given two arrays  $A$  and  $B$  (of length  $n$  and  $m$  respectively). After you perform the operations, the matrix should satisfy the following conditions:

the number of ones in the  $i$ -th row of the matrix should be exactly  $A_i$  for every  $i \in [1, n]$ .

the number of ones in the  $j$ -th column of the matrix should be exactly  $B_j$  for every  $j \in [1, m]$ .

Calculate the minimum number of operations you have to perform.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 # struct MinCostFlow;
6 int main() {
7     ios::sync_with_stdio(false);
8     cin.tie(nullptr);
9     int n, m;
10    cin >> n >> m;
11    vector<vector<int>> a(n, vector<int>(m));
12    for (int i = 0; i < n; i++) {
13        for (int j = 0; j < m; j++) {
14            cin >> a[i][j];
15        }
16    }
17    vector<int> A(n), B(m);
18    for (int i = 0; i < n; i++) {
19        cin >> A[i];
20    }
21    for (int i = 0; i < m; i++) {
22        cin >> B[i];
23    }
24    MinCostFlow<int> g(n + m + 2);
25    for (int i = 0; i < n; i++) {
26        for (int j = 0; j < m; j++) {
27            if (a[i][j] == 0) {
28                g.addEdge(i, n + j, 1, 1);
29            } else {
30                g.addEdge(n + j, i, 1, 1);
31                A[i] -= 1;
32                B[j] -= 1;
33            }
34        }
35    }
36    int S = n + m, T = S + 1;
37    int sum1 = 0, sum2 = 0;
38    for (int i = 0; i < n; i++) {
39        if (A[i] > 0) {
40            g.addEdge(S, i, A[i], 0);
41            sum1 += A[i];
42        } else {
43            g.addEdge(i, T, -A[i], 0);
44            sum2 += -A[i];
45        }
46    }
47    for (int j = 0; j < m; j++) {
48        if (B[j] > 0) {
49            g.addEdge(n + j, T, B[j], 0);
50            sum2 += B[j];
51        } else {
52            g.addEdge(S, n + j, -B[j], 0);
53            sum1 += -B[j];
54        }
55    }
56    if (sum1 != sum2) {
57        cout << -1 << "\n";
58        return 0;
59    }
60    auto [f, c] = g.flow(S, T);
```

```

61     if (f != sum1) {
62         cout << -1 << "\n";
63         return 0;
64     }
65     cout << c << "\n";
66     return 0;
67 }
```

## 767: Maximize Mex

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  students and  $m$  clubs in a college. The clubs are numbered from 1 to  $m$ . Each student has a potential  $p_i$  and is a member of the club with index  $c_i$ . Initially, each student is a member of exactly one club. A technical fest starts in the college, and it will run for the next  $d$  days. There is a coding competition every day in the technical fest.

Every day, in the morning, exactly one student of the college leaves their club. Once a student leaves their club, they will never join any club again. Every day, in the afternoon, the director of the college will select one student from each club (in case some club has no members, nobody is selected from that club) to form a team for this day's coding competition. The strength of a team is the mex of potentials of the students in the team. The director wants to know the maximum possible strength of the team for each of the coming  $d$  days. Thus, every day the director chooses such team, that the team strength is maximized.

The mex of the multiset  $S$  is the smallest non-negative integer that is not present in  $S$ . For example, the mex of the  $\{0, 1, 1, 2, 4, 5, 9\}$  is 3, the mex of  $\{1, 2, 3\}$  is 0 and the mex of  $\emptyset$  (empty set) is 0.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<int> p(n);
10    for (int i = 0; i < n; i++) {
11        cin >> p[i];
12    }
13    vector<int> c(n);
14    for (int i = 0; i < n; i++) {
```

```
15         cin >> c[i];
16         c[i]--;
17     }
18     int d;
19     cin >> d;
20     vector<int> k(d), del(n);
21     for (int i = 0; i < d; i++) {
22         cin >> k[i];
23         k[i]--;
24         del[k[i]] = 1;
25     }
26     vector<vector<int>> adj(m);
27     vector<int> yx, xy(m, -1);
28     int ans = 0;
29     auto add = [&](int x) {
30         adj[c[x]].push_back(p[x]);
31         while (true) {
32             yx.resize(ans + 1, -1);
33             vector<int> p(m, -1);
34             vector<bool> vis(m);
35             queue<int> q;
36             bool ok = false;
37             for (int i = 0; i < m; i++) {
38                 if (yx[i] == -1) {
39                     vis[i] = true;
40                     q.push(i);
41                 }
42             }
43             while (!q.empty()) {
44                 int x = q.front();
45                 q.pop();
46                 for (auto y : adj[x]) {
47                     if (y <= ans) {
48                         if (yx[y] == -1) {
49                             while (y != -1) {
50                                 int z = xy[x];
51                                 xy[x] = y;
52                                 yx[y] = x;
53                                 y = z;
54                                 x = p[x];
55                             }
56                             ok = true;
57                             break;
58                         } else if (!vis[yx[y]]) {
59                             p[yx[y]] = x;
60                             vis[yx[y]] = true;
61                             q.push(yx[y]);
62                         }
63                     }
64                 }
65                 if (ok) {
66                     break;
67                 }
68             }
69             if (!ok) {
70                 break;
71             }
72             ans++;
73         }
74     };
75     for (int i = 0; i < n; i++) {
76         if (!del[i]) {
77             add(i);
78         }
79     }
```

```

79      }
80      vector<int> out(d);
81      for (int i = d - 1; i >= 0; i--) {
82          out[i] = ans;
83          add(k[i]);
84      }
85      for (int i = 0; i < d; i++) {
86          cout << out[i] << "\n";
87      }
88  }
89 }
```

## 768: Lexichromatography

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Pak Chanek loves his faculty, the Faculty of Computer Science, University of Indonesia (Fasilkom). He wants to play with the colours of the faculty's logo, blue and red.

There is an array  $a$  consisting of  $n$  elements, element  $i$  has a value of  $a_i$ . Pak Chanek wants to colour each element in the array blue or red such that these following conditions are satisfied:

If all blue elements are formed into a subsequence<sup>†</sup> and so are all the red elements, the blue subsequence is strictly less than the red subsequence lexicographically<sup>‡</sup>.

Array  $a$  does not have any subarray that is imbalanced. A subarray is imbalanced if and only if there is a value  $k$  such that the absolute difference between the number of blue elements with value  $k$  and the number of red elements with value  $k$  in this subarray is 2 or more.

Note that it is possible to colour every element of the array the same colour.

How many different colourings satisfy all those conditions? Since the answer can be very big, print the answer modulo 998 244 353. Two colourings are different if and only if there is at least one element that is blue in one colouring, but red in the other.

<sup>†</sup> A subsequence of an array is a sequence that can be obtained from the array by deleting some elements (possibly none), without changing the order of the remaining elements.

<sup>‡</sup> Let  $p$  and  $q$  be two different sequences. Sequence  $p$  is said to be lexicographically less than sequence  $q$  if and only if  $p$  is a prefix of  $q$  or there is an index  $i$  such that  $p_j = q_j$  holds for every  $1 \leq j < i$ , and  $p_i < q_i$ . In particular, an empty sequence is always lexicographically less than any non-empty sequence.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 # struct DSU;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int n;
39     cin >> n;
40     vector<int> a(n);
41     int m = 0;
42     for (int i = 0; i < n; i++) {
43         cin >> a[i];
44         m = max(m, a[i]);
45         a[i]--;
46     }
47     vector<int> cnt(m);
48     for (auto x : a) {
49         cnt[x]++;
50     }
51     Z ans = 1;
52     int tot = 0;
53     for (int i = 0; i < m; i++) {
54         if (cnt[i]) {
55             ans *= 2;
56             tot++;
57         }
58     }
59     bool good = true;
60     for (int i = 0; i < m; i++) {
61         if (cnt[i] % 2) {
62             good = false;
63         }
64     }
65 }
```

```

63         }
64     }
65     if (good) {
66         vector<int> cur(m);
67         DSU dsu(m);
68         int u = m;
69         vector<int> x, y;
70         for (int i = 0, j = 0; i < n; i++) {
71             if (cur[a[i]] % 2 == 0) {
72                 x.push_back(a[i]);
73             } else {
74                 y.push_back(a[i]);
75             }
76             u -= cur[a[i]] % 2 == 0;
77             cur[a[i]]++;
78             u += cur[a[i]] % 2 == 0;
79             if (u == m) {
80                 for (int x = j; x < i; x++) {
81                     tot -= dsu.merge(a[x], a[x + 1]);
82                 }
83                 j = i + 1;
84             }
85         }
86         if (x == y) {
87             ans -= power(Z(2), tot);
88         }
89     }
90     ans /= 2;
91     cout << ans << "\n";
92     return 0;
93 }
```

## 769: Weights

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $A$  of length  $N$  weights of masses  $A_1, A_2 \dots A_N$ . No two weights have the same mass. You can put every weight on one side of the balance (left or right). You don't have to put weights in order  $A_1, \dots, A_N$ . There is also a string  $S$  consisting of characters "L" and "R", meaning that after putting the  $i$ -th weight (not  $A_i$ , but  $i$ -th weight of your choice) left or right side of the balance should be heavier. Find the order of putting the weights on the balance such that rules of string  $S$  are satisfied.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
```

```

6     cin.tie(nullptr);
7     int N;
8     cin >> N;
9     vector<int> A(N);
10    for (int i = 0; i < N; i++) {
11        cin >> A[i];
12    }
13    sort(A.begin(), A.end());
14    string S;
15    cin >> S;
16    vector<pair<int, char>> ans;
17    ans.reserve(N);
18    int l = 0, r = N - 1;
19    for (int i = N - 2; i >= 0; i--) {
20        if ((S[i] == S[N - 1]) == ((N - 1 - r) % 2 == 0)) {
21            ans.emplace_back(A[l], S[N - 1] ^ ((N - 1 - l) % 2 ? 'L' ^ 'R' : 0));
22            l++;
23        } else {
24            ans.emplace_back(A[r], S[N - 1] ^ ((N - 1 - r) % 2 ? 'L' ^ 'R' : 0));
25            r--;
26        }
27    }
28    ans.emplace_back(A[l], S[N - 1] ^ ((N - 1 - l) % 2 ? 'L' ^ 'R' : 0));
29    reverse(ans.begin(), ans.end());
30    for (auto [a, b] : ans) {
31        cout << a << " " << b << "\n";
32    }
33    return 0;
34 }

```

## 770: BerDonalds

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

BerDonalds, a well-known fast food restaurant, is going to open a cafe in Bertown. The important thing is to choose the new restaurant's location so that it would be easy to get there. The Bertown road system is represented by  $n$  junctions, connected by  $m$  bidirectional roads. For each road we know its length. We also know that we can get from any junction to any other one, moving along the roads.

Your task is to find such location of the restaurant, that the shortest distance along the roads from the cafe to the farthest junction would be minimum. Note that the restaurant can be located not only on the junction, but at any point of any road.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
```

```

2  using namespace std;
3  using i64 = long long;
4  constexpr int inf = 1E9;
5  int main() {
6      ios::sync_with_stdio(false);
7      cin.tie(nullptr);
8      int n, m;
9      cin >> n >> m;
10     vector dis(n, vector(n, inf));
11     for (int i = 0; i < n; i++) {
12         dis[i][i] = 0;
13     }
14     for (int i = 0; i < m; i++) {
15         int a, b, w;
16         cin >> a >> b >> w;
17         a--;
18         b--;
19         dis[a][b] = dis[b][a] = w;
20     }
21     for (int k = 0; k < n; k++) {
22         for (int i = 0; i < n; i++) {
23             for (int j = 0; j < n; j++) {
24                 dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
25             }
26         }
27     }
28     int ans = 1E9;
29     for (int i = 0; i < n; i++) {
30         for (int j = 0; j < n; j++) {
31             if (i == j) {
32                 continue;
33             }
34             multiset<int> sa{0}, sb{0};
35             for (int k = 0; k < n; k++) {
36                 sa.insert(dis[i][k]);
37             }
38             vector<int> p(n);
39             iota(p.begin(), p.end(), 0);
40             sort(p.begin(), p.end(),
41                   [&](int x, int y) {
42                     return dis[j][x] - dis[i][x] < dis[j][y] - dis[i][y];
43                 });
44             for (auto x : p) {
45                 sa.extract(dis[i][x]);
46                 sb.insert(dis[j][x]);
47                 int A = *sa.rbegin();
48                 int B = *sb.rbegin();
49                 ans = min(ans, max({A + B + dis[i][j], 2 * A, 2 * B}));
50             }
51         }
52         cout << fixed << setprecision(2);
53         cout << .5 * ans << "\n";
54     }
55 }

```

## 771: Opening Portals

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

Pavel plays a famous computer game. A player is responsible for a whole country and he can travel there freely, complete quests and earn experience.

This country has  $n$  cities connected by  $m$  bidirectional roads of different lengths so that it is possible to get from any city to any other one. There are portals in  $k$  of these cities. At the beginning of the game all portals are closed. When a player visits a portal city, the portal opens. Strange as it is, one can teleport from an open portal to an open one. The teleportation takes no time and that enables the player to travel quickly between rather remote regions of the country.

At the beginning of the game Pavel is in city number 1. He wants to open all portals as quickly as possible. How much time will he need for that?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, m;
9     cin >> n >> m;
10    vector<vector<pair<int, int>>> adj(n);
11    for (int i = 0; i < m; i++) {
12        int x, y, w;
13        cin >> x >> y >> w;
14        x--;
15        y--;
16        adj[x].emplace_back(y, w);
17        adj[y].emplace_back(x, w);
18    }
19    vector<i64> dis(n, -1);
20    vector<int> s(n, -1);
21    priority_queue<tuple<i64, int, int>, vector<tuple<i64, int, int>>, greater<>> h;
22    int k;
23    cin >> k;
24    vector<int> p(k);
25    for (int i = 0; i < k; i++) {
26        cin >> p[i];
27        p[i]--;
28        h.emplace(0LL, p[i], p[i]);
29    }
30    while (!h.empty()) {
31        auto [d, x, l] = h.top();
32        h.pop();
33        if (dis[x] != -1) {
34            continue;
35        }
36        dis[x] = d;
37        s[x] = l;
38        for (auto [y, w] : adj[x]) {
39            h.emplace(d + w, y, s[x]);
40        }
41    }
42}
```

```

39         }
40     }
41     i64 ans = dis[0];
42     DSU dsu(n);
43     vector<tuple<i64, int, int>> e;
44     for (int i = 0; i < n; i++) {
45         for (auto [j, w] : adj[i]) {
46             if (s[i] != s[j]) {
47                 e.emplace_back(dis[i] + dis[j] + w, s[i], s[j]);
48             }
49         }
50     }
51     sort(e.begin(), e.end());
52     for (auto [w, x, y] : e) {
53         if (dsu.merge(x, y)) {
54             ans += w;
55         }
56     }
57     cout << ans << "\n";
58     return 0;
59 }
```

## 772: Number Challenge

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Let's denote  $d(n)$  as the number of divisors of a positive integer  $n$ . You are given three integers  $a$ ,  $b$  and  $c$ . Your task is to calculate the following sum:

Find the sum modulo 1073741824 (230).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int a, b, c;
8     cin >> a >> b >> c;
9     int N = max({a, b, c});
10    vector<int> mu(N + 1);
11    mu[1] = 1;
12    for (int i = 1; i <= N; i++) {
13        for (int j = i + i; j <= N; j += i) {
14            mu[j] -= mu[i];
15        }
16    }
}
```

```

17     int ans = 0;
18     for (int i = 1; i <= a; i++) {
19         for (int d = 1; d <= N; d++) {
20             if (gcd(i, d) != 1) {
21                 continue;
22             }
23             int b1 = b / d;
24             int c1 = c / d;
25             int s1 = 0, s2 = 0;
26             for (int j = 1; j <= b1; j++) {
27                 if (gcd(i, j) == 1) {
28                     s1 += b1 / j;
29                 }
30             }
31             for (int k = 1; k <= c1; k++) {
32                 if (gcd(i, k) == 1) {
33                     s2 += c1 / k;
34                 }
35             }
36             ans += (a / i) * s1 * s2 * mu[d];
37         }
38     }
39     ans &= (1 << 30) - 1;
40     cout << ans << "\n";
41     return 0;
42 }
```

### 773: Similar Polynomials

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A polynomial  $A(x)$  of degree  $d$  is an expression of the form  $A(x) = a_0 + a_1x + a_2x^2 + \dots + a_dx^d$ , where  $a_i$  are integers, and  $a_d \neq 0$ . Two polynomials  $A(x)$  and  $B(x)$  are called similar if there is an integer  $s$  such that for any integer  $x$  it holds that

$$B(x) \equiv A(x + s) \pmod{10^9 + 7}.$$

For two similar polynomials  $A(x)$  and  $B(x)$  of degree  $d$ , you're given their values in the points  $x = 0, 1, \dots, d$  modulo  $10^9 + 7$ .

Find a value  $s$  such that  $B(x) \equiv A(x + s) \pmod{10^9 + 7}$  for all integers  $x$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
```

```

4  template<class T>
5  constexpr T power(T a, i64 b) {
6      T res = 1;
7      for (; b; b /= 2, a *= a) {
8          if (b % 2) {
9              res *= a;
10         }
11     }
12     return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int d;
39     cin >> d;
40     vector<Z> a(d + 1), b(d + 1);
41     for (int i = 0; i <= d; i++) {
42         cin >> a[i];
43     }
44     for (int i = 0; i <= d; i++) {
45         cin >> b[i];
46     }
47     auto get = [&](auto a) {
48         Z v0 = 0, v1 = 0;
49         Z sum = 1LL * d * (d + 1) / 2;
50         for (int i = 0; i <= d; i++) {
51             Z c = comb.invfac(i) * comb.invfac(d - i) * ((d - i) % 2 ? -1 : 1);
52             v0 += c * a[i];
53             v1 += c * a[i] * -(sum - i);
54         }
55         return v1 / v0 / d;
56     };
57     auto x = get(a), y = get(b);
58     auto ans = y - x;
59     cout << ans << "\n";
60     return 0;
61 }

```

## 774: Random Walk

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a tree consisting of  $n$  vertices and  $n - 1$  edges, and each vertex  $v$  has a counter  $c(v)$  assigned to it.

Initially, there is a chip placed at vertex  $s$  and all counters, except  $c(s)$ , are set to 0;  $c(s)$  is set to 1.

Your goal is to place the chip at vertex  $t$ . You can achieve it by a series of moves. Suppose right now the chip is placed at the vertex  $v$ . In one move, you do the following:

choose one of neighbors  $to$  of vertex  $v$  uniformly at random ( $to$  is neighbor of  $v$  if and only if there is an edge  $\{v, to\}$  in the tree);

move the chip to vertex  $to$  and increase  $c(to)$  by 1;

You'll repeat the move above until you reach the vertex  $t$ .

For each vertex  $v$  calculate the expected value of  $c(v)$  modulo 998 244 353.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>

```

```
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int n, s, t;
38     cin >> n >> s >> t;
39     s--, t--;
40     vector<vector<int>> adj(n);
41     for (int i = 1; i < n; i++) {
42         int x, y;
43         cin >> x >> y;
44         x--, y--;
45         adj[x].push_back(y);
46         adj[y].push_back(x);
47     }
48     vector<Z> k(n);
49     function<void(int, int)> dfs = [&](int x, int p) {
50         for (auto y : adj[x]) {
51             if (y == p) {
52                 continue;
53             }
54             dfs(y, x);
55         }
56         if (p != t) {
57             k[x] = 1;
58             for (auto y : adj[x]) {
59                 if (y == p || y == t) {
60                     continue;
61                 }
62                 k[x] -= k[y] / adj[y].size();
63             }
64             if (x == s) {
65                 k[x] = 1 / k[x];
66             } else {
67                 k[x] = Z(1) / adj[p].size() / k[x];
68             }
69         }
70     };
71     dfs(s, -1);
72     dfs = [&](int x, int p) {
73         for (auto y : adj[x]) {
74             if (y == p) {
75                 continue;
76             }
77             k[y] *= k[x];
78             dfs(y, x);
79         }
80     };
81     dfs(s, -1);
82     for (int i = 0; i < n; i++) {
83         cout << k[i] << " \n"[i == n - 1];
84     }
85     return 0;
86 }
```

## 775: Emperor's Problem

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

It happened at the times of the Great Berland Empire. Once the Emperor dreamt that the Messenger from the gods ordered to build a temple whose base would be a convex polygon with  $n$  angles. Next morning the Emperor gave the command to build a temple whose base was a regular polygon with  $n$  angles. The temple was built but soon the Empire was shaken with disasters and crop failures. After an earthquake destroyed the temple, the Emperor understood that he somehow caused the wrath of gods to fall on his people. He ordered to bring the wise man. When the wise man appeared, the Emperor retold the dream to him and asked "Oh the wisest among the wisest, tell me how could I have infuriated the Gods?". "My Lord," the wise man answered. "As far as I can judge, the gods are angry because you were too hasty to fulfill their order and didn't listen to the end of the message".

Indeed, on the following night the Messenger appeared again. He reproached the Emperor for having chosen an imperfect shape for the temple. "But what shape can be more perfect than a regular polygon!?" cried the Emperor in his dream. To that the Messenger gave a complete and thorough reply.

All the vertices of the polygon should be positioned in the lattice points.

All the lengths of its sides should be different.

From the possible range of such polygons a polygon which maximum side is minimal possible must be chosen.

You are an obedient architect who is going to make the temple's plan. Note that the polygon should be simple (having a border without self-intersections and overlapping) and convex, however, it is acceptable for three consecutive vertices to lie on the same line.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 # struct Point;
6 template<class T>
7 T dot(Point<T> a, Point<T> b) {
8     return a.x * b.x + a.y * b.y;
9 }
```

```

10 template<class T>
11 T cross(Point<T> a, Point<T> b) {
12     return a.x * b.y - a.y * b.x;
13 }
14 template<class T>
15 T square(Point<T> p) {
16     return dot(p, p);
17 }
18 template<class T>
19 double length(Point<T> p) {
20     return sqrt(double(square(p)));
21 }
22 long double length(Point<long double> p) {
23     return sqrt(square(p));
24 }
25 template<class T>
26 # struct Line;
27 template<class T>
28 Point<T> rotate(Point<T> a) {
29     return Point(-a.y, a.x);
30 }
31 template<class T>
32 int sgn(Point<T> a) {
33     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
34 }
35 template<class T>
36 bool pointOnLineLeft(Point<T> p, Line<T> l) {
37     return cross(l.b - l.a, p - l.a) > 0;
38 }
39 template<class T>
40 Point<T> lineIntersection(Line<T> l1, Line<T> l2) {
41     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1
42         .a - l1.b));
43 }
44 template<class T>
45 bool pointOnSegment(Point<T> p, Line<T> l) {
46     return cross(p - l.a, l.b - l.a) == 0 && min(l.a.x, l.b.x) <= p.x && p.x <= max(l.a.x,
47         l.b.x)
48         && min(l.a.y, l.b.y) <= p.y && p.y <= max(l.a.y, l.b.y);
49 }
50 template<class T>
51 bool pointInPolygon(Point<T> a, vector<Point<T>> p) {
52     int n = p.size();
53     for (int i = 0; i < n; i++) {
54         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
55             return true;
56         }
57         int t = 0;
58         for (int i = 0; i < n; i++) {
59             auto u = p[i];
60             auto v = p[(i + 1) % n];
61             if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
62                 t ^= 1;
63             }
64             if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
65                 t ^= 1;
66             }
67         }
68     }
69     return t == 1;
70 }
71 // 0 : not intersect
72 // 1 : strictly intersect
73 // 2 : overlap

```

```

72 // 3 : intersect at endpoint
73 template<class T>
74 tuple<int, Point<T>, Point<T>> segmentIntersection(Line<T> l1, Line<T> l2) {
75     if (max(l1.a.x, l1.b.x) < min(l2.a.x, l2.b.x)) {
76         return {0, Point<T>(), Point<T>()};
77     }
78     if (min(l1.a.x, l1.b.x) > max(l2.a.x, l2.b.x)) {
79         return {0, Point<T>(), Point<T>()};
80     }
81     if (max(l1.a.y, l1.b.y) < min(l2.a.y, l2.b.y)) {
82         return {0, Point<T>(), Point<T>()};
83     }
84     if (min(l1.a.y, l1.b.y) > max(l2.a.y, l2.b.y)) {
85         return {0, Point<T>(), Point<T>()};
86     }
87     if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
88         if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
89             return {0, Point<T>(), Point<T>()};
90         } else {
91             auto maxx1 = max(l1.a.x, l1.b.x);
92             auto minx1 = min(l1.a.x, l1.b.x);
93             auto maxy1 = max(l1.a.y, l1.b.y);
94             auto miny1 = min(l1.a.y, l1.b.y);
95             auto maxx2 = max(l2.a.x, l2.b.x);
96             auto minx2 = min(l2.a.x, l2.b.x);
97             auto maxy2 = max(l2.a.y, l2.b.y);
98             auto miny2 = min(l2.a.y, l2.b.y);
99             Point<T> p1(max(minx1, minx2), max(miny1, miny2));
100            Point<T> p2(min(maxx1, maxx2), min(maxy1, maxy2));
101            if (!pointOnSegment(p1, l1)) {
102                swap(p1.y, p2.y);
103            }
104            if (p1 == p2) {
105                return {3, p1, p2};
106            } else {
107                return {2, p1, p2};
108            }
109        }
110    }
111    auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
112    auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
113    auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
114    auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
115    if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) || (cp3 < 0
116        && cp4 < 0)) {
117        return {0, Point<T>(), Point<T>()};
118    }
119    Point p = lineIntersection(l1, l2);
120    if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
121        return {1, p, p};
122    } else {
123        return {3, p, p};
124    }
125 template<class T>
126 bool segmentInPolygon(Line<T> l, vector<Point<T>> p) {
127     int n = p.size();
128     if (!pointInPolygon(l.a, p)) {
129         return false;
130     }
131     if (!pointInPolygon(l.b, p)) {
132         return false;
133     }
134     for (int i = 0; i < n; i++) {

```

```
135     auto u = p[i];
136     auto v = p[(i + 1) % n];
137     auto w = p[(i + 2) % n];
138     auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
139     if (t == 1) {
140         return false;
141     }
142     if (t == 0) {
143         continue;
144     }
145     if (t == 2) {
146         if (pointOnSegment(v, l) && v != l.a && v != l.b) {
147             if (cross(v - u, w - v) > 0) {
148                 return false;
149             }
150         }
151     } else {
152         if (p1 != u && p1 != v) {
153             if (pointOnLineLeft(l.a, Line(v, u))
154                 || pointOnLineLeft(l.b, Line(v, u))) {
155                 return false;
156             }
157         } else if (p1 == v) {
158             if (l.a == v) {
159                 if (pointOnLineLeft(u, l)) {
160                     if (pointOnLineLeft(w, l)
161                         && pointOnLineLeft(w, Line(u, v))) {
162                         return false;
163                     }
164                 } else {
165                     if (pointOnLineLeft(w, l)
166                         || pointOnLineLeft(w, Line(u, v))) {
167                         return false;
168                     }
169                 }
170             } else if (l.b == v) {
171                 if (pointOnLineLeft(u, Line(l.b, l.a))) {
172                     if (pointOnLineLeft(w, Line(l.b, l.a))
173                         && pointOnLineLeft(w, Line(u, v))) {
174                         return false;
175                     }
176                 } else {
177                     if (pointOnLineLeft(w, Line(l.b, l.a))
178                         || pointOnLineLeft(w, Line(u, v))) {
179                         return false;
180                     }
181                 }
182             } else {
183                 if (pointOnLineLeft(u, l)) {
184                     if (pointOnLineLeft(w, Line(l.b, l.a))
185                         || pointOnLineLeft(w, Line(u, v))) {
186                         return false;
187                     }
188                 } else {
189                     if (pointOnLineLeft(w, l)
190                         || pointOnLineLeft(w, Line(u, v))) {
191                         return false;
192                     }
193                 }
194             }
195         }
196     }
197 }
198 return true;
```

```
199     }
200 template<class T>
201 vector<Point<T>> hp(vector<Line<T>> lines) {
202     sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
203         auto d1 = l1.b - l1.a;
204         auto d2 = l2.b - l2.a;
205         if (sgn(d1) != sgn(d2)) {
206             return sgn(d1) == 1;
207         }
208         return cross(d1, d2) > 0;
209     });
210     deque<Line<T>> ls;
211     deque<Point<T>> ps;
212     for (auto l : lines) {
213         if (ls.empty()) {
214             ls.push_back(l);
215             continue;
216         }
217         while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
218             ps.pop_back();
219             ls.pop_back();
220         }
221         while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
222             ps.pop_front();
223             ls.pop_front();
224         }
225         if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
226             if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
227                 if (!pointOnLineLeft(ls.back().a, l)) {
228                     assert(ls.size() == 1);
229                     ls[0] = l;
230                 }
231                 continue;
232             }
233             return {};
234         }
235         ps.push_back(lineIntersection(ls.back(), l));
236         ls.push_back(l);
237     }
238     while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
239         ps.pop_back();
240         ls.pop_back();
241     }
242     if (ls.size() <= 2) {
243         return {};
244     }
245     ps.push_back(lineIntersection(ls[0], ls.back()));
246     return vector(ps.begin(), ps.end());
247 }
248 using P = Point<i64>;
249 int main() {
250     ios::sync_with_stdio(false);
251     cin.tie(nullptr);
252     int n;
253     cin >> n;
254     vector<pair<int, int>> a;
255     int s = 0;
256     for (int v = 1; a.size() <= n && (a.size() < n || s % 2 == 1); v++) {
257         for (int x = 0; x * x <= v; x++) {
258             int y = sqrt(v - x * x);
259             if (x * x + y * y == v) {
260                 a.emplace_back(x, y);
261                 s += v;
262                 break;
263             }
264         }
265     }
266 }
```

```

263         }
264     }
265 }
266 if (a.size() > n) {
267     for (int i = n; ; i--) {
268         if ((a[i].first ^ a[i].second) % 2 == s % 2) {
269             a.erase(a.begin() + i);
270             break;
271         }
272     }
273 }
274 assert(a.size() == n);
275 vector<P> p;
276 P sum(0, 0);
277 reverse(a.begin(), a.end());
278 for (auto [x, y] : a) {
279     i64 v = 1E18;
280     P q;
281     for (auto f : {P(x, y), P(y, x), P(x, -y), P(y, -x), P(-x, y), P(-y, x), P(-x, -y),
282                     , P(-y, -x)}) {
283         if (square(sum + f) < v) {
284             v = square(sum + f);
285             q = f;
286         }
287     }
288     sum += q;
289     p.push_back(q);
290 }
291 sort(p.begin(), p.end(), [&](auto a, auto b) {
292     if (sgn(a) != sgn(b)) {
293         return sgn(a) == 1;
294     } else {
295         return cross(a, b) > 0;
296     }
297 });
298 assert(sum == P(0, 0));
299 sum = P(0, 0);
300 cout << "YES\n";
301 for (int i = 0; i < n; i++) {
302     cout << sum.x << " " << sum.y << "\n";
303     sum += p[i];
304 }
305 }

```

## 776: Number Table

- Time limit: 2 seconds
- Memory limit: 216 megabytes
- Input file: standard input
- Output file: standard output

As it has been found out recently, all the Berland's current economical state can be described using a simple table  $n \times m$  in size.  $n$  - the number of days in each Berland month,  $m$  - the number of months. Thus, a table cell corresponds to a day and a month of the Berland's year. Each cell will contain either 1, or -1, which means the state's gains in a particular month, on a particular day. 1 corresponds to profits,

-1 corresponds to losses. It turned out important for successful development to analyze the data on the state of the economy of the previous year, however when the treasurers referred to the archives to retrieve the data, it turned out that the table had been substantially damaged. In some table cells the number values had faded and were impossible to be deciphered. It is known that the number of cells in which the data had been preserved is strictly less than  $\max(n, m)$ . However, there is additional information - the product of the numbers in each line and column equaled -1. Your task is to find out how many different tables may conform to the preserved data. As the answer to the task can be quite large, you have to find it modulo p.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 using Z = MInt<0>;
33 int main() {
34     ios::sync_with_stdio(false);
35     cin.tie(nullptr);
36     int n, m, k;
37     cin >> n >> m >> k;
38     if ((n + m) % 2 == 1) {
39         cout << 0 << "\n";
40         return 0;
41     }
42     vector<int> r(n, 1), c(m, 1), cr(n), cc(m);
43     for (int i = 0; i < k; i++) {
44         int x, y, z;
45         cin >> x >> y >> z;
46         x--, y--;

```

```

47         r[x] *= z;
48         c[y] *= z;
49         cr[x]++;
50         cc[y]++;
51     }
52     int p;
53     cin >> p;
54     Z::setMod(p);
55     for (int i = 0; i < n; i++) {
56         if (r[i] == 1 && cr[i] == m) {
57             cout << 0 << "\n";
58             return 0;
59         }
60     }
61     for (int i = 0; i < m; i++) {
62         if (c[i] == 1 && cc[i] == n) {
63             cout << 0 << "\n";
64             return 0;
65         }
66     }
67     int f = (n-1) * (m-1) - k;
68     for (int i = 0; i < n; i++) {
69         if (cr[i] == m) {
70             f++;
71         }
72     }
73     for (int i = 0; i < m; i++) {
74         if (cc[i] == n) {
75             f++;
76         }
77     }
78     auto ans = power(Z(2), f);
79     cout << ans << "\n";
80     return 0;
81 }
```

## 777: Interesting Sequence

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Berland scientists noticed long ago that the world around them depends on Berland population. Due to persistent research in this area the scientists managed to find out that the Berland chronology starts from the moment when the first two people came to that land (it is considered to have happened in the first year). After one Berland year after the start of the chronology the population had already equaled 13 people (the second year). However, tracing the population number during the following years was an ultimately difficult task, still it was found out that if  $d_i$  - the number of people in Berland in the year of  $i$ , then either  $d_i = 12d_{i-1}$ , or  $d_i = 13d_{i-1} - 1$ . Of course no one knows how many people are living in Berland at the moment, but now we can tell if there could possibly be a year in which the country population equaled  $A$ . That's what we ask you to determine. Also, if possible, you have to find

out in which years it could be (from the beginning of Berland chronology). Let's suppose that it could be in the years of  $a_1, a_2, \dots, a_k$ . Then you have to define how many residents could be in the country during those years apart from the A variant. Look at the examples for further explanation.

Problem: [link](#)

Solution: [link](#)

```

1  A = int(input())
2  pw = 0
3  while A % 12 == 0 :
4      A /= 12
5      pw += 1
6  x, y = 2, 13
7  p = 1
8  pos = -1
9  while x <= A :
10     if x == A :
11         pos = p
12     x, y = y, 13 * y - 12 * x
13     p += 1
14 if pos == -1 :
15     print('NO')
16     exit()
17 print('YES')
18 print(1)
19 print(pos + 2 * pw)
20 x, y = 2, 13
21 p = 1
22 oth = []
23 while p <= pos + 2 * pw :
24     if (pos - p) % 2 == 0 and pos != p :
25         oth.append(x * 12 ** ((pos + 2 * pw - p) // 2))
26     x, y = y, 13 * y - 12 * x
27     p += 1
28 print(len(oth))
29 print(' '.join(map(str, oth)))

```

## 778: The Great Marathon

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

On the Berland Dependence Day it was decided to organize a great marathon. Berland consists of  $n$  cities, some of which are linked by two-way roads. Each road has a certain length. The cities are numbered from 1 to  $n$ . It is known that one can get from any city to any other one by the roads.

$n$  runners take part in the competition, one from each city. But Berland runners are talkative by nature and that's why the juries took measures to avoid large crowds of marathon participants. The jury decided that every runner should start the marathon from their hometown. Before the start every

sportsman will get a piece of paper containing the name of the city where the sportsman's finishing line is. The finish is chosen randomly for every sportsman but it can't coincide with the sportsman's starting point. Several sportsmen are allowed to finish in one and the same city. All the sportsmen start simultaneously and everyone runs the shortest route from the starting point to the finishing one. All the sportsmen run at one speed which equals to 1.

After the competition a follow-up table of the results will be composed where the sportsmen will be sorted according to the nondecrease of time they spent to cover the distance. The first  $g$  sportsmen in the table will get golden medals, the next  $s$  sportsmen will get silver medals and the rest will get bronze medals. Besides, if two or more sportsmen spend the same amount of time to cover the distance, they are sorted according to the number of the city where a sportsman started to run in the ascending order. That means no two sportsmen share one and the same place.

According to the rules of the competition the number of gold medals  $g$  must satisfy the inequation  $g_1 \leq g \leq g_2$ , where  $g_1$  and  $g_2$  are values formed historically. In a similar way, the number of silver medals  $s$  must satisfy the inequation  $s_1 \leq s \leq s_2$ , where  $s_1$  and  $s_2$  are also values formed historically.

At present, before the start of the competition, the destination points of every sportsman are unknown. However, the press demands details and that's why you are given the task of counting the number of the ways to distribute the medals. Two ways to distribute the medals are considered different if at least one sportsman could have received during those distributions different kinds of medals.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<vector<int>> dis(n, vector<int>(n, 1E9));
10    for (int i = 0; i < n; i++) {
11        dis[i][i] = 0;
12    }
13    for (int i = 0; i < m; i++) {
14        int a, b, c;
15        cin >> a >> b >> c;
16        a--, b--;
17        dis[a][b] = dis[b][a] = c;
18    }
19    for (int k = 0; k < n; k++) {
20        for (int i = 0; i < n; i++) {
21            for (int j = 0; j < n; j++) {
22                dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
23            }
24        }
25    }
26    int g1, g2, s1, s2;
27    cin >> g1 >> g2 >> s1 >> s2;

```

```
28     for (int i = 0; i < n; i++) {
29         sort(dis[i].begin(), dis[i].end());
30     }
31     vector<pair<int, int>> a, b;
32     for (int i = 0; i < n; i++) {
33         a.emplace_back(dis[i][1], i);
34         b.emplace_back(dis[i][n - 1], i);
35     }
36     sort(a.begin(), a.end());
37     sort(b.begin(), b.end());
38     i64 ans = 0;
39     for (auto l : a) {
40         for (auto r : b) {
41             if (r < l) {
42                 continue;
43             }
44             if (l.second == r.second) {
45                 continue;
46             }
47             vector<vector<i64>> dp(g2 + 1, vector<i64>(s2 + 1));
48             dp[0][0] = 1;
49             for (int i = 0; i < n; i++) {
50                 bool x = false, y = false, z = false;
51                 if (i == l.second) {
52                     x = true;
53                 } else if (i == r.second) {
54                     z = true;
55                 } else {
56                     for (auto d : dis[i]) {
57                         if (!d) {
58                             continue;
59                         }
60                         if (pair(d, i) < l) {
61                             x = true;
62                         } else if (pair(d, i) > r) {
63                             z = true;
64                         } else {
65                             y = true;
66                         }
67                     }
68                 }
69                 for (int j = g2; j >= 0; j--) {
70                     for (int k = s2; k >= 0; k--) {
71                         if (j < g2 && x) {
72                             dp[j + 1][k] += dp[j][k];
73                         }
74                         if (k < s2 && y) {
75                             dp[j][k + 1] += dp[j][k];
76                         }
77                         if (!z) {
78                             dp[j][k] = 0;
79                         }
80                     }
81                 }
82             }
83             i64 res = 0;
84             for (int i = g1; i <= g2; i++) {
85                 for (int j = s1; j <= s2; j++) {
86                     res += dp[i][j];
87                 }
88             }
89             ans += res;
90         }
91     }
```

```

92     cout << ans << "\n";
93     return 0;
94 }
```

## 779: Timber

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There is a beautiful alley with trees in front of a shopping mall. Unfortunately, it has to go to make space for the parking lot.

The trees on the alley all grow in a single line. There are  $n$  spots for trees, index 0 is the shopping mall, index  $n + 1$  is the road and indices from 1 to  $n$  are the spots for trees. Some of them are taken - there grow trees of the same height  $k$ . No more than one tree grows in each spot.

When you chop down a tree in the spot  $x$ , you can make it fall either left or right. If it falls to the left, it takes up spots from  $x - k$  to  $x$ , inclusive. If it falls to the right, it takes up spots from  $x$  to  $x + k$ , inclusive.

Let  $m$  trees on the alley grow in some spots  $x_1, x_2, \dots, x_m$ . Let an alley be called unfortunate if all  $m$  trees can be chopped down in such a way that:

no tree falls on the shopping mall or the road;

each spot is taken up by no more than one fallen tree.

Calculate the number of different unfortunate alleys with  $m$  trees of height  $k$ . Two alleys are considered different if there is a spot  $y$  such that a tree grows in  $y$  on the first alley and doesn't grow in  $y$  on the second alley.

Output the number modulo 998 244 353.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
```

```

9         res *= a;
10    }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int n, m, k;
39     cin >> n >> m >> k;
40     k++;
41     Z ans = 0;
42     vector<Z> pw(m + 1);
43     pw[0] = 1;
44     for (int i = 1; i <= m; i++) {
45         pw[i] = pw[i - 1] * 2;
46     }
47     for (int i = 0; i <= m; i++) {
48         i64 t = 1LL * i * k + 1LL * (m - i) * (2 * k - 1);
49         if (t <= n) {
50             ans += comb.binom(n - t + m, m) * pw[i] * ((m - i) % 2 ? -1 : 1) * comb.binom(
51                         m, i);
52         }
53     }
54     cout << ans << "\n";
55 }

```

## 780: Testing

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

You take part in the testing of new weapon. For the testing a polygon was created. The polygon is a rectangular field  $n \times m$  in size, divided into unit squares  $1 \times 1$  in size. The polygon contains  $k$  objects, each

of which is a rectangle with sides, parallel to the polygon sides and entirely occupying several unit squares. The objects don't intersect and don't touch each other.

The principle according to which the weapon works is highly secret. You only know that one can use it to strike any rectangular area whose area is not equal to zero with sides, parallel to the sides of the polygon. The area must completely cover some of the unit squares into which the polygon is divided and it must not touch the other squares. Of course the area mustn't cross the polygon border. Your task is as follows: you should hit no less than one and no more than three rectangular objects. Each object must either lay completely inside the area (in that case it is considered to be hit), or lay completely outside the area.

Find the number of ways of hitting.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int dx[] = {-1, 1, 0, 0};
5 constexpr int dy[] = {0, 0, -1, 1};
6 int main() {
7     ios::sync_with_stdio(false);
8     cin.tie(nullptr);
9     int n, m, k;
10    cin >> n >> m >> k;
11    vector<string> s(n);
12    for (int i = 0; i < n; i++) {
13        cin >> s[i];
14    }
15    vector<vector<bool>> vis(n, vector<bool>(m));
16    vector<int> vx{0, n + 1}, vy{0, m};
17    vector<array<int, 4>> rec;
18    for (int i = 0; i < n; i++) {
19        for (int j = 0; j < m; j++) {
20            if (s[i][j] == '.' || vis[i][j]) {
21                continue;
22            }
23            queue<pair<int, int>> q;
24            q.emplace(i, j);
25            vis[i][j] = true;
26            int L = j, R = j, U = i, D = i;
27            while (!q.empty()) {
28                auto [x, y] = q.front();
29                q.pop();
30                L = min(L, y);
31                R = max(R, y + 1);
32                U = min(U, x);
33                D = max(D, x + 1);
34                for (int k = 0; k < 4; k++) {
35                    int nx = x + dx[k];
36                    int ny = y + dy[k];
37                    if (0 <= nx && nx < n && 0 <= ny && ny < m && s[nx][ny] == '*' && !vis
[nx][ny]) {
38                        vis[nx][ny] = true;

```

```

39                     q.emplace(nx, ny);
40                 }
41             }
42         }
43         rec.push_back({U, D, L, R});
44         if (U) {
45             vx.push_back(U - 1);
46         }
47         vx.push_back(U);
48         vx.push_back(U + 1);
49         vx.push_back(D - 1);
50         vx.push_back(D);
51         if (D < n) {
52             vx.push_back(D - 1);
53         }
54         vy.push_back(L);
55         vy.push_back(R);
56     }
57 }
58 sort(vx.begin(), vx.end());
59 sort(vy.begin(), vy.end());
60 vx.erase(unique(vx.begin(), vx.end()), vx.end());
61 vy.erase(unique(vy.begin(), vy.end()), vy.end());
62 for (auto &[U, D, L, R] : rec) {
63     U = lower_bound(vx.begin(), vx.end(), U) - vx.begin();
64     D = lower_bound(vx.begin(), vx.end(), D) - vx.begin();
65     L = lower_bound(vy.begin(), vy.end(), L) - vy.begin();
66     R = lower_bound(vy.begin(), vy.end(), R) - vy.begin();
67 }
68 i64 ans = 0;
69 for (int u = 0; u + 1 < vx.size(); u++) {
70     for (int d = u + 1; d + 1 < vx.size(); d++) {
71         vector<int> ord;
72         for (int i = 0; i < k; i++) {
73             if (rec[i][0] < d && u < rec[i][1]) {
74                 ord.push_back(i);
75             }
76         }
77         sort(ord.begin(), ord.end(), [&](int i, int j) {
78             return rec[i][2] < rec[j][2];
79         });
80         int mx = 0;
81         for (int i = 0; i < ord.size(); i++) {
82             int x = ord[i];
83             if (rec[x][2] >= mx) {
84                 int mmx = 0;
85                 for (int j = i; j <= i + 2 && j < ord.size(); j++) {
86                     int y = ord[j];
87                     auto [U, D, L, R] = rec[y];
88                     mmx = max(mmx, R);
89                     if (U < u || D > d) {
90                         break;
91                     }
92                     int lim = vy.size() - 1;
93                     if (j + 1 < ord.size()) {
94                         lim = rec[ord[j + 1]][2];
95                     }
96                     if (mmx > lim) {
97                         continue;
98                     }
99                     ans += 1LL * (vx[u + 1] - vx[u]) * (vx[d + 1] - vx[d])
100                      * (vy[rec[x][2]] - vy[mx] + 1) * (vy[lim] - vy[mmx] + 1);
101                 }
102             }
}

```

```

103             mx = max(mx, rec[x][3]);
104         }
105     }
106 }
107 cout << ans << "\n";
108 return 0;
109 }
```

## 781: Inverse Function

- Time limit: 5 seconds
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

Petya wrote a programme on C++ that calculated a very interesting function  $f(n)$ . Petya ran the program with a certain value of  $n$  and went to the kitchen to have some tea. The history has no records concerning how long the program had been working. By the time Petya returned, it had completed the calculations and had the result. However while Petya was drinking tea, a sly virus managed to destroy the input file so that Petya can't figure out for which value of  $n$  the program was run. Help Petya, carry out the inverse function!

Mostly, the program consists of a function in C++ with the following simplified syntax:

```

function ::= int f(int n) {operatorSequence}

operatorSequence ::= operator | operator operatorSequence

operator ::= return arithmExpr; | if (logicalExpr) return arithmExpr;

logicalExpr ::= arithmExpr > arithmExpr | arithmExpr < arithmExpr | arithmExpr == arithmExpr

arithmExpr ::= sum

sum ::= product | sum + product | sum - product

product ::= multiplier | product * multiplier | product / multiplier

multiplier ::= n | number | f(arithmExpr)

number ::= 0|1|2|... |32767
```

The whitespaces in a operatorSequence are optional.

Thus, we have a function, in which body there are two kinds of operators. There is the operator “return arithmExpr;” that returns the value of the expression as the value of the function, and there is the conditional operator “if (logicalExpr) return arithmExpr;” that returns the value of the arithmetical

expression when and only when the logical expression is true. Guaranteed that no other constructions of C++ language - cycles, assignment operators, nested conditional operators etc, and other variables except the n parameter are used in the function. All the constants are integers in the interval [0..32767].

The operators are performed sequentially. After the function has returned a value other operators in the sequence are not performed. Arithmetical expressions are performed taking into consideration the standard priority of the operations. It means that first all the products that are part of the sum are calculated. During the calculation of the products the operations of multiplying and division are performed from the left to the right. Then the summands are summed, and the addition and the subtraction are also performed from the left to the right. Operations “>” (more), “<” (less) and “==” (equals) also have standard meanings.

Now you've got to pay close attention! The program is compiled with the help of 15-bit Berland C++ compiler invented by a Berland company BerSoft, that's why arithmetical operations are performed in a non-standard way. Addition, subtraction and multiplication are performed modulo 32768 (if the result of subtraction is negative, then 32768 is added to it until the number belongs to the interval [0..32767]). Division “/” is a usual integer division where the remainder is omitted.

Examples of arithmetical operations:

Guaranteed that for all values of n from 0 to 32767 the given function is performed correctly. That means that:

1. Division by 0 never occurs.
2. When performing a function for the value  $n = N$  recursive calls of the function f may occur only for the parameter value of  $0, 1, \dots, N - 1$ . Consequently, the program never has an infinite recursion.
3. As the result of the sequence of the operators, the function always returns a value.

We have to mention that due to all the limitations the value returned by the function f is independent from either global variables or the order of performing the calculations of arithmetical expressions as part of the logical one, or from anything else except the value of n parameter. That's why the f function can be regarded as a function in its mathematical sense, i.e. as a unique correspondence between any value of n from the interval [0..32767] and a value of f(n) from the same interval.

Given the value of f(n), and you should find n. If the suitable n value is not unique, you should find the maximal one (from the interval [0..32767]).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 import sys
2 fn = int(input())
3 program = ''
4 s = sys.stdin.readline()
5 while s != '' :
6     program += s
7     s = sys.stdin.readline()
8 f = [0] * (2 ** 15)
9 def multiplier(s, n) :
10    global f
11    i = s.find('f')
12    if i != -1 :
13        l = i+1
14        r = len(s)-1
15        while s[l] != '(' :
16            l += 1
17        while s[r] != ')' :
18            r -= 1
19        return f[arithmExpr(s[l+1:r], n)]
20    if s.find('n') != -1 :
21        return n
22    return int(s)
23 def product(s, n) :
24    bal = 0
25    for i in range(len(s)-1,-1,-1) :
26        if s[i] == ')' :
27            bal += 1
28        elif s[i] == '(' :
29            bal -= 1
30        elif bal == 0 and s[i] == '*' :
31            return sum(s[:i], n) * product(s[i+1:], n) % 2 ** 15
32        elif bal == 0 and s[i] == '/' :
33            return sum(s[:i], n) // product(s[i+1:], n)
34    return multiplier(s, n)
35 def sum(s, n) :
36    bal = 0
37    for i in range(len(s)-1,-1,-1) :
38        if s[i] == ')' :
39            bal += 1
40        elif s[i] == '(' :
41            bal -= 1
42        elif bal == 0 and s[i] == '+' :
43            return (sum(s[:i], n) + product(s[i+1:], n)) % 2 ** 15
44        elif bal == 0 and s[i] == '-' :
45            return (sum(s[:i], n) - product(s[i+1:], n) + 2 ** 15) % 2 ** 15
46    return product(s, n)
47 def arithmExpr(s, n) :
48    return sum(s, n)
49 def logicalExpr(s, n) :
50    lt = s.find('<')
51    gt = s.find('>')
52    eq = s.find('==')
53    if lt != -1 :
54        return arithmExpr(s[:lt], n) < arithmExpr(s[lt+1:], n)
55    if gt != -1 :
56        return arithmExpr(s[:gt], n) > arithmExpr(s[gt+1:], n)
57    return arithmExpr(s[:eq], n) == arithmExpr(s[eq+2:], n)
58 def operator(s, n) :
59    i = s.find('if')
60    j = s.find('return')
61    if i != -1 :
62        l = i+2
63        r = j-1
64        while s[l] != '(' :

```

```

65         l += 1
66     while s[r] != ')' :
67         r -= 1
68     if not logicalExpr(s[l+1:r], n) :
69         return -1
70     return arithExpr(s[j+6:s.find(';')], n)
71 def operatorSequence(s, n) :
72     i = s.find(';')
73     res = operator(s[:i+1], n)
74     if res != -1 :
75         return res
76     return operatorSequence(s[i+1:], n)
77 def function(s, n) :
78     l = s.find('{')
79     r = s.find('}')
80     return operatorSequence(s[l+1:r], n)
81 for n in range(2 ** 15) :
82     f[n] = function(program, n)
83 ans = 2 ** 15 - 1
84 while ans >= 0 and f[ans] != fn :
85     ans -= 1
86 print(ans)

```

## 782: Tram

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

In a Berland city S\*\*\* there is a tram engine house and only one tram. Three people work in the house - the tram driver, the conductor and the head of the engine house. The tram used to leave the engine house every morning and drove along his loop route. The tram needed exactly  $c$  minutes to complete the route. The head of the engine house controlled the trams movement, going outside every  $c$  minutes when the tram drove by the engine house, and the head left the driver without a bonus if he was even one second late.

It used to be so. Afterwards the Berland Federal Budget gave money to make more tramlines in S, ***and, as it sometimes happens, the means were used as it was planned. The tramlines were rebuilt and as a result they turned into a huge network. The previous loop route may have been destroyed. S*** has  $n$  crossroads and now  $m$  tramlines that links the pairs of crossroads. The traffic in Berland is one way so the tram can move along each tramline only in one direction. There may be several tramlines between two crossroads, which go same way or opposite ways. Every tramline links two different crossroads and for each crossroad there is at least one outgoing tramline.

So, the tramlines were built but for some reason nobody gave a thought to increasing the number of trams in S! ***The tram continued to ride alone but now the driver had an excellent opportunity to get rid of the unending control of the engine house head. For now due to the tramline network he could***

**choose the route freely! Now at every crossroad the driver can arbitrarily choose the way he can go. The tram may even go to the parts of S** from where it cannot return due to one way traffic. The driver is not afraid of the challenge: at night, when the city is asleep, he can return to the engine house safely, driving along the tramlines in the opposite direction.

The city people were rejoicing for some of the had been waiting for the tram to appear on their streets for several years. However, the drivers behavior enraged the engine house head. Now he tries to carry out an insidious plan of installing cameras to look after the rebellious tram.

The plan goes as follows. The head of the engine house wants to install cameras at some crossroads, to choose a period of time  $t$  and every  $t$  minutes turn away from the favourite TV show to check where the tram is. Also the head of the engine house wants at all moments of time, divisible by  $t$ , and only at such moments the tram to appear on a crossroad under a camera. There must be a camera on the crossroad by the engine house to prevent possible terrorist attacks on the engine house head. Among all the possible plans the engine house head chooses the plan with the largest possible value of  $t$  (as he hates being distracted from his favourite TV show but he has to). If such a plan is not unique, pick the plan that requires the minimal possible number of cameras. Find such a plan.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<vector<int>> adj(n);
10    for (int i = 0; i < m; i++) {
11        int u, v;
12        cin >> u >> v;
13        u--, v--;
14        adj[u].push_back(v);
15    }
16    vector<int> vis(n, -1);
17    int x = 0, t = 0;
18    while (vis[x] == -1) {
19        vis[x] = t;
20        t++;
21        x = adj[x][0];
22    }
23    int d = t - vis[x];
24    for (int t = d; ; t--) {
25        if (d % t == 0) {
26            vector<int> f(n, -1);
27            queue<int> q;
28            f[0] = 0;
29            q.push(0);
30            vector<int> ans;
```

```

31         bool ok = true;
32         while (!q.empty()) {
33             int x = q.front();
34             q.pop();
35             if (f[x] == 0) {
36                 ans.push_back(x);
37             }
38             for (auto y : adj[x]) {
39                 if (f[y] == -1) {
40                     f[y] = (f[x] + 1) % t;
41                     q.push(y);
42                 } else if (f[y] != (f[x] + 1) % t) {
43                     ok = false;
44                 }
45             }
46             if (!ok) {
47                 continue;
48             }
49             cout << t << "\n";
50             cout << ans.size() << "\n";
51             sort(ans.begin(), ans.end());
52             for (auto x : ans) {
53                 cout << x + 1 << " \n"[x == ans.back()];
54             }
55         }
56         return 0;
57     }
58 }
59
60 }
```

### 783: Helper

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

It's unbelievable, but an exam period has started at the OhWord University. It's even more unbelievable, that Valera got all the tests before the exam period for excellent work during the term. As now he's free, he wants to earn money by solving problems for his groupmates. He's made a list of subjects that he can help with. Having spoken with  $n$  of his groupmates, Valera found out the following information about them: what subject each of them passes, time of the exam and sum of money that each person is ready to pay for Valera's help.

Having this data, Valera's decided to draw up a timetable, according to which he will solve problems for his groupmates. For sure, Valera can't solve problems round the clock, that's why he's found for himself an optimum order of day and plans to stick to it during the whole exam period. Valera assigned time segments for sleep, breakfast, lunch and dinner. The rest of the time he can work.

Obviously, Valera can help a student with some subject, only if this subject is on the list. It happened,

that all the students, to whom Valera spoke, have different, but one-type problems, that's why Valera can solve any problem of subject list in  $t_i$  minutes.

Moreover, if Valera starts working at some problem, he can break off only for sleep or meals, but he can't start a new problem, not having finished the current one. Having solved the problem, Valera can send it instantly to the corresponding student via the Internet.

If this student's exam hasn't started yet, he can make a crib, use it to pass the exam successfully, and pay Valera the promised sum. Since Valera has little time, he asks you to write a program that finds the order of solving problems, which can bring Valera maximum profit.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int timeToInt(string s) {
5     int hh = stoi(s.substr(0, 2));
6     int mm = stoi(s.substr(3));
7     return hh * 60 + mm;
8 }
9 string intToTime(int x) {
10    int hh = x / 60;
11    int mm = x % 60;
12    string s;
13    s += '0' + hh / 10;
14    s += '0' + hh % 10;
15    s += ':';
16    s += '0' + mm / 10;
17    s += '0' + mm % 10;
18    return s;
19 }
20 int main() {
21     ios::sync_with_stdio(false);
22     cin.tie(nullptr);
23     int m, n, k;
24     cin >> m >> n >> k;
25     vector<string> name(m);
26     map<string, int> id;
27     for (int i = 0; i < m; i++) {
28         cin >> name[i];
29         id[name[i]] = i;
30     }
31     vector<int> cost(m);
32     for (int i = 0; i < m; i++) {
33         cin >> cost[i];
34     }
35     array<bool, 1440> free;
36     free.fill(true);
37     for (int i = 0; i < 4; i++) {
38         string s;
39         cin >> s;
40         int S = timeToInt(s.substr(0, 5));
41         int T = timeToInt(s.substr(6));
42         for (int i = S; i <= T; i++) {

```

```

43         free[i] = false;
44     }
45 }
46 array<int, 1441> pre{};
47 for (int i = 0; i < 1440; i++) {
48     pre[i + 1] = pre[i] + free[i];
49 }
50 int freePerDay = pre[1440];
51 int totalFree = freePerDay * k;
52 vector<array<int, 4>> people;
53 for (int i = 0; i < n; i++) {
54     string name;
55     int day;
56     string time;
57     int money;
58     cin >> name >> day >> time >> money;
59     if (!id.count(name)) {
60         continue;
61     }
62     int t = timeToInt(time);
63     t = (day - 1) * freePerDay + pre[t];
64     people.push_back({t, cost[id[name]], money, i});
65 }
66 sort(people.begin(), people.end());
67 n = people.size();
68 vector<int> point;
69 for (int i = 0; i < k; i++) {
70     for (int j = 0; j < 1440; j++) {
71         if (free[j]) {
72             point.push_back(i * 1440 + j);
73         }
74     }
75 }
76 vector<int> dp(n + 1, vector<int>(totalFree + 1));
77 for (int i = 0; i < n; i++) {
78     auto [t, cost, money, j] = people[i];
79     dp[i + 1] = dp[i];
80     for (int j = t - cost; j >= 0; j--) {
81         dp[i + 1][j + cost] = max(dp[i + 1][j + cost], dp[i][j] + money);
82     }
83     for (int j = 1; j <= totalFree; j++) {
84         dp[i + 1][j] = max(dp[i + 1][j], dp[i + 1][j - 1]);
85     }
86 }
87 cout << dp[n][totalFree] << "\n";
88 vector<tuple<int, int, string, int, string>> ans;
89 int t = totalFree;
90 for (int i = n - 1; i >= 0; i--) {
91     auto [lim, cost, money, j] = people[i];
92     while (t && dp[i + 1][t - 1] == dp[i + 1][t]) {
93         t--;
94     }
95     if (t <= lim && t >= cost && dp[i + 1][t] == dp[i][t - cost] + money) {
96         int end = point[t - 1];
97         t -= cost;
98         int start = point[t];
99         ans.emplace_back(j + 1, start / 1440 + 1, intToTime(start % 1440),
100                         , end / 1440 + 1, intToTime(end % 1440));
101    } else {
102        assert(dp[i + 1][t] == dp[i][t]);
103    }
104 }
105 reverse(ans.begin(), ans.end());
106 cout << ans.size() << "\n";

```

```

107     for (auto [a, b, c, d, e] : ans) {
108         cout << a << " " << b << " " << c << " " << d << " " << e << "\n";
109     }
110 }
111 }
```

## 784: Hide-and-Seek

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Victor and Peter are playing hide-and-seek. Peter has hidden, and Victor is to find him. In the room where they are playing, there is only one non-transparent wall and one double-sided mirror. Victor and Peter are points with coordinates  $(x_v, y_v)$  and  $(x_p, y_p)$  respectively. The wall is a segment joining points with coordinates  $(x_w, 1, y_w, 1)$  and  $(x_w, 2, y_w, 2)$ , the mirror - a segment joining points  $(x_m, 1, y_m, 1)$  and  $(x_m, 2, y_m, 2)$ .

If an obstacle has a common point with a line of vision, it's considered, that the boys can't see each other with this line of vision. If the mirror has a common point with the line of vision, it's considered, that the boys can see each other in the mirror, i.e. reflection takes place. The reflection process is governed by laws of physics - the angle of incidence is equal to the angle of reflection. The incident ray is in the same half-plane as the reflected ray, relative to the mirror. I.e. to see each other Victor and Peter should be to the same side of the line, containing the mirror (see example 1). If the line of vision is parallel to the mirror, reflection doesn't take place, and the mirror isn't regarded as an obstacle (see example 4).

Victor got interested if he can see Peter, while standing at the same spot. Help him solve this problem.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3 template<class T>
4 # struct Frac;
5 template<class T>
6 # struct Point;
7 template<class T>
8 T dot(Point<T> a, Point<T> b) {
9     return a.x * b.x + a.y * b.y;
10 }
11 template<class T>
12 T cross(Point<T> a, Point<T> b) {
13     return a.x * b.y - a.y * b.x;
14 }
15 template<class T>
```

```

16 T square(Point<T> p) {
17     return dot(p, p);
18 }
19 template<class T>
20 double length(Point<T> p) {
21     return std::sqrt(double(square(p)));
22 }
23 long double length(Point<long double> p) {
24     return std::sqrt(square(p));
25 }
26 template<class T>
27 # struct Line;
28 template<class T>
29 Point<T> rotate(Point<T> a) {
30     return Point(-a.y, a.x);
31 }
32 template<class T>
33 int sgn(Point<T> a) {
34     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
35 }
36 template<class T>
37 bool pointOnLineLeft(Point<T> p, Line<T> l) {
38     return cross(l.b - l.a, p - l.a) > 0;
39 }
40 template<class T>
41 Point<T> lineIntersection(Line<T> l1, Line<T> l2) {
42     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1
        .a - l1.b));
43 }
44 template<class T>
45 bool pointOnSegment(Point<T> p, Line<T> l) {
46     return cross(p - l.a, l.b - l.a) == T(0) && std::min(l.a.x, l.b.x) <= p.x && p.x <=
        std::max(l.a.x, l.b.x)
        && std::min(l.a.y, l.b.y) <= p.y && p.y <= std::max(l.a.y, l.b.y);
47 }
48 template<class T>
49 bool pointInPolygon(Point<T> a, std::vector<Point<T>> p) {
50     int n = p.size();
51     for (int i = 0; i < n; i++) {
52         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
53             return true;
54         }
55     }
56     int t = 0;
57     for (int i = 0; i < n; i++) {
58         auto u = p[i];
59         auto v = p[(i + 1) % n];
60         if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
61             t ^= 1;
62         }
63         if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
64             t ^= 1;
65         }
66     }
67     return t == 1;
68 }
69 // 0 : not intersect
70 // 1 : strictly intersect
71 // 2 : overlap
72 // 3 : intersect at endpoint
73 template<class T>
74 std::tuple<int, Point<T>, Point<T>> segmentIntersection(Line<T> l1, Line<T> l2) {
75     if (std::max(l1.a.x, l1.b.x) < std::min(l2.a.x, l2.b.x)) {
76         return {0, Point<T>(), Point<T>()};
77 }

```

```

79 } if (std::min(l1.a.x, l1.b.x) > std::max(l2.a.x, l2.b.x)) {
80     return {0, Point<T>(), Point<T>()};
81 }
82 if (std::max(l1.a.y, l1.b.y) < std::min(l2.a.y, l2.b.y)) {
83     return {0, Point<T>(), Point<T>()};
84 }
85 if (std::min(l1.a.y, l1.b.y) > std::max(l2.a.y, l2.b.y)) {
86     return {0, Point<T>(), Point<T>()};
87 }
88 if (cross(l1.b - l1.a, l2.b - l2.a) == T(0)) {
89     if (cross(l1.b - l1.a, l2.a - l1.a) != T(0)) {
90         return {0, Point<T>(), Point<T>()};
91     } else {
92         auto maxx1 = std::max(l1.a.x, l1.b.x);
93         auto minx1 = std::min(l1.a.x, l1.b.x);
94         auto maxy1 = std::max(l1.a.y, l1.b.y);
95         auto miny1 = std::min(l1.a.y, l1.b.y);
96         auto maxx2 = std::max(l2.a.x, l2.b.x);
97         auto minx2 = std::min(l2.a.x, l2.b.x);
98         auto maxy2 = std::max(l2.a.y, l2.b.y);
99         auto miny2 = std::min(l2.a.y, l2.b.y);
100        Point<T> p1(std::max(minx1, minx2), std::max(miny1, miny2));
101        Point<T> p2(std::min(maxx1, maxx2), std::min(maxy1, maxy2));
102        if (!pointOnSegment(p1, l1)) {
103            std::swap(p1.y, p2.y);
104        }
105        if (p1 == p2) {
106            return {3, p1, p2};
107        } else {
108            return {2, p1, p2};
109        }
110    }
111 }
112 auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
113 auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
114 auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
115 auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
116 if ((cp1 > T(0) && cp2 > T(0)) || (cp1 < T(0) && cp2 < T(0)) || (cp3 > T(0) && cp4 > T(0)) || (cp3 < T(0) && cp4 < T(0))) {
117     return {0, Point<T>(), Point<T>()};
118 }
119 Point p = lineIntersection(l1, l2);
120 if (cp1 != T(0) && cp2 != T(0) && cp3 != T(0) && cp4 != T(0)) {
121     return {1, p, p};
122 } else {
123     return {3, p, p};
124 }
125 }
126 template<class T>
127 bool segmentInPolygon(Line<T> l, std::vector<Point<T>> p) {
128     int n = p.size();
129     if (!pointInPolygon(l.a, p)) {
130         return false;
131     }
132     if (!pointInPolygon(l.b, p)) {
133         return false;
134     }
135     for (int i = 0; i < n; i++) {
136         auto u = p[i];
137         auto v = p[(i + 1) % n];
138         auto w = p[(i + 2) % n];
139         auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
140         if (t == 1) {

```

```
141         return false;
142     }
143     if (t == 0) {
144         continue;
145     }
146     if (t == 2) {
147         if (pointOnSegment(v, l) && v != l.a && v != l.b) {
148             if (cross(v - u, w - v) > 0) {
149                 return false;
150             }
151         }
152     } else {
153         if (p1 != u && p1 != v) {
154             if (pointOnLineLeft(l.a, Line(v, u))
155                 || pointOnLineLeft(l.b, Line(v, u))) {
156                 return false;
157             }
158         } else if (p1 == v) {
159             if (l.a == v) {
160                 if (pointOnLineLeft(u, l)) {
161                     if (pointOnLineLeft(w, l)
162                         && pointOnLineLeft(w, Line(u, v))) {
163                         return false;
164                     }
165                 } else {
166                     if (pointOnLineLeft(w, l)
167                         || pointOnLineLeft(w, Line(u, v))) {
168                         return false;
169                     }
170                 }
171             } else if (l.b == v) {
172                 if (pointOnLineLeft(u, Line(l.b, l.a))) {
173                     if (pointOnLineLeft(w, Line(l.b, l.a))
174                         && pointOnLineLeft(w, Line(u, v))) {
175                         return false;
176                     }
177                 } else {
178                     if (pointOnLineLeft(w, Line(l.b, l.a))
179                         || pointOnLineLeft(w, Line(u, v))) {
180                         return false;
181                     }
182                 }
183             } else {
184                 if (pointOnLineLeft(u, l)) {
185                     if (pointOnLineLeft(w, Line(l.b, l.a))
186                         || pointOnLineLeft(w, Line(u, v))) {
187                         return false;
188                     }
189                 } else {
190                     if (pointOnLineLeft(w, l)
191                         || pointOnLineLeft(w, Line(u, v))) {
192                         return false;
193                     }
194                 }
195             }
196         }
197     }
198 }
199 return true;
200 }
201 template<class T>
202 std::vector<Point<T>> hp(std::vector<Line<T>> lines) {
203     std::sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
204         auto d1 = l1.b - l1.a;
```

```

205         auto d2 = l2.b - l2.a;
206         if (sgn(d1) != sgn(d2)) {
207             return sgn(d1) == 1;
208         }
209         return cross(d1, d2) > 0;
210     });
211     std::deque<Line<T>> ls;
212     std::deque<Point<T>> ps;
213     for (auto l : lines) {
214         if (ls.empty()) {
215             ls.push_back(l);
216             continue;
217         }
218         while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
219             ps.pop_back();
220             ls.pop_back();
221         }
222         while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
223             ps.pop_front();
224             ls.pop_front();
225         }
226         if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
227             if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
228                 if (!pointOnLineLeft(ls.back().a, l)) {
229                     assert(ls.size() == 1);
230                     ls[0] = l;
231                 }
232                 continue;
233             }
234             return {};
235         }
236         ps.push_back(lineIntersection(ls.back(), l));
237         ls.push_back(l);
238     }
239     while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
240         ps.pop_back();
241         ls.pop_back();
242     }
243     if (ls.size() <= 2) {
244         return {};
245     }
246     ps.push_back(lineIntersection(ls[0], ls.back()));
247     return std::vector(ps.begin(), ps.end());
248 }
249 /*
250  * @author panks
251  * Big Integer library in C++, single file implementation.
252  */
253 #include <iostream>
254 #include <string>
255 #include <sstream>
256 #include <cmath>
257 #define MAX 10000 // for strings
258 using namespace std;
259 class BigInteger {
260 private:
261     string number;
262     bool sign;
263 public:
264     BigInteger(); // empty constructor initializes
265     BigInteger(string s) { // "string" constructor if( else {
266         setNumber( s.substr(1) );
267         sign = (s[0] == '-');
268     }

```

```
269 }
270 BigInteger::BigInteger(string s, bool sin) { // "string" constructor setNumber()
271     void BigInteger::setNumber(string s) {
272         number = s;
273     }
274     const string& BigInteger::getNumber() { // retrieves the number
275         return number;
276     }
277     void BigInteger::setSign(bool s) {
278         sign = s;
279     }
280     const bool& BigInteger::getSign() {
281         return sign;
282     }
283     BigInteger BigInteger::absolute() {
284         return BigInteger( getNumber() ); // +ve by default
285     }
286     void BigInteger::operator = (BigInteger b) {
287         setNumber( b.getNumber() );
288         setSign( b.getSign() );
289     }
290     bool BigInteger::operator == (BigInteger b) {
291         return equals((*this) , b);
292     }
293     bool BigInteger::operator != (BigInteger b) {
294         return ! equals((*this) , b);
295     }
296     bool BigInteger::operator > (BigInteger b) {
297         return greater((*this) , b);
298     }
299     bool BigInteger::operator < (BigInteger b) {
300         return less((*this) , b);
301     }
302     bool BigInteger::operator >= (BigInteger b) {
303         return equals((*this) , b)
304             || greater((*this) , b);
305     }
306     bool BigInteger::operator <= (BigInteger b) {
307         return equals((*this) , b)
308             || less((*this) , b);
309     }
310     BigInteger& BigInteger::operator ++() { // prefix
311         (*this) = (*this) + 1;
312         return (*this);
313     }
314     BigInteger BigInteger::operator ++(int) { // postfix
315         BigInteger before = (*this);
316         (*this) = (*this) + 1;
317         return before;
318     }
319     BigInteger& BigInteger::operator --() { // prefix
320         (*this) = (*this) - 1;
321         return (*this);
322     }
323     BigInteger BigInteger::operator --(int) { // postfix
324         BigInteger before = (*this);
325         (*this) = (*this) - 1;
326         return before;
327     }
328     BigInteger BigInteger::operator + (BigInteger b) {
329         BigInteger addition;
330         if( getSign() == b.getSign() ) { // both +ve or -ve
331             addition.setNumber( add(getNumber(), b.getNumber() ) );
332             addition.setSign( getSign() );
```

```

333     } else { // sign different
334         if( absolute() > b.absolute() ) {
335             addition.setNumber( subtract(getNumber(), b.getNumber() ) );
336             addition.setSign( getSign() );
337         } else {
338             addition.setNumber( subtract(b.getNumber(), getNumber() ) );
339             addition.setSign( b.getSign() );
340         }
341     }
342     if(addition.getNumber() == "0") // avoid (-0) problem
343         addition.setSign(false);
344     return addition;
345 }
346 BigInteger BigInteger::operator - (BigInteger b) {
347     b.setSign( ! b.getSign() ); // x - y = x + (-y)
348     return (*this) + b;
349 }
350 BigInteger BigInteger::operator * (BigInteger b) {
351     BigInteger mul;
352     mul.setNumber( multiply(getNumber(), b.getNumber() ) );
353     mul.setSign( getSign() != b.getSign() );
354     if(mul.getNumber() == "0") // avoid (-0) problem
355         mul.setSign(false);
356     return mul;
357 }
358 // Warning: Denominator must be within "long long" size not "BigInteger"
359 BigInteger BigInteger::operator / (BigInteger b) {
360     long long den = toInt( b.getNumber() );
361     BigInteger div;
362     div.setNumber( divide(getNumber(), den).first );
363     div.setSign( getSign() != b.getSign() );
364     if(div.getNumber() == "0") // avoid (-0) problem
365         div.setSign(false);
366     return div;
367 }
368 // Warning: Denominator must be within "long long" size not "BigInteger"
369 BigInteger BigInteger::operator % (BigInteger b) {
370     long long den = toInt( b.getNumber() );
371     BigInteger rem;
372     long long rem_int = divide(number, den).second;
373     rem.setNumber( toString(rem_int) );
374     rem.setSign( getSign() != b.getSign() );
375     if(rem.getNumber() == "0") // avoid (-0) problem
376         rem.setSign(false);
377     return rem;
378 }
379 BigInteger& BigInteger::operator += (BigInteger b) {
380     (*this) = (*this) + b;
381     return (*this);
382 }
383 BigInteger& BigInteger::operator -= (BigInteger b) {
384     (*this) = (*this) - b;
385     return (*this);
386 }
387 BigInteger& BigInteger::operator *= (BigInteger b) {
388     (*this) = (*this) * b;
389     return (*this);
390 }
391 BigInteger& BigInteger::operator /= (BigInteger b) {
392     (*this) = (*this) / b;
393     return (*this);
394 }
395 BigInteger& BigInteger::operator %= (BigInteger b) {
396     (*this) = (*this) % b;

```

```

397     return (*this);
398 }
399 BigInteger& BigInteger::operator [] (int n) {
400     return *(this + (n*sizeof(BigInteger)));
401 }
402 BigInteger BigInteger::operator -() { // unary minus sign
403     return (*this) * -1;
404 }
405 BigInteger::operator string() { // for conversion from BigInteger to string
406     string signedString = ( getSign() ) ? "-" : "";
407     signedString += number;
408     return signedString;
409 }
410 bool BigInteger::equals(BigInteger n1, BigInteger n2) {
411     return n1.getNumber() == n2.getNumber()
412         && n1.getSign() == n2.getSign();
413 }
414 bool BigInteger::less(BigInteger n1, BigInteger n2) {
415     bool sign1 = n1.getSign();
416     bool sign2 = n2.getSign();
417     if(sign1 && !sign2) // if n1 is -ve and n2 is +ve
418         return true;
419     else if(!sign1 && sign2)
420         return false;
421     else if(!sign1) { // both +ve
422         if(n1.getNumber().length() < n2.getNumber().length())
423             return true;
424         if(n1.getNumber().length() > n2.getNumber().length())
425             return false;
426         return n1.getNumber() < n2.getNumber();
427     } else { // both -ve
428         if(n1.getNumber().length() > n2.getNumber().length())
429             return true;
430         if(n1.getNumber().length() < n2.getNumber().length())
431             return false;
432         return n1.getNumber().compare( n2.getNumber() ) > 0; // greater with -ve sign is
433             LESS
434     }
435 }
436 bool BigInteger::greater(BigInteger n1, BigInteger n2) {
437     return ! equals(n1, n2) && ! less(n1, n2);
438 }
439 string BigInteger::add(string number1, string number2) {
440     string add = (number1.length() > number2.length()) ? number1 : number2;
441     char carry = '0';
442     int differenceInLength = abs( (int) (number1.size() - number2.size() ) );
443     if(number1.size() > number2.size())
444         number2.insert(0, differenceInLength, '0'); // put zeros from left
445     else// if(number1.size() < number2.size())
446         number1.insert(0, differenceInLength, '0');
447     for(int i=number1.size()-1; i>=0; --i) {
448         add[i] = ((carry-'0')+(number1[i]-'0')+(number2[i]-'0')) + '0';
449         if(i != 0) {
450             if(add[i] > '9') {
451                 add[i] -= 10;
452                 carry = '1';
453             } else
454                 carry = '0';
455         }
456         if(add[0] > '9') {
457             add[0]-= 10;
458             add.insert(0,1,'1');
459     }

```

```

460     return add;
461 }
462 string BigInteger::subtract(string number1, string number2) {
463     string sub = (number1.length()>number2.length())? number1 : number2;
464     int differenceInLength = abs( (int)(number1.size() - number2.size()) );
465     if(number1.size() > number2.size())
466         number2.insert(0, differenceInLength, '0');
467     else
468         number1.insert(0, differenceInLength, '0');
469     for(int i=number1.length()-1; i>=0; --i) {
470         if(number1[i] < number2[i]) {
471             number1[i] += 10;
472             number1[i-1]--;
473         }
474         sub[i] = ((number1[i]-'0')-(number2[i]-'0')) + '0';
475     }
476     while(sub[0]=='0' && sub.length()!=1) // erase leading zeros
477         sub.erase(0,1);
478     return sub;
479 }
480 string BigInteger::multiply(string n1, string n2) {
481     if(n1.length() > n2.length())
482         n1.swap(n2);
483     string res = "0";
484     for(int i=n1.length()-1; i>=0; --i) {
485         string temp = n2;
486         int currentDigit = n1[i]-'0';
487         int carry = 0;
488         for(int j=temp.length()-1; j>=0; --j) {
489             temp[j] = ((temp[j]-'0') * currentDigit) + carry;
490             if(temp[j] > 9) {
491                 carry = (temp[j]/10);
492                 temp[j] -= (carry*10);
493             } else
494                 carry = 0;
495             temp[j] += '0'; // back to string mood
496         }
497         if(carry > 0)
498             temp.insert(0, 1, (carry+'0'));
499         temp.append((n1.length()-i-1), '0'); // as like mult by 10, 100, 1000, 10000 and
500         so on
501         res = add(res, temp); // O(n)
502     }
503     while(res[0] == '0' && res.length()!=1) // erase leading zeros
504         res.erase(0,1);
505     return res;
506 }
507 pair<string, long long> BigInteger::divide(string n, long long den) {
508     long long rem = 0;
509     string result;
510     result.resize(MAX);
511     for(int indx=0, len = n.length(); indx<len; ++indx) {
512         rem = (rem * 10) + (n[indx] - '0');
513         result[indx] = rem / den + '0';
514         rem %= den;
515     }
516     result.resize( n.length() );
517     while( result[0] == '0' && result.length() != 1)
518         result.erase(0,1);
519     if(result.length() == 0)
520         result = "0";
521     return make_pair(result, rem);
522 }
523 string BigInteger::toString(long long n) {

```

```

523     stringstream ss;
524     string temp;
525     ss << n;
526     ss >> temp;
527     return temp;
528 }
529 long long BigInteger::toInt(string s) {
530     long long sum = 0;
531     for(int i=0; i<s.length(); i++)
532         sum = (sum*10) + (s[i] - '0');
533     return sum;
534 }
535 using P = Point<Frac<BigInteger>>;
536 P read() {
537     int x, y;
538     std::cin >> x >> y;
539     return P(BigInteger(x), BigInteger(y));
540 }
541 int main() {
542     std::ios::sync_with_stdio(false);
543     std::cin.tie(nullptr);
544     P A = read(), B = read();
545     P C = read(), D = read();
546     P E = read(), F = read();
547     if (std::get<0>(segmentIntersection(Line(A, B), Line(C, D))) == 0
548         && std::get<0>(segmentIntersection(Line(A, B), Line(E, F))) % 2 == 0) {
549         std::cout << "YES\n";
550         return 0;
551     }
552     auto B1 = ((B - E) / (F - E)).conj() * (F - E) + E;
553     auto [t, U, V] = segmentIntersection(Line(A, B1), Line(E, F));
554     if (t != 1 && t != 3) {
555         std::cout << "NO\n";
556         return 0;
557     }
558     if (std::get<0>(segmentIntersection(Line(A, U), Line(C, D))) == 0
559         && std::get<0>(segmentIntersection(Line(B, U), Line(C, D))) == 0) {
560         std::cout << "YES\n";
561         return 0;
562     }
563     std::cout << "NO\n";
564     return 0;
565 }

```

## 785: TV Game

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There is a new TV game on BerTV. In this game two players get a number A consisting of  $2n$  digits. Before each turn players determine who will make the next move. Each player should make exactly  $n$  moves. On it's turn  $i$ -th player takes the leftmost digit of A and appends it to his or her number  $S_i$ . After that this leftmost digit is erased from A. Initially the numbers of both players ( $S_1$  and  $S_2$ ) are empty.

Leading zeroes in numbers A, S1, S2 are allowed. In the end of the game the first player gets S1 dollars, and the second gets S2 dollars.

One day Homer and Marge came to play the game. They managed to know the number A beforehand. They want to find such sequence of their moves that both of them makes exactly n moves and which maximizes their total prize. Help them.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 2E18;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n;
9     cin >> n;
10    string s;
11    cin >> s;
12    vector dp(2 * n + 1, vector(n + 1, -inf));
13    dp[2 * n][0] = 1;
14    vector<i64> pw(n);
15    pw[0] = 1;
16    for (int i = 1; i < n; i++) {
17        pw[i] = pw[i - 1] * 10;
18    }
19    for (int i = 2 * n - 1; i >= 0; i--) {
20        for (int j = 0; j <= n; j++) {
21            int k = 2 * n - 1 - i - j;
22            if (k < 0) {
23                continue;
24            }
25            if (j < n) {
26                dp[i][j + 1] = max(dp[i][j + 1], dp[i + 1][j] + (s[i] - '0') * pw[j]);
27            }
28            if (k < n) {
29                dp[i][j] = max(dp[i][j], dp[i + 1][j] + (s[i] - '0') * pw[k]);
30            }
31        }
32    }
33    string ans;
34    int j = n;
35    for (int i = 0; i < 2 * n; i++) {
36        if (j && dp[i][j] == dp[i + 1][j - 1] + (s[i] - '0') * pw[j - 1]) {
37            j--;
38            ans += 'H';
39        } else {
40            ans += 'M';
41        }
42    }
43    cout << ans << "\n";
44    return 0;
45 }
```

## 786: Quarrel

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Friends Alex and Bob live in Bertown. In this town there are  $n$  crossroads, some of them are connected by bidirectional roads of equal length. Bob lives in a house at the crossroads number 1, Alex - in a house at the crossroads number  $n$ .

One day Alex and Bob had a big quarrel, and they refused to see each other. It occurred that today Bob needs to get from his house to the crossroads  $n$  and Alex needs to get from his house to the crossroads 1. And they don't want to meet at any of the crossroads, but they can meet in the middle of the street, when passing it in opposite directions. Alex and Bob asked you, as their mutual friend, to help them with this difficult task.

Find for Alex and Bob such routes with equal number of streets that the guys can follow these routes and never appear at the same crossroads at the same time. They are allowed to meet in the middle of the street when moving toward each other (see Sample 1). Among all possible routes, select such that the number of streets in it is the least possible. Until both guys reach their destinations, none of them can stay without moving.

The guys are moving simultaneously with equal speeds, i.e. it is possible that when one of them reaches some of the crossroads, the other one leaves it. For example, Alex can move from crossroad 1 to crossroad 2, while Bob moves from crossroad 2 to crossroad 3.

If the required routes don't exist, your program should output -1.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<vector<int>> adj(n);
10    for (int i = 0; i < m; i++) {
11        int u, v;
12        cin >> u >> v;
13        u--, v--;
14        adj[u].push_back(v);
15        adj[v].push_back(u);
16    }
17    vector<int> dis(n, vector(n, -1));

```

```

18     vector<vector<pair<int, int>> lst(n, vector<n, pair{-1, -1}>());
19     queue<pair<int, int>> q;
20     q.emplace(0, n - 1);
21     dis[0][n - 1] = 0;
22     while (!q.empty()) {
23         auto [u, v] = q.front();
24         q.pop();
25         for (auto x : adj[u]) {
26             for (auto y : adj[v]) {
27                 if (x != y && dis[x][y] == -1) {
28                     dis[x][y] = dis[u][v] + 1;
29                     lst[x][y] = {u, v};
30                     q.emplace(x, y);
31                 }
32             }
33         }
34     }
35     if (dis[n - 1][0] == -1) {
36         cout << -1 << "\n";
37         return 0;
38     }
39     vector<int> a, b;
40     for (int x = n - 1, y = 0; x != -1; tie(x, y) = lst[x][y]) {
41         a.push_back(x);
42         b.push_back(y);
43     }
44     cout << a.size() - 1 << "\n";
45     for (int i = 0; i < a.size(); i++) {
46         cout << b[i] + 1 << " \n"[i == a.size() - 1];
47     }
48     for (int i = 0; i < a.size(); i++) {
49         cout << a[i] + 1 << " \n"[i == a.size() - 1];
50     }
51     return 0;
52 }
```

## 787: Tickets

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

As a big fan of Formula One, Charlie is really happy with the fact that he has to organize ticket sells for the next Grand Prix race in his own city. Unfortunately, the financial crisis is striking everywhere and all the banknotes left in his country are valued either 10 euros or 20 euros. The price of all tickets for the race is 10 euros, so whenever someone comes to the ticket store only with 20 euro banknote Charlie must have a 10 euro banknote to give them change. Charlie realizes that with the huge deficit of banknotes this could be a problem. Charlie has some priceless information but couldn't make use of it, so he needs your help. Exactly  $n + m$  people will come to buy a ticket.  $n$  of them will have only a single 10 euro banknote, and  $m$  of them will have only a single 20 euro banknote. Currently Charlie has  $k$  10 euro banknotes, which he can use for change if needed. All  $n + m$  people will come to the ticket store in

random order, all orders are equiprobable. Return the probability that the ticket selling process will run smoothly, i.e. Charlie will have change for every person with 20 euro banknote.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m, k;
8     cin >> n >> m >> k;
9     cout << fixed << setprecision(10);
10    if (n - m < -k) {
11        cout << 0. << "\n";
12        return 0;
13    }
14    if (m <= k) {
15        cout << 1. << "\n";
16        return 0;
17    }
18    double ans = 1;
19    for (int i = 1; i <= k + 1; i++) {
20        ans *= m - i + 1;
21        ans /= n + i;
22    }
23    ans = 1 - ans;
24    cout << ans << "\n";
25    return 0;
26 }
```

## 788: XOR Counting

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Given two positive integers  $n$  and  $m$ . Find the sum of all possible values of  $a_1 \oplus a_2 \oplus \dots \oplus a_m$ , where  $a_1, a_2, \dots, a_m$  are non-negative integers such that  $a_1 + a_2 + \dots + a_m = n$ .

Note that all possible values  $a_1 \oplus a_2 \oplus \dots \oplus a_m$  should be counted in the sum exactly once.

As the answer may be too large, output your answer modulo 998244353.

Here,  $\oplus$  denotes the bitwise XOR operation.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 void solve() {
35     i64 n;
36     int m;
37     cin >> n >> m;
38     Z ans;
39     if (m == 1) {
40         ans = n;
41     } else if (m >= 3) {
42         if (n % 2 == 1) {
43             ans = Z(n + 1) * ((n + 1) / 2) / 2;
44         } else {
45             ans = Z(n) * (n / 2 + 1) / 2;
46         }
47     } else {
48         array<array<Z, 2>, 2> dp;
49         dp[0][0] = 1;
50         for (int i = 0; i < 60; i++) {
51             array<array<Z, 2>, 2> g;
52             for (int c = 0; c < 2; c++) {
53                 for (int x = 0; x <= 2; x++) {
54                     int nc = (c + x) / 2;
55                     if ((n >> i & 1) != (c + x) % 2) {
56                         continue;
57                     }
58                     g[nc][0] += dp[c][0];
59                     g[nc][1] += dp[c][1];
60                     g[nc][1] += dp[c][0] * (1LL << i) * (x % 2);
61                 }
62             }
63             dp = g;
64         }
}

```

```

65         ans = dp[0][1];
66     }
67     cout << ans << "\n";
68 }
69 int main() {
70     ios::sync_with_stdio(false);
71     cin.tie(nullptr);
72     int t;
73     cin >> t;
74     while (t--) {
75         solve();
76     }
77     return 0;
78 }
```

### 789: Broken robot

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You received as a gift a very clever robot walking on a rectangular board. Unfortunately, you understood that it is broken and behaves rather strangely (randomly). The board consists of N rows and M columns of cells. The robot is initially at some cell on the i-th row and the j-th column. Then at every step the robot could go to some another cell. The aim is to go to the bottommost (N-th) row. The robot can stay at its current cell, move to the left, move to the right, or move to the cell below the current. If the robot is in the leftmost column it cannot move to the left, and if it is in the rightmost column it cannot move to the right. At every step all possible moves are equally probable. Return the expected number of step to reach the bottommost row.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     cout << fixed << setprecision(10);
8     int N, M;
9     cin >> N >> M;
10    int i, j;
11    cin >> i >> j;
12    i--, j--;
13    vector<vector<double>> dp(N, vector<double>(M));
14    if (M == 1) {
15        cout << 2. * (N - 1 - i) << "\n";
```

```

16         return 0;
17     }
18     for (int i = N - 2; i >= 0; i--) {
19         vector<array<double, 4>> a(M);
20         for (int j = 0; j < M; j++) {
21             if (j == 0) {
22                 a[j][1] = 2;
23                 a[j][2] = -1;
24                 a[j][3] = dp[i + 1][j] + 3;
25             } else if (j < M - 1) {
26                 a[j][0] = -1;
27                 a[j][1] = 3;
28                 a[j][2] = -1;
29                 a[j][3] = dp[i + 1][j] + 4;
30             } else {
31                 a[j][0] = -1;
32                 a[j][1] = 2;
33                 a[j][3] = dp[i + 1][j] + 3;
34             }
35         }
36         for (int j = 0; j < M - 1; j++) {
37             double t = a[j + 1][0] / a[j][1];
38             a[j + 1][0] -= a[j][1] * t;
39             a[j + 1][1] -= a[j][2] * t;
40             a[j + 1][3] -= a[j][3] * t;
41         }
42         for (int j = M - 1; j; j--) {
43             double t = a[j - 1][2] / a[j][1];
44             a[j - 1][2] -= a[j][1] * t;
45             a[j - 1][3] -= a[j][3] * t;
46         }
47         for (int j = 0; j < M; j++) {
48             dp[i][j] = a[j][3] / a[j][1];
49         }
50     }
51     cout << dp[i][j] << "\n";
52     return 0;
53 }
```

**790: Tree**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Recently Bob invented a new game with a tree (we should remind you, that a tree is a connected graph without cycles): he deletes any (possibly, zero) amount of edges of the tree, and counts the product of sizes of the connected components left after the deletion. Your task is to find out the maximum number that Bob can get in his new game for a given tree.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1  n = int(input())
2  adj = [[] for _ in range(n)]
3  for _ in range(n-1) :
4      u, v = map(int, input().split(' '))
5      u -= 1
6      v -= 1
7      adj[u].append(v)
8      adj[v].append(u)
9  dp = [[1, 2] for _ in range(n)]
10 g = [1] * n
11 def dfs(x, p) :
12     global dp
13     prod = 1
14     deg = 0
15     a = []
16     for y in adj[x] :
17         if y == p :
18             continue
19         deg += 1
20         dfs(y, x)
21         dp[x][0] = max(dp[x][0] * dp[y][0], prod * dp[y][1])
22         g[x] *= dp[y][0]
23         prod *= dp[y][0]
24         a.append(y)
25     prod = 1
26     for y in a :
27         prod *= g[y]
28     a.sort(key = lambda y : g[y] / dp[y][0])
29     dp[x][0] = max(dp[x][0], prod * (deg + 1))
30     dp[x][1] = max(dp[x][1], prod * (deg + 2))
31     for y in a :
32         prod = prod // g[y] * dp[y][0]
33         deg -= 1
34         dp[x][0] = max(dp[x][0], prod * (deg + 1))
35         dp[x][1] = max(dp[x][1], prod * (deg + 2))
36 dfs(0, -1)
37 print(dp[0][0])

```

### 791: Tetragon

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You're given the centers of three equal sides of a strictly convex tetragon. Your task is to restore the initial tetragon.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 # struct Point;
6 template<class T>
7 T dot(Point<T> a, Point<T> b) {
8     return a.x * b.x + a.y * b.y;
9 }
10 template<class T>
11 T cross(Point<T> a, Point<T> b) {
12     return a.x * b.y - a.y * b.x;
13 }
14 template<class T>
15 T square(Point<T> p) {
16     return dot(p, p);
17 }
18 template<class T>
19 double length(Point<T> p) {
20     return sqrt(double(square(p)));
21 }
22 long double length(Point<long double> p) {
23     return sqrt(square(p));
24 }
25 template<class T>
26 # struct Line;
27 template<class T>
28 Point<T> rotate(Point<T> a) {
29     return Point(-a.y, a.x);
30 }
31 template<class T>
32 int sgn(Point<T> a) {
33     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
34 }
35 template<class T>
36 bool pointOnLineLeft(Point<T> p, Line<T> l) {
37     return cross(l.b - l.a, p - l.a) > 1E-8;
38 }
39 template<class T>
40 Point<T> lineIntersection(Line<T> l1, Line<T> l2) {
41     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1
42         .a - l1.b));
43 }
44 template<class T>
45 bool pointOnSegment(Point<T> p, Line<T> l) {
46     return cross(p - l.a, l.b - l.a) == 0 && min(l.a.x, l.b.x) <= p.x && p.x <= max(l.a.x,
47         l.b.x)
48         && min(l.a.y, l.b.y) <= p.y && p.y <= max(l.a.y, l.b.y);
49 }
50 template<class T>
51 bool pointInPolygon(Point<T> a, vector<Point<T>> p) {
52     int n = p.size();
53     for (int i = 0; i < n; i++) {
54         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
55             return true;
56         }
57     }
58     int t = 0;
59     for (int i = 0; i < n; i++) {
60         auto u = p[i];
61         auto v = p[(i + 1) % n];
62         if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
63             t ^= 1;
64         }
65     }
66 }
```

```

63         if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
64             t ^= 1;
65         }
66     }
67     return t == 1;
68 }
69 // 0 : not intersect
70 // 1 : strictly intersect
71 // 2 : overlap
72 // 3 : intersect at endpoint
73 template<class T>
74 tuple<int, Point<T>, Point<T>> segmentIntersection(Line<T> l1, Line<T> l2) {
75     if (max(l1.a.x, l1.b.x) < min(l2.a.x, l2.b.x)) {
76         return {0, Point<T>(), Point<T>()};
77     }
78     if (min(l1.a.x, l1.b.x) > max(l2.a.x, l2.b.x)) {
79         return {0, Point<T>(), Point<T>()};
80     }
81     if (max(l1.a.y, l1.b.y) < min(l2.a.y, l2.b.y)) {
82         return {0, Point<T>(), Point<T>()};
83     }
84     if (min(l1.a.y, l1.b.y) > max(l2.a.y, l2.b.y)) {
85         return {0, Point<T>(), Point<T>()};
86     }
87     if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
88         if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
89             return {0, Point<T>(), Point<T>()};
90         } else {
91             int maxx1 = max(l1.a.x, l1.b.x);
92             int minx1 = min(l1.a.x, l1.b.x);
93             int maxy1 = max(l1.a.y, l1.b.y);
94             int miny1 = min(l1.a.y, l1.b.y);
95             int maxx2 = max(l2.a.x, l2.b.x);
96             int minx2 = min(l2.a.x, l2.b.x);
97             int maxy2 = max(l2.a.y, l2.b.y);
98             int miny2 = min(l2.a.y, l2.b.y);
99             Point<T> p1(max(minx1, minx2), max(miny1, miny2));
100            Point<T> p2(min(maxx1, maxx2), min(maxy1, maxy2));
101            if (!pointOnSegment(p1, l1)) {
102                swap(p1.y, p2.y);
103            }
104            if (p1 == p2) {
105                return {3, p1, p2};
106            } else {
107                return {2, p1, p2};
108            }
109        }
110    }
111    auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
112    auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
113    auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
114    auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
115    if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) || (cp3 < 0
116        && cp4 < 0)) {
117        return {0, Point<T>(), Point<T>()};
118    }
119    Point p = lineIntersection(l1, l2);
120    if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
121        return {1, p, p};
122    } else {
123        return {3, p, p};
124    }
125 }
```

```
126 bool segmentInPolygon(Line<T> l, vector<Point<T>> p) {
127     int n = p.size();
128     if (!pointInPolygon(l.a, p)) {
129         return false;
130     }
131     if (!pointInPolygon(l.b, p)) {
132         return false;
133     }
134     for (int i = 0; i < n; i++) {
135         auto u = p[i];
136         auto v = p[(i + 1) % n];
137         auto w = p[(i + 2) % n];
138         auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
139         if (t == 1) {
140             return false;
141         }
142         if (t == 0) {
143             continue;
144         }
145         if (t == 2) {
146             if (pointOnSegment(v, l) && v != l.a && v != l.b) {
147                 if (cross(v - u, w - v) > 0) {
148                     return false;
149                 }
150             }
151         } else {
152             if (p1 != u && p1 != v) {
153                 if (pointOnLineLeft(l.a, Line(v, u))
154                     || pointOnLineLeft(l.b, Line(v, u))) {
155                     return false;
156                 }
157             } else if (p1 == v) {
158                 if (l.a == v) {
159                     if (pointOnLineLeft(u, l)) {
160                         if (pointOnLineLeft(w, l)
161                             && pointOnLineLeft(w, Line(u, v))) {
162                             return false;
163                         }
164                     } else {
165                         if (pointOnLineLeft(w, l)
166                             || pointOnLineLeft(w, Line(u, v))) {
167                             return false;
168                         }
169                     }
170                 } else if (l.b == v) {
171                     if (pointOnLineLeft(u, Line(l.b, l.a))) {
172                         if (pointOnLineLeft(w, Line(l.b, l.a))
173                             && pointOnLineLeft(w, Line(u, v))) {
174                             return false;
175                         }
176                     } else {
177                         if (pointOnLineLeft(w, Line(l.b, l.a))
178                             || pointOnLineLeft(w, Line(u, v))) {
179                             return false;
180                         }
181                     }
182                 } else {
183                     if (pointOnLineLeft(u, l)) {
184                         if (pointOnLineLeft(w, Line(l.b, l.a))
185                             || pointOnLineLeft(w, Line(u, v))) {
186                             return false;
187                         }
188                     } else {
189                         if (pointOnLineLeft(w, l)
```

```
190                         || pointOnLineLeft(w, Line(u, v))) {  
191                     return false;  
192                 }  
193             }  
194         }  
195     }  
196 }  
197     return true;  
198 }  
199 }  
200 template<class T>  
201 vector<Point<T>> hp(vector<Line<T>> lines) {  
202     sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {  
203         auto d1 = l1.b - l1.a;  
204         auto d2 = l2.b - l2.a;  
205         if (sgn(d1) != sgn(d2)) {  
206             return sgn(d1) == 1;  
207         }  
208         return cross(d1, d2) > 0;  
209     });  
210     deque<Line<T>> ls;  
211     deque<Point<T>> ps;  
212     for (auto l : lines) {  
213         if (ls.empty()) {  
214             ls.push_back(l);  
215             continue;  
216         }  
217         while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {  
218             ps.pop_back();  
219             ls.pop_back();  
220         }  
221         while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {  
222             ps.pop_front();  
223             ls.pop_front();  
224         }  
225         if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {  
226             if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {  
227                 if (!pointOnLineLeft(ls.back().a, l)) {  
228                     assert(ls.size() == 1);  
229                     ls[0] = l;  
230                 }  
231                 continue;  
232             }  
233             return {};  
234         }  
235         ps.push_back(lineIntersection(ls.back(), l));  
236         ls.push_back(l);  
237     }  
238     while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {  
239         ps.pop_back();  
240         ls.pop_back();  
241     }  
242     if (ls.size() <= 2) {  
243         return {};  
244     }  
245     ps.push_back(lineIntersection(ls[0], ls.back()));  
246     return vector(ps.begin(), ps.end());  
247 }  
248 void solve() {  
249     Point<double> p[3];  
250     for (int i = 0; i < 3; i++) {  
251         int x, y;  
252         cin >> x >> y;  
253         p[i].x = x;
```

```

254         p[i].y = y;
255     }
256     if (cross(p[1] - p[0], p[2] - p[0]) == 0) {
257         cout << "NO\n";
258         return;
259     }
260     if (cross(p[1] - p[0], p[2] - p[0]) < 0) {
261         swap(p[1], p[2]);
262     }
263     for (int t = 0; t < 3; t++) {
264         rotate(p, p + 1, p + 3);
265         auto A = p[0], B = p[1], C = p[2];
266         Line LAB(0.5 * (A + B), 0.5 * (A + B) + rotate(B - A));
267         Line LAC(0.5 * (A + C), 0.5 * (A + C) + rotate(C - A));
268         auto p = lineIntersection(LAB, LAC);
269         Point q = 2 * A - p;
270         Line l(q, q + rotate(B - A));
271         auto X = lineIntersection(l, LAC);
272         auto Y = 2 * A - X;
273         auto Z = 2 * B - Y;
274         auto W = 2 * C - X;
275         if (!pointOnLineLeft(W, Line(Y, Z))) {
276             continue;
277         }
278         if (!pointOnLineLeft(Z, Line(W, X))) {
279             continue;
280         }
281         if (!pointOnLineLeft(W, Line(X, Y))) {
282             continue;
283         }
284         if (!pointOnLineLeft(Z, Line(X, Y))) {
285             continue;
286         }
287         cout << "YES\n";
288         cout << X.x << " " << X.y << " " << Y.x << " " << Y.y << " "
289             << Z.x << " " << Z.y << " " << W.x << " " << W.y << "\n";
290         return;
291     }
292     cout << "NO\n";
293 }
294 int main() {
295     ios::sync_with_stdio(false);
296     cin.tie(nullptr);
297     cout << fixed << setprecision(10);
298     int t;
299     cin >> t;
300     while (t--) {
301         solve();
302     }
303     return 0;
304 }
```

## 792: Traveling Graph

- Time limit: 0.5 second
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

You are given undirected weighted graph. Find the length of the shortest cycle which starts from the vertex 1 and passes through all the edges at least once. Graph may contain multiply edges between a pair of vertices and loops (edges from the vertex to itself).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     int S = 0;
10    vector<bool> ed(n);
11    vector dis(n, vector<int>(n, 1E9));
12    int ans = 0;
13    for (int i = 0; i < m; i++) {
14        int a, b, c;
15        cin >> a >> b >> c;
16        a--, b--;
17        ans += c;
18        S ^= 1 << a;
19        S ^= 1 << b;
20        ed[a] = ed[b] = true;
21        dis[a][b] = dis[b][a] = min(dis[a][b], c);
22    }
23    for (int i = 0; i < n; i++) {
24        dis[i][i] = 0;
25    }
26    for (int k = 0; k < n; k++) {
27        for (int i = 0; i < n; i++) {
28            for (int j = 0; j < n; j++) {
29                dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
30            }
31        }
32    }
33    for (int i = 0; i < n; i++) {
34        if (dis[0][i] == 1E9 && ed[i]) {
35            cout << -1 << "\n";
36            return 0;
37        }
38    }
39    int min = 1E9;
40    function<void(int, int)> dfs = [&](int s, int sum) {
41        if (s == 0) {
42            min = min(min, sum);
43            return;
44        }
45        int x = __builtin_ctz(s);
46        for (int i = x + 1; i < n; i++) {
47            if (s >> i & 1) {
48                dfs(s ^ (1 << x) ^ (1 << i), sum + dis[x][i]);
49            }
50        }
51    };
52    dfs(S, 0);
53    ans += min;
54    cout << ans << "\n";

```

```

55     return 0;
56 }
```

### 793: Notepad

- Time limit: 2 seconds
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

Nick is attracted by everything unconventional. He doesn't like decimal number system any more, and he decided to study other number systems. A number system with base b caught his attention. Before he starts studying it, he wants to write in his notepad all the numbers of length n without leading zeros in this number system. Each page in Nick's notepad has enough space for c numbers exactly. Nick writes every suitable number only once, starting with the first clean page and leaving no clean spaces. Nick never writes number 0 as he has unpleasant memories about zero divide.

Would you help Nick find out how many numbers will be written on the last page.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
```

```

29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 using Z = MInt<0>;
33 int main() {
34     ios::sync_with_stdio(false);
35     cin.tie(nullptr);
36     string sb, sn;
37     int c;
38     cin >> sb >> sn >> c;
39     Z::setMod(c);
40     Z b;
41     for (auto x : sb) {
42         b = 10 * b + (x - '0');
43     }
44     Z ans = b - 1;
45     Z pb[10];
46     pb[0] = 1;
47     for (int i = 1; i < 10; i++) {
48         pb[i] = pb[i - 1] * b;
49     }
50     int t = sn.size() - 1;
51     while (sn[t] == '0') {
52         sn[t] = '9';
53         t--;
54     }
55     sn[t]--;
56     b = 1;
57     for (auto x : sn) {
58         Z b2 = b * b;
59         Z b4 = b2 * b2;
60         b = b4 * b4 * b2;
61         b *= pb[x - '0'];
62     }
63     ans *= b;
64     int res = int(ans);
65     if (res == 0) {
66         res = c;
67     }
68     cout << res << "\n";
69     return 0;
70 }
```

## 794: Balance

- Time limit: 3 seconds
- Memory limit: 128 megabytes
- Input file: standard input
- Output file: standard output

Nick likes strings very much, he likes to rotate them, sort them, rearrange characters within a string... Once he wrote a random string of characters a, b, c on a piece of paper and began to perform the following operations:

to take two adjacent characters and replace the second character with the first one,

to take two adjacent characters and replace the first character with the second one

To understand these actions better, let's take a look at a string abc. All of the following strings can be obtained by performing one of the described operations on abc: bbc, abb, acc. Let's denote the frequency of a character for each of the characters a, b and c as the number of occurrences of this character in the string. For example, for string abc:  $|a| = 1$ ,  $|b| = 1$ ,  $|c| = 1$ , and for string bbc:  $|a| = 0$ ,  $|b| = 2$ ,  $|c| = 1$ .

While performing the described operations, Nick sometimes got balanced strings. Let's say that a string is balanced, if the frequencies of each character differ by at most 1. That is  $-1 \leq |a| - |b| \leq 1$ ,  $-1 \leq |a| - |c| \leq 1$ ,  $-1 \leq |b| - |c| \leq 1$ .

Would you help Nick find the number of different balanced strings that can be obtained by performing the operations described above, perhaps multiple times, on the given string s. This number should be calculated modulo 51123987.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 51123987;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);

```

```
36     cin.tie(nullptr);
37     int n;
38     cin >> n;
39     string s;
40     cin >> s;
41     vector f(3, vector(n + 1, n + 1));
42     for (int c = 0; c < 3; c++) {
43         for (int i = n - 1; i >= 0; i--) {
44             f[c][i] = s[i] == 'a' + c ? i + 1 : f[c][i + 1];
45         }
46     }
47     vector dp(n + 1, vector(n + 1, vector(n + 1, array<Z, 3>{0})));
48     dp[0][0][0][0] = 1;
49     Z ans = 0;
50     for (int i = 0; i <= n; i++) {
51         for (int a = i; a >= 0; a--) {
52             for (int b = i - a; b >= 0; b--) {
53                 int c = i - a - b;
54                 for (int k = n; k >= 0; k--) {
55                     for (int x = 0; x < 3; x++) {
56                         if (i == n) {
57                             if (abs(a - c) <= 1 && abs(b - c) <= 1 && abs(a - b) <= 1) {
58                                 ans += dp[a][b][k][x];
59                             }
60                         } else {
61                             Z val = dp[a][b][k][x];
62                             dp[a][b][k][x] = 0;
63                             if (x == 0 && i > 0) {
64                                 dp[a + 1][b][k][0] += val;
65                             } else if (f[0][k] <= n) {
66                                 dp[a + 1][b][f[0][k]][0] += val;
67                             }
68                             if (x == 1 && i > 0) {
69                                 dp[a][b + 1][k][1] += val;
70                             } else if (f[1][k] <= n) {
71                                 dp[a][b + 1][f[1][k]][1] += val;
72                             }
73                             if (x == 2 && i > 0) {
74                                 dp[a][b][k][2] += val;
75                             } else if (f[2][k] <= n) {
76                                 dp[a][b][f[2][k]][2] += val;
77                             }
78                         }
79                     }
80                 }
81             }
82         }
83     }
84     cout << ans << "\n";
85     return 0;
86 }
```

**red****combinatorics****795: Last Number**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a multiset  $S$ . Initially,  $S = \{1, 2, 3, \dots, n\}$ .

You will perform the following operation  $n - 1$  times.

Choose the largest number  $S_{\max}$  in  $S$  and the smallest number  $S_{\min}$  in  $S$ . Remove the two numbers from  $S$ , and add  $S_{\max} - S_{\min}$  into  $S$ .

It's easy to show that there will be exactly one number left after  $n - 1$  operations. Output that number.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 const long double phi = (1 + sqrt(5.0)) / 2;
5 i64 floor_sum(i64 n, i64 m, i64 a, i64 b) {
6     i64 ans = 0;
7     if (a >= m) {
8         ans += (n - 1) * n * (a / m) / 2;
9         a %= m;
10    }
11    if (b >= m) {
12        ans += n * (b / m);
13        b %= m;
14    }
15    long long y_max = (a * n + b) / m, x_max = (y_max * m - b);
16    if (y_max == 0) return ans;
17    ans += (n - (x_max + a - 1) / a) * y_max;
18    ans += floor_sum(y_max, a, m, (a - x_max % a) % a);
19    return ans;
20}
21 constexpr i64 X = 701408733;
22 constexpr i64 Y = 1134903170;
23 int g(int n) {
24     int ans = 0;
25     ans -= floor_sum(n / 2 + 1, Y, 2 * X, 0);
26     ans += floor_sum((n + 1) / 2, Y, 2 * X, X);
27     return ans;
28}
29 int get(int n) {
30     int t = (n + 1) * Y / (2 * Y + X);
31     int ans = 0;
```

```

32     ans += g(n - 1);
33     ans -= g(t);
34     if (n % 2 == 0) {
35         ans *= -1;
36     }
37     ans += int(n * X / Y) * ((n - t - 1) % 2 == 0 ? -1 : 1);
38     if ((n - t) % 2 == 1) {
39         ans += n;
40     }
41     return ans;
42 }
43 void solve() {
44     int n;
45     cin >> n;
46     cout << get(n) << "\n";
47 }
48 int main() {
49     ios::sync_with_stdio(false);
50     cin.tie(nullptr);
51     int t;
52     cin >> t;
53     while (t--) {
54         solve();
55     }
56     return 0;
57 }
```

## 796: Festival Organization

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The Prodiggers are quite a cool band and for this reason, they have been the surprise guest at the ENTER festival for the past 80 years. At the beginning of their careers, they weren't so successful, so they had to spend time digging channels to earn money; hence the name. Anyway, they like to tour a lot and have surprising amounts of energy to do extremely long tours. However, they hate spending two consecutive days without having a concert, so they would like to avoid it.

A tour is defined by a sequence of concerts and days-off. You need to count in how many ways The Prodiggers can select  $k$  different tours of the same length between  $l$  and  $r$ .

For example if  $k = 2$ ,  $l = 1$  and  $r = 2$ , if we define concert day as  $\{1\}$  and day-off as  $\{0\}$ , here are all possible tours:  $\{0\}, \{1\}, \{00\}, \{01\}, \{10\}, \{11\}$ . But tour  $00$  can not be selected because it has 2 days-off in a row. Now, we need to count in how many ways we can select  $k = 2$  tours of the same length in range  $[1;2]$ . Here they are:  $\{0,1\}; \{01,10\}; \{01,11\}; \{10,11\}$ .

Since their schedule is quite busy, they want you to tell them in how many ways can do that, modulo 1 000 000 007 (109 + 7).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 template<int P>
15 # struct MInt;
16 template<>
17 int MInt<0>::Mod = 1;
18 template<int V, int P>
19 constexpr MInt<P> CInv = MInt<P>(V).inv();
20 constexpr int P = 1000000007;
21 using Z = MInt<P>;
22 vector<int> rev;
23 template<int P>
24 vector<MInt<P>> roots{0, 1};
25 template<int P>
26 constexpr MInt<P> findPrimitiveRoot() {
27     MInt<P> i = 2;
28     int k = __builtin_ctz(P - 1);
29     while (true) {
30         if (power(i, (P - 1) / 2) != 1) {
31             break;
32         }
33         i += 1;
34     }
35     return power(i, (P - 1) >> k);
36 }
37 template<int P>
38 constexpr MInt<P> primitiveRoot = findPrimitiveRoot<P>();
39 template<>
40 constexpr MInt<998244353> primitiveRoot<998244353> {31};
41 template<int P>
42 constexpr void dft(vector<MInt<P>> &a) {
43     int n = a.size();
44     if (int(rev.size()) != n) {
45         int k = __builtin_ctz(n) - 1;
46         rev.resize(n);
47         for (int i = 0; i < n; i++) {
48             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
49         }
50     }
51     for (int i = 0; i < n; i++) {
52         if (rev[i] < i) {
53             swap(a[i], a[rev[i]]);
54         }
55     }
56     if (roots<P>.size() < n) {
57         int k = __builtin_ctz(roots<P>.size());
58         roots<P>.resize(n);

```

```

59         while ((1 << k) < n) {
60             auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k - 1));
61             for (int i = 1 << (k - 1); i < (1 << k); i++) {
62                 roots<P>[2 * i] = roots<P>[i];
63                 roots<P>[2 * i + 1] = roots<P>[i] * e;
64             }
65             k++;
66         }
67     }
68     for (int k = 1; k < n; k *= 2) {
69         for (int i = 0; i < n; i += 2 * k) {
70             for (int j = 0; j < k; j++) {
71                 MInt<P> u = a[i + j];
72                 MInt<P> v = a[i + j + k] * roots<P>[k + j];
73                 a[i + j] = u + v;
74                 a[i + j + k] = u - v;
75             }
76         }
77     }
78 }
79 template<int P>
80 constexpr void idft(vector<MInt<P>> &a) {
81     int n = a.size();
82     reverse(a.begin() + 1, a.end());
83     dft(a);
84     MInt<P> inv = (1 - P) / n;
85     for (int i = 0; i < n; i++) {
86         a[i] *= inv;
87     }
88 }
89 template<int P = 998244353>
90 # struct Poly;
91 template<int P = 998244353>
92 Poly<P> berlekampMassey(const Poly<P> &s) {
93     Poly<P> c;
94     Poly<P> oldC;
95     int f = -1;
96     for (int i = 0; i < s.size(); i++) {
97         auto delta = s[i];
98         for (int j = 1; j <= c.size(); j++) {
99             delta -= c[j - 1] * s[i - j];
100        }
101        if (delta == 0) {
102            continue;
103        }
104        if (f == -1) {
105            c.resize(i + 1);
106            f = i;
107        } else {
108            auto d = oldC;
109            d *= -1;
110            d.insert(d.begin(), 1);
111            MInt<P> df1 = 0;
112            for (int j = 1; j <= d.size(); j++) {
113                df1 += d[j - 1] * s[f + 1 - j];
114            }
115            assert(df1 != 0);
116            auto coef = delta / df1;
117            d *= coef;
118            Poly<P> zeros(i - f - 1);
119            zeros.insert(zeros.end(), d.begin(), d.end());
120            d = zeros;
121            auto temp = c;
122            c += d;

```

```
123         if (i - temp.size() > f - oldC.size()) {
124             oldC = temp;
125             f = i;
126         }
127     }
128 }
129 c *= -1;
130 c.insert(c.begin(), 1);
131 return c;
132 }
133 template<int P = 998244353>
134 MInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
135     int m = q.size() - 1;
136     while (n > 0) {
137         auto newq = q;
138         for (int i = 1; i <= m; i += 2) {
139             newq[i] *= -1;
140         }
141         auto newp = p * newq;
142         newq = q * newq;
143         for (int i = 0; i < m; i++) {
144             p[i] = newp[i * 2 + n % 2];
145         }
146         for (int i = 0; i <= m; i++) {
147             q[i] = newq[i * 2];
148         }
149         n /= 2;
150     }
151     return p[0] / q[0];
152 }
153 int main() {
154     ios::sync_with_stdio(false);
155     cin.tie(nullptr);
156     int k;
157     i64 l, r;
158     cin >> k >> l >> r;
159     int N = 8 * k;
160     vector<Z> f(N + 1);
161     f[0] = 1;
162     f[1] = 2;
163     for (int i = 2; i <= N; i++) {
164         f[i] = f[i - 1] + f[i - 2];
165     }
166     for (int i = 0; i <= N; i++) {
167         Z res = 1;
168         for (int j = 1; j <= k; j++) {
169             res = res * (f[i] - j + 1) / j;
170         }
171         f[i] = res;
172     }
173     for (int i = 1; i <= N; i++) {
174         f[i] += f[i - 1];
175     }
176     auto h = berlekampMassey(Poly(f));
177     auto g = (Poly(f) * h).trunc(h.size() - 1);
178     auto ans = linearRecurrence(g, h, r) - linearRecurrence(g, h, l - 1);
179     cout << ans << "\n";
180     return 0;
181 }
```

## 797: Graph Coloring (easy version)

- Time limit: 5.5 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

The only difference between the easy and the hard version is the constraint on  $n$ .

You are given an undirected complete graph on  $n$  vertices. A complete graph is a graph where each pair of vertices is connected by an edge. You have to paint the edges of the graph into two colors, red and blue (each edge will have one color).

A set of vertices  $S$  is red-connected if, for every pair of vertices  $(v_1, v_2)$  such that  $v_1 \in S$  and  $v_2 \in S$ , there exists a path from  $v_1$  to  $v_2$  that goes only through red edges and vertices from  $S$ . Similarly, a set of vertices  $S$  is blue-connected if, for every pair of vertices  $(v_1, v_2)$  such that  $v_1 \in S$  and  $v_2 \in S$ , there exists a path from  $v_1$  to  $v_2$  that goes only through blue edges and vertices from  $S$ .

You have to paint the graph in such a way that:

there is at least one red edge;

there is at least one blue edge;

for each set of vertices  $S$  such that  $|S| \geq 2$ ,  $S$  is either red-connected or blue-connected, but not both.

Calculate the number of ways to paint the graph, and print it modulo 998244353.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 998244353;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
8     if (x < 0) {
9         x += P;
10    }
11    if (x >= P) {
12        x -= P;
13    }
14    return x;
15}
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;

```

```

19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25 }
26 # struct Z;
27 vector<int> rev;
28 vector<Z> roots{0, 1};
29 void dft(vector<Z> &a) {
30     int n = a.size();
31     if (int(rev.size()) != n) {
32         int k = __builtin_ctz(n) - 1;
33         rev.resize(n);
34         for (int i = 0; i < n; i++) {
35             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
36         }
37     }
38     for (int i = 0; i < n; i++) {
39         if (rev[i] < i) {
40             swap(a[i], a[rev[i]]);
41         }
42     }
43     if (int(roots.size()) < n) {
44         int k = __builtin_ctz(roots.size());
45         roots.resize(n);
46         while ((1 << k) < n) {
47             Z e = power(Z(3), (P - 1) >> (k + 1));
48             for (int i = 1 << (k - 1); i < (1 << k); i++) {
49                 roots[2 * i] = roots[i];
50                 roots[2 * i + 1] = roots[i] * e;
51             }
52             k++;
53         }
54     }
55     for (int k = 1; k < n; k *= 2) {
56         for (int i = 0; i < n; i += 2 * k) {
57             for (int j = 0; j < k; j++) {
58                 Z u = a[i + j];
59                 Z v = a[i + j + k] * roots[k + j];
60                 a[i + j] = u + v;
61                 a[i + j + k] = u - v;
62             }
63         }
64     }
65 }
66 void idft(vector<Z> &a) {
67     int n = a.size();
68     reverse(a.begin() + 1, a.end());
69     dft(a);
70     Z inv = (1 - P) / n;
71     for (int i = 0; i < n; i++) {
72         a[i] *= inv;
73     }
74 }
75 # struct Poly;
76 int main() {
77     ios::sync_with_stdio(false);
78     cin.tie(nullptr);
79     int n;
80     cin >> n;
81     Poly f{0};
82     int m = 1;

```

```

83     while (m <= n) {
84         m *= 2;
85         auto ef = f.exp(m);
86         f = (f - (ef - 2 * f + Poly{-1, 1}) * (ef - Poly{2}).inv(m)).modxk(m);
87     }
88     auto ans = f[n];
89     for (int i = 1; i <= n; i++) {
90         ans *= i;
91     }
92     ans = (ans - 1) * 2;
93     cout << ans << "\n";
94     return 0;
95 }
```

### 798: Graph Coloring (hard version)

- Time limit: 5.5 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

The only difference between the easy and the hard version is the constraint on  $n$ .

You are given an undirected complete graph on  $n$  vertices. A complete graph is a graph where each pair of vertices is connected by an edge. You have to paint the edges of the graph into two colors, red and blue (each edge will have one color).

A set of vertices  $S$  is red-connected if, for every pair of vertices  $(v_1, v_2)$  such that  $v_1 \in S$  and  $v_2 \in S$ , there exists a path from  $v_1$  to  $v_2$  that goes only through red edges and vertices from  $S$ . Similarly, a set of vertices  $S$  is blue-connected if, for every pair of vertices  $(v_1, v_2)$  such that  $v_1 \in S$  and  $v_2 \in S$ , there exists a path from  $v_1$  to  $v_2$  that goes only through blue edges and vertices from  $S$ .

You have to paint the graph in such a way that:

there is at least one red edge;

there is at least one blue edge;

for each set of vertices  $S$  such that  $|S| \geq 2$ ,  $S$  is either red-connected or blue-connected, but not both.

Calculate the number of ways to paint the graph, and print it modulo 998244353.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 998244353;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
8     if (x < 0) {
9         x += P;
10    }
11    if (x >= P) {
12        x -= P;
13    }
14    return x;
15 }
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25 }
26 # struct Z;
27 vector<int> rev;
28 vector<Z> roots{0, 1};
29 void dft(vector<Z> &a) {
30     int n = a.size();
31     if (int(rev.size()) != n) {
32         int k = __builtin_ctz(n) - 1;
33         rev.resize(n);
34         for (int i = 0; i < n; i++) {
35             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
36         }
37     }
38     for (int i = 0; i < n; i++) {
39         if (rev[i] < i) {
40             swap(a[i], a[rev[i]]);
41         }
42     }
43     if (int(roots.size()) < n) {
44         int k = __builtin_ctz(roots.size());
45         roots.resize(n);
46         while ((1 << k) < n) {
47             Z e = power(Z(3), (P - 1) >> (k + 1));
48             for (int i = 1 << (k - 1); i < (1 << k); i++) {
49                 roots[2 * i] = roots[i];
50                 roots[2 * i + 1] = roots[i] * e;
51             }
52             k++;
53         }
54     }
55     for (int k = 1; k < n; k *= 2) {
56         for (int i = 0; i < n; i += 2 * k) {
57             for (int j = 0; j < k; j++) {
58                 Z u = a[i + j];
59                 Z v = a[i + j + k] * roots[k + j];
60                 a[i + j] = u + v;
61                 a[i + j + k] = u - v;
62             }
63         }
64     }
}

```

```

65 }
66 void idft(vector<Z> &a) {
67     int n = a.size();
68     reverse(a.begin() + 1, a.end());
69     dft(a);
70     Z inv = (1 - P) / n;
71     for (int i = 0; i < n; i++) {
72         a[i] *= inv;
73     }
74 }
75 # struct Poly;
76 int main() {
77     ios::sync_with_stdio(false);
78     cin.tie(nullptr);
79     int n;
80     cin >> n;
81     Poly f{0};
82     int m = 1;
83     while (m <= n) {
84         m *= 2;
85         auto ef = f.exp(m);
86         f = (f - (ef - 2 * f + Poly{-1, 1}) * (ef - Poly{2}).inv(m)).modxk(m);
87     }
88     auto ans = f[n];
89     for (int i = 1; i <= n; i++) {
90         ans *= i;
91     }
92     ans = (ans - 1) * 2;
93     cout << ans << "\n";
94     return 0;
95 }

```

## 799: Majority

- Time limit: 2 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

Everyone was happy coding, until suddenly a power shortage happened and the best competitive programming site went down. Fortunately, a system administrator bought some new equipment recently, including some UPSs. Thus there are some servers that are still online, but we need all of them to be working in order to keep the round rated.

Imagine the servers being a binary string  $s$  of length  $n$ . If the  $i$ -th server is online, then  $s_i = 1$ , and  $s_i = 0$  otherwise.

A system administrator can do the following operation called electricity spread, that consists of the following phases:

Select two servers at positions  $1 \leq i < j \leq n$  such that both are online (i.e.  $s_i = s_j = 1$ ). The spread starts only from online servers.

Check if we have enough power to make the spread. We consider having enough power if the number of turned on servers in range  $[i, j]$  is at least the number of turned off servers in range  $[i, j]$ . More formally, check whether  $2 \cdot (s_i + s_{i+1} + \dots + s_j) \geq j - i + 1$ .

If the check is positive, turn on all the offline servers in range  $[i, j]$ . More formally, make  $s_k := 1$  for all  $k$  from  $i$  to  $j$ .

We call a binary string  $s$  of length  $n$  rated if we can turn on all servers (i.e. make  $s_i = 1$  for  $1 \leq i \leq n$ ) using the electricity spread operation any number of times (possibly, 0). Your task is to find the number of rated strings of length  $n$  modulo  $m$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector f(n + 1, vector<int>(n + 1));
10    f[1][1] = 1;
11    vector<int> s(2 * n + 1);
12    s[2] = 1;
13    int p = 1;
14    for (int i = 2; i <= n; i++) {
15        if (i > 2) {
16            p = 2 * p % m;
17        }
18        int v = p;
19        auto sum = s;
20        for (int j = 1; j <= 2 * n; j++) {
21            sum[j] = (sum[j] + sum[j - 1]) % m;
22        }
23        for (int j = 1; j < i; j++) {
24            f[i][j] = 1LL * sum[max(0, i - 2 * j - 1)] * f[j][j] % m;
25            v = (v - f[i][j] + m) % m;
26            s[i + j] = (s[i + j] + f[i][j]) % m;
27        }
28        f[i][i] = v;
29        s[i + i] = (s[i + i] + f[i][i]) % m;
30    }
31    cout << f[n][n] << "\n";
32    return 0;
33 }
```

## 800: List Generation

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

For given integers  $n$  and  $m$ , let's call a pair of arrays  $a$  and  $b$  of integers good, if they satisfy the following conditions:

$a$  and  $b$  have the same length, let their length be  $k$ .

$k \geq 2$  and  $a_1 = 0, a_k = n, b_1 = 0, b_k = m$ .

For each  $1 < i \leq k$  the following holds:  $a_i \geq a_{i-1}, b_i \geq b_{i-1}$ , and  $a_i + b_i \neq a_{i-1} + b_{i-1}$ .

Find the sum of  $|a|$  over all good pairs of arrays  $(a, b)$ . Since the answer can be very large, output it modulo  $10^9 + 7$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 1000000007;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
8     if (x < 0) {
9         x += P;
10    }
11    if (x >= P) {
12        x -= P;
13    }
14    return x;
15}
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25}
26 # struct Z;
27 constexpr int N = 1E7;
28 Z fac[N + 1], invfac[N + 1], p[N + 1];
29 Z get(int n, int m) {
30     if (!n && !m) {
31         return 1;
32     }
33     Z ans = 0;
34     for (int k = 0; k <= min(n, m); k++) {
35         ans += (k & 1 ? -1 : 1) * p[n + m - k] * fac[n + m - k] * invfac[n - k] * invfac[m - k] * invfac[k];
36         ans += (k & 1 ? -1 : 1) * p[n + m - k - 1] * fac[n + m - k] * invfac[n - k] * invfac[m - k] * invfac[k] * invfac[m - k - 1];
37     }
38     return ans;
39 }
40 void solve() {

```

```

41     int n, m;
42     cin >> n >> m;
43     cout << get(n, m) - get(n - 1, m) - get(n, m - 1) + get(n - 1, m - 1) << "\n";
44 }
45 int main() {
46     ios::sync_with_stdio(false);
47     cin.tie(nullptr);
48     fac[0] = p[0] = 1;
49     for (int i = 1; i <= N; i++) {
50         fac[i] = fac[i - 1] * i;
51         p[i] = p[i - 1] * 2;
52     }
53     invfac[N] = fac[N].inv();
54     for (int i = N; i; i--) {
55         invfac[i - 1] = invfac[i] * i;
56     }
57     int t;
58     cin >> t;
59     while (t--) {
60         solve();
61     }
62     return 0;
63 }
```

## 801: Antifibonacci Cut

- Time limit: 12 seconds
- Memory limit: 4 megabytes
- Input file: standard input
- Output file: standard output

Note that the memory limit is unusual.

Let's define the sequence of Fibonacci strings as follows:  $f_0$  is 0,  $f_1$  is 1,  $f_i$  is  $f_{i-1} + f_{i-2}$  for  $i > 1$  (+ denotes the concatenation of two strings). So, for example,  $f_2$  is 10,  $f_3$  is 101,  $f_4$  is 10110.

For a given string  $s$ , let's define  $g(s)$  as the number of ways to cut it into several (any number, possibly even just one) strings such that none of these strings are Fibonacci strings. For example, if  $s$  is 10110101,  $g(s) = 3$  since there are three ways to cut it:

101101 + 01;

1011 + 0101;

1011 + 01 + 01.

You are given a sequence of strings  $s_1, s_2, \dots, s_n$ . Calculate  $g(s_1), g(s_1 + s_2), \dots, g(s_1 + s_2 + \dots + s_n)$ . Since these values can be huge, print them modulo 998244353.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 998244353;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
8     if (x < 0) {
9         x += P;
10    }
11    if (x >= P) {
12        x -= P;
13    }
14    return x;
15 }
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25 }
26 # struct Z;
27 vector<int> fib{1, 2};
28 int get(int n) {
29     if (n == 0) {
30         return 1;
31     }
32     if (n == 1) {
33         return 0;
34     }
35     auto it = prev(upper_bound(fib.begin(), fib.end(), n));
36     return get(n - *it);
37 }
38 int main() {
39     ios::sync_with_stdio(false);
40     cin.tie(nullptr);
41     for (int i = 0; i < 30; i++) {
42         fib.push_back(fib[fib.size() - 2] + fib.back());
43     }
44     int n;
45     cin >> n;
46     Z dp = 1, sum = 1;
47     vector<pair<int, Z>> suf;
48     auto add = [&](int x) {
49         Z ndp = sum - dp;
50         vector<pair<int, Z>> nsuf;
51         if (x) {
52             nsuf.push_back({1, dp});
53         }
54         for (auto [l, v] : suf) {
55             if (x == get(l)) {
56                 nsuf.emplace_back(l + 1, v);
57                 if (binary_search(fib.begin(), fib.end(), l + 1)) {
58                     ndp -= v;
59                 }
60             }
61         }
62     };
63     swap(suf, nsuf);

```

```

63         dp = ndp;
64         sum += ndp;
65     };
66     for (int i = 0; i < n; i++) {
67         string s;
68         cin >> s;
69         for (auto c : s) {
70             add(c - '0');
71         }
72         cout << dp << "\n";
73     }
74     return 0;
75 }
```

## data structures

### 802: Palindromic Problem

- Time limit: 5 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given a string  $s$  of length  $n$ , consisting of lowercase Latin letters.

You are allowed to replace at most one character in the string with an arbitrary lowercase Latin letter.

Print the lexicographically minimal string that can be obtained from the original string and contains the maximum number of palindromes as substrings. Note that if a palindrome appears more than once as a substring, it is counted the same number of times it appears.

The string  $a$  is lexicographically smaller than the string  $b$  if and only if one of the following holds:

$a$  is a prefix of  $b$ , but  $a \neq b$ ;

in the first position where  $a$  and  $b$  are different, the string  $a$  contains a letter that appears earlier in the alphabet than the corresponding letter in  $b$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct SuffixArray;
5 template<class T,
6         class Cmp = less<T>>
7 # struct RMQ;
8 int main() {
```

```

9     ios::sync_with_stdio(false);
10    cin.tie(nullptr);
11    int n;
12    cin >> n;
13    string s;
14    cin >> s;
15    s = s + '0' + string(s.rbegin(), s.rend());
16    SuffixArray sa(s);
17    RMQ<int> rmq(sa.lc);
18    auto lcp = [&](int i, int j) {
19        i = sa.rk[i];
20        j = sa.rk[j];
21        if (i > j) {
22            swap(i, j);
23        }
24        assert(i != j);
25        return rmq(i, j);
26    };
27    i64 res = 0;
28    vector<i64> d1(n), d0(n);
29    vector<array<i64, 26>> add(n);
30    auto change = [&](int l, int r, int k, int b) {
31        if (l < n) {
32            d1[l] += k;
33            d0[l] += b;
34        }
35        if (r < n) {
36            d1[r] -= k;
37            d0[r] -= b;
38        }
39    };
40    for (int i = 0; i < n; i++) {
41        int len = lcp(i, 2 * n - i);
42        res += len;
43        if (i + len < n && i - len >= 0) {
44            int more = 1;
45            if (i + len + 1 < n && i - len - 1 >= 0) {
46                more += lcp(i + len + 1, 2 * n - (i - len - 1));
47            }
48            add[i + len][s[i - len] - 'a'] += more;
49            add[i - len][s[i + len] - 'a'] += more;
50        }
51        change(i - len + 1, i, 1, -(i - len));
52        change(i + 1, i + len, -1, i + len);
53    }
54    for (int i = 1; i < n; i++) {
55        int len = lcp(i, 2 * n - (i - 1));
56        res += len;
57        if (i + len < n && i - 1 - len >= 0) {
58            int more = 1;
59            if (i + len + 1 < n && i - 1 - len - 1 >= 0) {
60                more += lcp(i + len + 1, 2 * n - (i - 1 - len - 1));
61            }
62            add[i + len][s[i - 1 - len] - 'a'] += more;
63            add[i - 1 - len][s[i + len] - 'a'] += more;
64        }
65        change(i - len, i, 1, -(i - len - 1));
66        change(i, i + len, -1, i + len);
67    }
68    for (int i = 1; i < n; i++) {
69        d0[i] += d0[i - 1];
70        d1[i] += d1[i - 1];
71    }
72    vector<i64> f(n);

```

```

73     for (int i = 0; i < n; i++) {
74         f[i] = d0[i] + d1[i] * i;
75     }
76     i64 ans = -1;
77     int x = -1, y = -1;
78     for (int i = 0; i < n; i++) {
79         for (int j = 0; j < s[i] - 'a'; j++) {
80             i64 val = res + add[i][j] - f[i];
81             if (val > ans) {
82                 ans = val;
83                 x = i;
84                 y = j;
85             }
86         }
87     }
88     if (res > ans) {
89         ans = res;
90         x = -1;
91         y = -1;
92     }
93     for (int i = n - 1; i >= 0; i--) {
94         for (int j = s[i] - 'a' + 1; j < 26; j++) {
95             i64 val = res + add[i][j] - f[i];
96             if (val > ans) {
97                 ans = val;
98                 x = i;
99                 y = j;
100            }
101        }
102    }
103    cout << ans << "\n";
104    if (x != -1) {
105        s[x] = 'a' + y;
106    }
107    s.resize(n);
108    cout << s << "\n";
109    return 0;
110 }
```

### 803: Local Deletions

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

For an array  $b_1, b_2, \dots, b_m$ , for some  $i$  ( $1 < i < m$ ), element  $b_i$  is said to be a local minimum if  $b_i < b_{i-1}$  and  $b_i < b_{i+1}$ . Element  $b_1$  is said to be a local minimum if  $b_1 < b_2$ . Element  $b_m$  is said to be a local minimum if  $b_m < b_{m-1}$ .

For an array  $b_1, b_2, \dots, b_m$ , for some  $i$  ( $1 < i < m$ ), element  $b_i$  is said to be a local maximum if  $b_i > b_{i-1}$  and  $b_i > b_{i+1}$ . Element  $b_1$  is said to be a local maximum if  $b_1 > b_2$ . Element  $b_m$  is said to be a local maximum if  $b_m > b_{m-1}$ .

Let  $x$  be an array of distinct elements. We define two operations on it:

1 - delete all elements from  $x$  that are not local minima.

2 - delete all elements from  $x$  that are not local maxima.

Define  $f(x)$  as follows. Repeat operations 1, 2, 1, 2, ... in that order until you get only one element left in the array. Return that element.

For example, take an array  $[1, 3, 2]$ . We will first do type 1 operation and get  $[1, 2]$ . Then we will perform type 2 operation and get  $[2]$ . Therefore,  $f([1, 3, 2]) = 2$ .

You are given a permutation<sup>†</sup>  $a$  of size  $n$  and  $q$  queries. Each query consists of two integers  $l$  and  $r$  such that  $1 \leq l \leq r \leq n$ . The query asks you to compute  $f([a_l, a_{l+1}, \dots, a_r])$ .

<sup>†</sup> A permutation of length  $n$  is an array of  $n$  distinct integers from 1 to  $n$  in arbitrary order. For example,  $[2, 3, 1, 5, 4]$  is a permutation, but  $[1, 2, 2]$  is not a permutation (2 appears twice in the array), and  $[1, 3, 4]$  is also not a permutation ( $n = 3$ , but there is 4 in the array).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, q;
8     cin >> n >> q;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12        a[i]--;
13    }
14    vector<vector<int>> f{a};
15    while (f.back().size() > 1) {
16        auto v = f.back();
17        vector<int> u;
18        int t = f.size() % 2;
19        for (int i = 0; i < v.size(); i++) {
20            if ((i == 0 || (v[i] < v[i - 1]) == t) && (i == v.size() - 1 || (v[i] < v[i + 1]) == t)) {
21                u.push_back(v[i]);
22            }
23        }
24        f.push_back(u);
25    }
26    int R = f.size();
27    vector pos(R, vector<int>(n, -1));
28    for (int i = 0; i < R; i++) {
29        for (int j = 0; j < f[i].size(); j++) {
30            pos[i][f[i][j]] = j;
31        }
32    }
33    while (q--) {
34        int l, r;
35        cin >> l >> r;
36        l--, r--;
37        int t = 1;

```

```

38     int i = 0;
39     int al = -1, ar = -1;
40     while ((r - l + 1) + (al != -1) + (ar != -1) > 1) {
41         if (r < l) {
42             if ((al < ar) == t) {
43                 ar = -1;
44             } else {
45                 al = -1;
46             }
47         } else if (l == r) {
48             if ((al == -1 || (f[i][l] < al) == t) && (ar == -1 || (f[i][r] < ar) == t))
49                 {
50                     al = ar = -1;
51                 } else {
52                     if (al != -1 && (f[i][l] < al) == t) {
53                         al = -1;
54                     }
55                     if (ar != -1 && (f[i][r] < ar) == t) {
56                         ar = -1;
57                     }
58                     r = l - 1;
59                 }
60             } else {
61                 int ni = min(R - 1, i + 1);
62                 if ((al == -1 || (f[i][l] < al) == t) && (f[i][l] < f[i][l + 1]) == t) {
63                     al = f[i][l];
64                 } else if (al != -1 && (f[i][l] < al) == t) {
65                     al = -1;
66                 }
67                 if ((ar == -1 || (f[i][r] < ar) == t) && (f[i][r] < f[i][r - 1]) == t) {
68                     ar = f[i][r];
69                 } else if (ar != -1 && (f[i][r] < ar) == t) {
70                     ar = -1;
71                 }
72                 l += 1;
73                 r -= 1;
74                 int nl, nr;
75                 if (l > r) {
76                     nl = l;
77                     nr = l - 1;
78                 } else {
79                     nl = lower_bound(f[ni].begin(), f[ni].end(), f[i][l],
80                                     [&](int x, int y) {
81                         return pos[i][x] < pos[i][y];
82                     }) - f[ni].begin();
83                     nr = upper_bound(f[ni].begin(), f[ni].end(), f[i][r],
84                                     [&](int x, int y) {
85                         return pos[i][x] < pos[i][y];
86                     }) - f[ni].begin() - 1;
87                     assert(nr >= nl - 1);
88                 }
89                 l = nl;
90                 r = nr;
91                 i = ni;
92             }
93             t ^= 1;
94         }
95         int ans;
96         if (l == r) {
97             ans = f[i][l];
98         } else if (al != -1) {
99             ans = al;
100        } else {
101            ans = ar;
102        }
103    }
104 }

```

```

101         }
102         cout << ans + 1 << "\n";
103     }
104     return 0;
105 }
```

## 804: Leha and security system

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Bankopolis, the city you already know, finally got a new bank opened! Unfortunately, its security system is not yet working fine... Meanwhile hacker Leha arrived in Bankopolis and decided to test the system!

Bank has  $n$  cells for clients' money. A sequence from  $n$  numbers  $a_1, a_2, \dots, a_n$  describes the amount of money each client has. Leha wants to make requests to the database of the bank, finding out the total amount of money on some subsegments of the sequence and changing values of the sequence on some subsegments. Using a bug in the system, Leha can request two types of queries to the database:

$l \ l \ r \ y$  denoting that Leha changes each digit  $x$  to digit  $y$  in each element of sequence  $a_i$ , for which  $l \leq i \leq r$  holds. For example, if we change in number 11984381 digit 8 to 4, we get 11944341. It's worth noting that Leha, in order to stay in the shadow, never changes digits in the database to 0, i.e.  $y \neq 0$ .

$l \ r$  denoting that Leha asks to calculate and print the sum of such elements of sequence  $a_i$ , for which  $l \leq i \leq r$  holds.

As Leha is a white-hat hacker, he doesn't want to test this vulnerability on a real database. You are to write a similar database for Leha to test.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class Info, class Tag>
5 # struct LazySegmentTree;
6 # struct Tag;
7 # struct Info;
8 Info operator+(const Info &a, const Info &b) {
9     Info c;
10    for (int i = 0; i < 10; i++) {
11        c.sum[i] = a.sum[i] + b.sum[i];
12    }
13    return c;
```

```

14 }
15 int main() {
16     ios::sync_with_stdio(false);
17     cin.tie(nullptr);
18     int n, q;
19     cin >> n >> q;
20     vector<int> a(n);
21     vector<Info> init(n);
22     for (int i = 0; i < n; i++) {
23         cin >> a[i];
24         int x = a[i];
25         int p = 1;
26         while (x > 0) {
27             init[i].sum[x % 10] += p;
28             p *= 10;
29             x /= 10;
30         }
31     }
32     LazySegmentTree<Info, Tag> seg(init);
33     while (q--) {
34         int o;
35         cin >> o;
36         if (o == 1) {
37             int l, r, x, y;
38             cin >> l >> r >> x >> y;
39             l--;
40             auto tag = Tag();
41             tag.a[x] = y;
42             seg.rangeApply(l, r, tag);
43         } else {
44             int l, r;
45             cin >> l >> r;
46             l--;
47             auto res = seg.rangeQuery(l, r);
48             i64 ans = 0;
49             for (int i = 0; i < 10; i++) {
50                 ans += res.sum[i] * i;
51             }
52             cout << ans << "\n";
53         }
54     }
55     return 0;
56 }
```

## 805: BRT Contract

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In the last war of PMP, he defeated all his opponents and advanced to the final round. But after the end of semi-final round evil attacked him from behind and killed him! God bless him.

Before his death, PMP signed a contract with the bus rapid transit (BRT) that improves public transports by optimizing time of travel estimation. You should help PMP finish his last contract.

Each BRT line is straight line that passes  $n$  intersections on its ways. At each intersection there is traffic light that periodically cycles between green and red. It starts illuminating green at time zero. During the green phase which lasts for  $g$  seconds, traffic is allowed to proceed. After the green phase the light changes to red and remains in this color for  $r$  seconds. During the red phase traffic is prohibited from proceeding. If a vehicle reaches the intersection exactly at a time when the light changes to red, it should stop, but the vehicle is clear to proceed if the light has just changed to green.

All traffic lights have the same timing and are synchronized. In other words the period of red (and green) phase is the same for all of traffic lights and they all start illuminating green at time zero.

The BRT Company has calculated the time that a bus requires to pass each road segment. A road segment is the distance between two consecutive traffic lights or between a traffic light and source (or destination) station. More precisely BRT specialists provide  $n + 1$  positive integers  $l_i$ , the time in seconds that a bus needs to traverse  $i$ -th road segment in the path from source to destination. The  $l_1$  value denotes the time that a bus needs to pass the distance between source and the first intersection. The  $l_{n+1}$  value denotes the time between the last intersection and destination.

In one day  $q$  buses leave the source station. The  $i$ -th bus starts from source at time  $t_i$  (in seconds). Decision makers of BRT Company want to know what time a bus gets to destination?

The bus is considered as point. A bus will always move if it can. The buses do not interfere with each other.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, g, r;
8     cin >> n >> g >> r;
9     n++;
10    vector<int> l(n);
11    for (int i = 0; i < n; i++) {
12        cin >> l[i];
13    }
14    int q;
15    cin >> q;
16    vector<i64> ans(n + q), t(n + q);
17    map<i64, int> s;
18    int tot = q;
19    vector<int> p(n + q, -1);
20    vector<int> a;
21    const i64 T = g + r;
22    for (int i = 0; i < q; i++) {
23        cin >> t[i];
24        t[i] += l[0];
25        if (!s.count(t[i] % T)) {
26            s[t[i] % T] = i;

```

```

27         } else {
28             p[i] = s[t[i] % T];
29             a.push_back(i);
30         }
31     }
32     i64 sum = 0;
33     for (int i = 1; i < n; i++) {
34         i64 L = (g - sum % T + T) % T;
35         i64 R = (-sum % T + T) % T;
36         if (L < R) {
37             if (!s.count(R)) {
38                 s[R] = tot++;
39             }
40             auto it = s.lower_bound(L);
41             while (it != s.end() && it->first < R) {
42                 auto r = next(it);
43                 p[it->second] = s[R];
44                 ans[it->second] += R - it->first;
45                 a.push_back(it->second);
46                 it = s.erase(it);
47             }
48         } else {
49             if (!s.count(R)) {
50                 s[R] = tot++;
51             }
52             auto it = s.lower_bound(L);
53             while (it != s.end()) {
54                 auto r = next(it);
55                 p[it->second] = s[R];
56                 ans[it->second] += R - it->first + T;
57                 a.push_back(it->second);
58                 it = s.erase(it);
59             }
60             it = s.begin();
61             while (it != s.end() && it->first < R) {
62                 auto r = next(it);
63                 p[it->second] = s[R];
64                 ans[it->second] += R - it->first;
65                 a.push_back(it->second);
66                 it = s.erase(it);
67             }
68         }
69         sum += l[i];
70     }
71     reverse(a.begin(), a.end());
72     for (auto i : a) {
73         ans[i] += ans[p[i]];
74     }
75     for (int i = 0; i < q; i++) {
76         ans[i] += t[i] + sum;
77         cout << ans[i] << "\n";
78     }
79     return 0;
80 }

```

## 806: Last Man Standing

- Time limit: 7 seconds
- Memory limit: 512 megabytes
- Input file: standard input

- Output file: standard output

There are  $n$  heroes in a videogame. Each hero has some health value  $h$  and initial armor value  $a$ . Let the current value of armor be  $a_{cur}$ , initially equal to  $a$ .

When  $x$  points of damage are inflicted on a hero, the following happens: if  $x < a_{cur}$ , then  $x$  gets subtracted from  $a_{cur}$ ; otherwise, 1 gets subtracted from  $h$  and  $a_{cur}$  gets assigned back to  $a$ .

In the start of the game, you choose the value  $x$  (an integer strictly greater than 0, arbitrarily large). Then you keep attacking all heroes in rounds: in one round, you inflict  $x$  points of damage to all alive heroes. A hero dies when his health becomes 0. The game ends when all heroes are dead.

The last hero to die earns the number of points, equal to the number of rounds he was the only hero alive. The other heroes get 0 points. In particular, if the last round ends with multiple heroes dying, then every hero gets 0 points.

The game is played for every possible  $x$  (from 1 to infinity). The points are reset between the games. What's the maximum number of points each hero has had?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int V = 2E5;
5 void solve() {
6     int n;
7     cin >> n;
8     vector<int> h(n), a(n);
9     for (int i = 0; i < n; i++) {
10         cin >> h[i];
11     }
12     for (int i = 0; i < n; i++) {
13         cin >> a[i];
14     }
15     map<pair<int, int>, int> id;
16     for (int i = 0; i < n; i++) {
17         id[{h[i], a[i]}] = i;
18     }
19     vector<array<pair<int, int>, 2>> f(V + 1);
20     for (int i = 0; i < n; i++) {
21         auto v = make_pair(h[i], a[i]);
22         for (auto &y : f[a[i]]) {
23             if (v > y) {
24                 swap(v, y);
25             }
26         }
27     }
28     for (int i = V - 1; i >= 0; i--) {
29         for (auto v : f[i + 1]) {
30             for (auto &y : f[i]) {
31                 if (v > y) {

```

```

32             swap(v, y);
33         }
34     }
35 }
36 vector<i64> ans(n);
37 for (int x = 1; x <= V; x++) {
38     i64 mx = 0;
39     int l = 0;
40     for (int i = 1; i <= V; i += x) {
41         i64 res = 1LL * (i + x - 1) / x * f[i][0].first;
42         if (res > mx) {
43             mx = res;
44             l = i;
45         }
46     }
47     auto v = f[l][0];
48     i64 sec = 0;
49     for (int i = 1; i <= V; i += x) {
50         i64 res = 1LL * (i + x - 1) / x * f[i][i <= l && f[i][0] == v].first;
51         if (res > sec) {
52             sec = res;
53         }
54     }
55     if (sec < mx) {
56         int i = id[v];
57         ans[i] = max(ans[i], mx - sec);
58     }
59 }
60 for (int i = 0; i < n; i++) {
61     cout << ans[i] << " \n"[i == n - 1];
62 }
63 }
64 int main() {
65     ios::sync_with_stdio(false);
66     cin.tie(nullptr);
67     int t;
68     cin >> t;
69     while (t--) {
70         solve();
71     }
72     return 0;
73 }
74 }
```

**807: ...Wait for it...**

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Barney is searching for his dream girl. He lives in NYC. NYC has  $n$  junctions numbered from 1 to  $n$  and  $n - 1$  roads connecting them. We will consider the NYC as a rooted tree with root being junction 1.  $m$  girls live in NYC,  $i$ -th of them lives along junction  $c_i$  and her weight initially equals  $i$  pounds.

Barney consider a girl  $x$  to be better than a girl  $y$  if and only if: girl  $x$  has weight strictly less than girl  $y$

or girl x and girl y have equal weights and index of girl x living junction index is strictly less than girl y living junction index, i.e.  $cx < cy$ . Thus for any two girls one of them is always better than another one.

For the next q days, one event happens each day. There are two types of events:

Barney goes from junction v to junction u. As a result he picks at most k best girls he still have not invited from junctions on his way and invites them to his house to test if one of them is his dream girl. If there are less than k not invited girls on his path, he invites all of them.

Girls living along junctions in subtree of junction v (including v itself) put on some weight. As result, their weights increase by k pounds.

Your task is for each event of first type tell Barney the indices of girls he will invite to his home in this event.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct HLD;
5 template<class Info, class Tag>
6 # struct LazySegmentTree;
7 constexpr i64 inf = 1E18;
8 # struct Tag;
9 # struct Info;
10 Info operator+(Info a, Info b) {
11     if (a.x < b.x || (a.x == b.x && a.y < b.y)) {
12         return a;
13     } else {
14         return b;
15     }
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20     int n, m, q;
21     cin >> n >> m >> q;
22     vector<vector<int>> adj(n);
23     HLD t(n);
24     for (int i = 1; i < n; i++) {
25         int u, v;
26         cin >> u >> v;
27         u--, v--;
28         t.addEdge(u, v);
29     }
30     vector<int> c(m);
31     for (int i = 0; i < m; i++) {
32         cin >> c[i];
33         c[i]--;
34     }
35     vector<vector<int>> p(n);
36     for (int i = m - 1; i >= 0; i--) {
37         p[c[i]].push_back(i);
38     }

```

```

39     t.work();
40     LazySegmentTree<Info, Tag> seg(n);
41     for (int i = 0; i < n; i++) {
42         if (!p[i].empty()) {
43             seg.modify(t.in[i], {p[i].back(), p[i].back()});
44         }
45     }
46     while (q--) {
47         int o;
48         cin >> o;
49         if (o == 1) {
50             int u, v, k;
51             cin >> u >> v >> k;
52             u--;
53             v--;
54             vector<int> ans;
55             while (k > 0) {
56                 int x = u, y = v;
57                 Info info;
58                 while (t.top[x] != t.top[y]) {
59                     if (t.dep[t.top[x]] < t.dep[t.top[y]]) {
60                         swap(x, y);
61                     }
62                     info = info + seg.rangeQuery(t.in[t.top[x]], t.in[x] + 1);
63                     x = t.parent[t.top[x]];
64                 }
65                 if (t.dep[x] > t.dep[y]) {
66                     swap(x, y);
67                 }
68                 info = info + seg.rangeQuery(t.in[x], t.in[y] + 1);
69                 if (info.y == -1) {
70                     break;
71                 }
72                 ans.push_back(info.y + 1);
73                 x = c[info.y];
74                 p[x].pop_back();
75                 if (!p[x].empty()) {
76                     seg.modify(t.in[x], {info.x - info.y + p[x].back(), p[x].back()});
77                 } else {
78                     seg.modify(t.in[x], {inf, -1});
79                 }
80                 k--;
81             }
82             cout << ans.size();
83             for (auto x : ans) {
84                 cout << " " << x;
85             }
86             cout << "\n";
87         } else {
88             int v, k;
89             cin >> v >> k;
90             v--;
91             seg.rangeApply(t.in[v], t.out[v], {k});
92         }
93     }
94 }
```

**808: XOR Partition**

- Time limit: 7 seconds

- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

For a set of integers  $S$ , let's define its cost as the minimum value of  $x \oplus y$  among all pairs of different integers from the set (here,  $\oplus$  denotes bitwise XOR). If there are less than two elements in the set, its cost is equal to  $2^{30}$ .

You are given a set of integers  $\{a_1, a_2, \dots, a_n\}$ . You have to partition it into two sets  $S_1$  and  $S_2$  in such a way that every element of the given set belongs to exactly one of these two sets. The value of the partition is the minimum among the costs of  $S_1$  and  $S_2$ .

Find the partition with the maximum possible value.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n;
9     cin >> n;
10    vector<pair<int, int>> a(n);
11    for (int i = 0; i < n; i++) {
12        cin >> a[i].first;
13        a[i].second = i;
14    }
15    sort(a.begin(), a.end());
16    vector<array<int, 3>> edges;
17    for (int i = 0; i < n; i++) {
18        for (int j = i + 1; j < n && j <= i + 3; j++) {
19            edges.push_back({a[i].first ^ a[j].first, a[i].second, a[j].second});
20        }
21    }
22    sort(edges.begin(), edges.end());
23    DSU dsu(n);
24    vector<vector<int>> adj(n);
25    for (auto [_, i, j] : edges) {
26        if (dsu.merge(i, j)) {
27            adj[i].push_back(j);
28            adj[j].push_back(i);
29        }
30    }
31    vector<int> ans(n);
32    auto dfs = [&](auto self, int x, int p) -> void {
33        for (auto y : adj[x]) {
34            if (y == p) {
35                continue;
36            }
37            ans[y] = ans[x] ^ 1;
38            self(self, y, x);

```

```

39         }
40     };
41     dfs(dfs, 0, -1);
42     for (int i = 0; i < n; i++) {
43         cout << ans[i];
44     }
45     cout << "\n";
46     return 0;
47 }
```

## 809: Two Centroids

- Time limit: 1.5 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

You are given a tree (an undirected connected acyclic graph) which initially only contains vertex 1. There will be several queries to the given tree. In the  $i$ -th query, vertex  $i + 1$  will appear and be connected to vertex  $p_i$  ( $1 \leq p_i \leq i$ ).

After each query, please find out the least number of operations required to make the current tree has two centroids. In one operation, you can add one vertex and one edge to the tree such that it remains a tree.

A vertex is called a centroid if its removal splits the tree into subtrees with at most  $\lfloor \frac{n}{2} \rfloor$  vertices each, with  $n$  as the number of vertices of the tree. For example, the centroid of the following tree is 3 because the biggest subtree after removing the centroid has 2 vertices.

In the next tree, vertex 1 and 2 are both centroids.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct HLD;
5 template <typename T>
6 # struct Fenwick;
7 void solve() {
8     int n;
9     cin >> n;
10    vector<int> p(n);
11    HLD t(n);
12    for (int i = 1; i < n; i++) {
13        cin >> p[i];
14        p[i]--;
15        t.addEdge(p[i], i);
16    }
17    t.work();
```

```

18     Fenwick<int> fen(n);
19     fen.add(0, 1);
20     int x = 1;
21     for (int i = 1; i < n; i++) {
22         fen.add(t.in[i], 1);
23         if (i == 1) {
24             cout << 0;
25         } else {
26             int s = fen.rangeSum(t.in[x], t.out[x]);
27             int ans = abs(2 * s - (i + 1));
28             if (t.isAncestor(x, i)) {
29                 int y = t.rootedParent(i, x);
30                 int s = fen.rangeSum(t.in[y], t.out[y]);
31                 int val = abs(2 * s - (i + 1));
32                 if (val < ans) {
33                     ans = val;
34                     x = y;
35                 }
36             } else {
37                 int y = t.parent[x];
38                 if (y) {
39                     int s = fen.rangeSum(t.in[y], t.out[y]);
40                     int val = abs(2 * s - (i + 1));
41                     if (val < ans) {
42                         ans = val;
43                         x = y;
44                     }
45                 }
46                 if (t.isAncestor(y, i)) {
47                     y = t.rootedParent(i, y);
48                     int s = fen.rangeSum(t.in[y], t.out[y]);
49                     int val = abs(2 * s - (i + 1));
50                     if (val < ans) {
51                         ans = val;
52                         x = y;
53                     }
54                 }
55             }
56             cout << ans;
57         }
58         cout << " \n"[i == n - 1];
59     }
60 }
61 int main() {
62     ios::sync_with_stdio(false);
63     cin.tie(nullptr);
64     int t;
65     cin >> t;
66     while (t--) {
67         solve();
68     }
69     return 0;
70 }
```

## 810: LuoTianyi and the Function

- Time limit: 7 seconds
- Memory limit: 1024 megabytes
- Input file: standard input

- Output file: standard output

LuoTianyi gives you an array  $a$  of  $n$  integers and the index begins from 1.

Define  $g(i, j)$  as follows:

$g(i, j)$  is the largest integer  $x$  that satisfies  $\{a_p : i \leq p \leq j\} \subseteq \{a_q : x \leq q \leq j\}$  while  $i \leq j$ ;

and  $g(i, j) = 0$  while  $i > j$ .

There are  $q$  queries. For each query you are given four integers  $l, r, x, y$ , you need to calculate  $\sum_{i=l}^r \sum_{j=x}^y g(i, j)$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class Info, class Tag>
5 # struct LazySegmentTree;
6 # struct Tag;
7 # struct Info;
8 Info operator+(Info a, Info b) {
9     Info c;
10    c.len = a.len + b.len;
11    c.sum = a.sum + b.sum;
12    c.hsum = a.hsum + b.hsum;
13    return c;
14 }
15 int main() {
16     ios::sync_with_stdio(false);
17     cin.tie(nullptr);
18     int n, q;
19     cin >> n >> q;
20     vector<int> lst(n, -1), prev(n);
21     vector<int> a(n);
22     for (int i = 0; i < n; i++) {
23         cin >> a[i];
24         a[i]--;
25     }
26     for (int i = 0; i < n; i++) {
27         prev[i] = lst[a[i]];
28         lst[a[i]] = i;
29     }
30     vector<i64> ans(q);
31     vector<vector<tuple<int, int, int, int>>> qry(n);
32     for (int i = 0; i < q; i++) {
33         int l, r, x, y;
34         cin >> l >> r >> x >> y;
35         l--;
36         x--;
37         qry[x].emplace_back(l, r, i, 1);
38         if (y < n) {
39             qry[y].emplace_back(l, r, i, -1);
40         }
41     }
42     LazySegmentTree<Info, Tag> seg(n, Info{1, 0, 0});
43     set<int> s{-1};
44     auto add = [&](int x) {

```

```

44         auto it = s.insert(x).first;
45         seg.rangeApply(*prev(it) + 1, x + 1, {0, 0, x + 1});
46     };
47     for (int i = 0; i < n; i++) {
48         if (lst[i] != -1) {
49             add(lst[i]);
50         }
51     }
52     for (int i = n - 1; i >= 0; i--) {
53         seg.rangeApply(0, n, {1, 0, -1});
54         for (auto [l, r, k, t] : qry[i]) {
55             ans[k] += t * seg.rangeQuery(l, r).hsum;
56         }
57         if (prev[i] != -1) {
58             add(prev[i]);
59         }
60         seg.rangeApply(i, i + 1, {0, 0, 0});
61     }
62     for (int i = 0; i < q; i++) {
63         cout << ans[i] << "\n";
64     }
65     return 0;
66 }

```

## 811: Cyclical Quest

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Some days ago, WJMZBMR learned how to answer the query “how many times does a string  $x$  occur in a string  $s$ ” quickly by preprocessing the string  $s$ . But now he wants to make it harder.

So he wants to ask “how many consecutive substrings of  $s$  are cyclical isomorphic to a given string  $x$ ”. You are given string  $s$  and  $n$  strings  $x_i$ , for each string  $x_i$  find, how many consecutive substrings of  $s$  are cyclical isomorphic to  $x_i$ .

Two strings are called cyclical isomorphic if one can rotate one string to get the other one. ‘Rotate’ here means ‘to take some consecutive chars (maybe none) from the beginning of a string and put them back at the end of the string in the same order’. For example, string “abcde” can be rotated to string “deabc”. We can take characters “abc” from the beginning and put them at the end of “de”.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;

```

```
4 # struct SAM;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     string s;
9     cin >> s;
10    SAM sam;
11    int n = s.size();
12    vector<int> p(n + 1);
13    p[0] = 1;
14    for (int i = 0; i < n; i++) {
15        p[i + 1] = sam.extend(p[i], s[i]);
16    }
17    vector<int> freq(sam.size());
18    for (int i = 1; i <= n; i++) {
19        freq[p[i]]++;
20    }
21    vector<vector<int>> adj(sam.size());
22    for (int i = 2; i < sam.size(); i++) {
23        adj[sam.link(i)].push_back(i);
24    }
25    function<void(int)> dfs = [&](int x) {
26        for (auto y : adj[x]) {
27            dfs(y);
28            freq[x] += freq[y];
29        }
30    };
31    dfs(1);
32    int q;
33    cin >> q;
34    while (q--) {
35        string x;
36        cin >> x;
37        int m = x.size();
38        vector<int> f(m + 1);
39        for (int i = 1, j = 0; i < m; i++) {
40            while (j && x[i] != x[j]) {
41                j = f[j];
42            }
43            j += (x[i] == x[j]);
44            f[i + 1] = j;
45        }
46        int T = m % (m - f[m]) ? m : m - f[m];
47        i64 ans = 0;
48        int p = 1, len = 0;
49        for (auto c : x) {
50            while (!sam.next(p, c)) {
51                p = sam.link(p);
52                len = sam.len(p);
53            }
54            p = sam.next(p, c);
55            len++;
56        }
57        for (int i = 0; i < T; i++) {
58            char c = x[i];
59            while (!sam.next(p, c)) {
60                p = sam.link(p);
61                len = sam.len(p);
62            }
63            p = sam.next(p, c);
64            len++;
65            while (len > m) {
66                len--;
67                if (len <= sam.len(sam.link(p))) {
```

```

68             p = sam.link(p);
69         }
70     }
71     if (len == m) {
72         ans += freq[p];
73     }
74 }
75 cout << ans << "\n";
76 }
77 return 0;
78 }
```

## 812: k-Maximum Subsequence Sum

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Consider integer sequence  $a_1, a_2, \dots, a_n$ . You should run queries of two types:

The query format is “0 i val”. In reply to this query you should make the following assignment:  $a_i = val$ .

The query format is “1 l r k”. In reply to this query you should print the maximum sum of at most  $k$  non-intersecting subsegments of sequence  $a_l, a_{l+1}, \dots, a_r$ . Formally, you should choose at most  $k$  pairs of integers  $(x_1, y_1), (x_2, y_2), \dots, (x_t, y_t)$  ( $l \leq x_1 \leq y_1 < x_2 \leq y_2 < \dots < x_t \leq y_t \leq r$ ;  $t \leq k$ ) such that the sum  $a_{x_1} + a_{x_1+1} + \dots + a_{y_1} + a_{x_2} + a_{x_2+1} + \dots + a_{y_2} + \dots + a_{x_t} + a_{x_t+1} + \dots + a_{y_t}$  is as large as possible. Note that you should choose at most  $k$  subsegments. Particularly, you can choose 0 subsegments. In this case the described sum considered equal to zero.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class Info, class Tag>
5 # struct LazySegmentTree;
6 # struct Tag;
7 # struct Info;
8 Info operator+(Info a, Info b) {
9     Info c;
10    for (int t = 0; t < 2; t++) {
11        c.sum[t] = a.sum[t] + b.sum[t];
12        c.pre[t] = max(a.pre[t], array{b.pre[t][0] + a.sum[t], b.pre[t][1]});
13        c.suf[t] = max(b.suf[t], array{a.suf[t][0] + b.sum[t], a.suf[t][1]});
14        c.ans[t] = max({a.ans[t], b.ans[t], array{a.suf[t][0] + b.pre[t][0], a.suf[t][1],
15            b.pre[t][1]}});
16    }
17 }
```

```

15     }
16     return c;
17 }
18 int main() {
19     ios::sync_with_stdio(false);
20     cin.tie(nullptr);
21     int n;
22     cin >> n;
23     vector<int> a(n);
24     for (int i = 0; i < n; i++) {
25         cin >> a[i];
26     }
27     LazySegmentTree<Info, Tag> seg(n);
28     for (int i = 0; i < n; i++) {
29         seg.modify(i, Info(i, a[i]));
30     }
31     int m;
32     cin >> m;
33     while (m--) {
34         int t;
35         cin >> t;
36         if (t == 0) {
37             int i, val;
38             cin >> i >> val;
39             i--;
40             seg.modify(i, Info(i, val));
41         } else {
42             int l, r, k;
43             cin >> l >> r >> k;
44             l--;
45             vector<pair<int, int>> a;
46             int ans = 0;
47             for (int i = 0; i < k; i++) {
48                 auto res = seg.rangeQuery(l, r).ans[0];
49                 if (res[0] <= 0) {
50                     break;
51                 }
52                 ans += res[0];
53                 a.emplace_back(res[1], res[2]);
54                 seg.rangeApply(res[1], res[2], {-1});
55             }
56             cout << ans << "\n";
57             for (auto [l, r] : a) {
58                 seg.rangeApply(l, r, {-1});
59             }
60         }
61     }
62     return 0;
63 }
```

### 813: Tricky and Clever Password

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In his very young years the hero of our story, king Copa, decided that his private data was hidden not

enough securely, what is unacceptable for the king. That's why he invented tricky and clever password (later he learned that his password is a palindrome of odd length), and coded all his data using it.

Copa is afraid to forget his password, so he decided to write it on a piece of paper. He is aware that it is insecure to keep password in such way, so he decided to cipher it the following way: he cut  $x$  characters from the start of his password and from the end of it ( $x$  can be 0, and  $2x$  is strictly less than the password length). He obtained 3 parts of the password. Let's call it prefix, middle and suffix correspondingly, both prefix and suffix having equal length and middle always having odd length. From these parts he made a string  $A + \text{prefix} + B + \text{middle} + C + \text{suffix}$ , where  $A, B$  and  $C$  are some (possibly empty) strings invented by Copa, and  $+$  means concatenation.

Many years have passed, and just yesterday the king Copa found the piece of paper where his ciphered password was written. The password, as well as the strings  $A, B$  and  $C$ , was completely forgotten by Copa, so he asks you to find a password of maximum possible length, which could be invented, ciphered and written by Copa.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct SAM;
5 vector<int> manacher(string s) {
6     int n = s.size();
7     vector<int> r(n);
8     for (int i = 0, j = 0; i < n; i++) {
9         if (2 * j - i >= 0 && j + r[j] > i) {
10             r[i] = min(r[2 * j - i], j + r[j] - i);
11         }
12         while (i - r[i] >= 0 && i + r[i] < n && s[i - r[i]] == s[i + r[i]]) {
13             r[i] += 1;
14         }
15         if (i + r[i] > j + r[j]) {
16             j = i;
17         }
18     }
19     return r;
20 }
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     string s;
25     cin >> s;
26     int n = s.size();
27     auto t = s;
28     reverse(s.begin(), s.end());
29     auto r = manacher(s);
30     int len = 0;
31     vector<pair<int, int>> ans;
32     SAM sam;
33     vector<int> f(n + 1);
34     f[0] = 1;

```

```

35     for (int i = 0; i < n; i++) {
36         f[i + 1] = sam.extend(f[i], t[i] - 'a');
37     }
38     vector<int> fst(sam.t.size(), n);
39     for (int i = 1; i <= n; i++) {
40         fst[f[i]] = i;
41     }
42     vector<vector<int>> adj(sam.t.size());
43     for (int i = 2; i < sam.t.size(); i++) {
44         adj[sam.t[i].link].push_back(i);
45     }
46     function<void(int)> dfs = [&](int x) {
47         for (auto y : adj[x]) {
48             dfs(y);
49             fst[x] = min(fst[x], fst[y]);
50         }
51     };
52     dfs(1);
53     vector<vector<int>> cand(n + 1);
54     for (int i = 0; i < n; i++) {
55         int L = i - (r[i] - 1);
56         int R = i + r[i];
57         if (R - L > len) {
58             len = R - L;
59             ans = {{n - R + 1, R - L}};
60         }
61         cand[L].push_back(R);
62     }
63     vector<int> stk;
64     vector<int> lastl(n + 1);
65     for (int i = 0, p = 1; i < n; i++) {
66         if (!sam.t[p].next[s[i] - 'a']) {
67             break;
68         }
69         p = sam.t[p].next[s[i] - 'a'];
70         lastl[i + 1] = n - fst[p];
71     }
72     for (int i = 0; i < n; i++) {
73         int L = i;
74         for (auto R : cand[L]) {
75             auto it = partition_point(stk.begin(), stk.end(), [&](int x) {
76                 return lastl[x] >= R;
77             });
78             if (it != stk.begin()) {
79                 int x = *(it - 1);
80                 if (x * 2 + R - L > len) {
81                     len = x * 2 + R - L;
82                     ans = {{n - lastl[x] - x + 1, x}, {n - R + 1, R - L}, {n - x + 1, x}};
83                 }
84             }
85         }
86         while (!stk.empty() && lastl[i + 1] > lastl[stk.back()]) {
87             stk.pop_back();
88         }
89         stk.push_back(i + 1);
90     }
91     cout << ans.size() << "\n";
92     for (auto [x, l] : ans) {
93         cout << x << " " << l << "\n";
94     }
95     return 0;
96 }

```

## 814: Li Hua and Path

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Li Hua has a tree of  $n$  vertices and  $n - 1$  edges. The vertices are numbered from 1 to  $n$ .

A pair of vertices  $(u, v)$  ( $u < v$ ) is considered cute if exactly one of the following two statements is true:

$u$  is the vertex with the minimum index among all vertices on the path  $(u, v)$ .

$v$  is the vertex with the maximum index among all vertices on the path  $(u, v)$ .

There will be  $m$  operations. In each operation, he decides an integer  $k_j$ , then inserts a vertex numbered  $n + j$  to the tree, connecting with the vertex numbered  $k_j$ .

He wants to calculate the number of cute pairs before operations and after each operation.

Suppose you were Li Hua, please solve this problem.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template <typename T>
5 # struct Fenwick;
6 # struct DSU;
7 # struct HLD;
8 int main() {
9     ios::sync_with_stdio(false);
10    cin.tie(nullptr);
11    int n;
12    cin >> n;
13    vector<vector<int>> adj(n);
14    for (int i = 1; i < n; i++) {
15        int u, v;
16        cin >> u >> v;
17        u--, v--;
18        adj[u].push_back(v);
19        adj[v].push_back(u);
20    }
21    DSU dsu(n);
22    HLD t1(n), t2(n);
23    for (int i = n - 1; i >= 0; i--) {
24        for (auto j : adj[i]) {
25            if (j > i) {
26                t1.addEdge(i, dsu.leader(j));
27                dsu.merge(i, j);
28            }
29        }
}

```

```

30     }
31     dsu.init(n);
32     for (int i = 0; i < n; i++) {
33         for (auto j : adj[i]) {
34             if (j < i) {
35                 t2.addEdge(i, dsu.leader(j));
36                 dsu.merge(i, j);
37             }
38         }
39     }
40     t1.work(0);
41     t2.work(n - 1);
42     i64 ans = 0;
43     Fenwick<int> fen(n);
44     for (int i = 0; i < n; i++) {
45         ans += t1.siz[i] - 1;
46         ans += t2.siz[i] - 1;
47     }
48     function<void(int)> dfs = [&](int x) {
49         ans -= 2 * fen.rangeSum(t2.in[x], t2.out[x]);
50         fen.add(t2.in[x], 1);
51         for (auto y : t1.adj[x]) {
52             dfs(y);
53         }
54         fen.add(t2.in[x], -1);
55     };
56     dfs(0);
57     auto dep = t1.dep;
58     cout << ans << "\n";
59     int m;
60     cin >> m;
61     for (int i = 0; i < m; i++) {
62         int x;
63         cin >> x;
64         x--;
65         dep.push_back(dep[x] + 1);
66         ans += n + i - dep[n + i];
67         cout << ans << "\n";
68     }
69     return 0;
70 }
```

## 815: Points

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Pete and Bob invented a new interesting game. Bob takes a sheet of paper and locates a Cartesian coordinate system on it as follows: point  $(0, 0)$  is located in the bottom-left corner,  $Ox$  axis is directed right,  $Oy$  axis is directed up. Pete gives Bob requests of three types:

$\text{add } x \ y$  - on the sheet of paper Bob marks a point with coordinates  $(x, y)$ . For each request of this type it's guaranteed that point  $(x, y)$  is not yet marked on Bob's sheet at the time of the request.

remove  $x$   $y$  - on the sheet of paper Bob erases the previously marked point with coordinates  $(x, y)$ . For each request of this type it's guaranteed that point  $(x, y)$  is already marked on Bob's sheet at the time of the request.

find  $x$   $y$  - on the sheet of paper Bob finds all the marked points, lying strictly above and strictly to the right of point  $(x, y)$ . Among these points Bob chooses the leftmost one, if it is not unique, he chooses the bottommost one, and gives its coordinates to Pete.

Bob managed to answer the requests, when they were 10, 100 or 1000, but when their amount grew up to 2.105, Bob failed to cope. Now he needs a program that will answer all Pete's requests. Help Bob, please!

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class Info,
5          class Merge = plus<Info>>
6 # struct SegmentTree;
7 # struct Info;
8 Info operator+(Info a, Info b) {
9     return {max(a.max, b.max)};
10 }
11 int main() {
12     ios::sync_with_stdio(false);
13     cin.tie(nullptr);
14     int n;
15     cin >> n;
16     vector<string> o(n);
17     vector<int> x(n), y(n);
18     for (int i = 0; i < n; i++) {
19         cin >> o[i] >> x[i] >> y[i];
20     }
21     auto v = x;
22     sort(v.begin(), v.end());
23     vector<set<int>> s(n, {-1});
24     SegmentTree<Info> seg(n);
25     for (int i = 0; i < n; i++) {
26         x[i] = lower_bound(v.begin(), v.end(), x[i]) - v.begin();
27         if (o[i] == "add") {
28             s[x[i]].insert(y[i]);
29             seg.modify(x[i], {*s[x[i]].rbegin()});
30         } else if (o[i] == "remove") {
31             s[x[i]].erase(y[i]);
32             seg.modify(x[i], {*s[x[i]].rbegin()});
33         } else {
34             int X = seg.findFirst(x[i] + 1, n, [&](auto a) {
35                 return a.max > y[i];
36             });
37             if (X == -1) {
38                 cout << -1 << "\n";
39                 continue;
40             }
41             cout << v[X] << " " << *s[X].upper_bound(y[i]) << "\n";
42         }
43     }
}

```

```

44     return 0;
45 }
```

## 816: Holes

- Time limit: 1 second
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

Little Petya likes to play a lot. Most of all he likes to play a game Holes. This is a game for one person with following rules:

There are N holes located in a single row and numbered from left to right with numbers from 1 to N. Each hole has its own power (hole number i has the power  $a_i$ ). If you throw a ball into hole  $i$  it will immediately jump to hole  $i + a_i$ , then it will jump out of it and so on. If there is no hole with such number, the ball will just jump out of the row. On each of the M moves the player can perform one of two actions:

Set the power of the hole  $a$  to value  $b$ .

Throw a ball into the hole  $a$  and count the number of jumps of a ball before it jump out of the row and also write down the number of the hole from which it jumped out just before leaving the row.

Petya is not good at math, so, as you have already guessed, you are to perform all computations.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int N, M;
8     cin >> N >> M;
9     constexpr int B = 400;
10    vector<int> a(N), b(N), c(N), d(N);
11    for (int i = 0; i < N; i++) {
12        cin >> a[i];
13        a[i] += i;
14    }
15    for (int i = N - 1; i >= 0; i--) {
16        if (a[i] >= N || a[i] / B != i / B) {
17            b[i] = i;
18            c[i] = 1;
19        } else {
```

```

20         b[i] = b[a[i]];
21         c[i] = 1 + c[a[i]];
22     }
23 }
24 while (M--) {
25     int t;
26     cin >> t;
27     if (t == 0) {
28         int x, y;
29         cin >> x >> y;
30         x--;
31         a[x] = x + y;
32         for (int i = x; i >= x / B * B; i--) {
33             if (a[i] >= N || a[i] / B != i / B) {
34                 b[i] = i;
35                 c[i] = 1;
36             } else {
37                 b[i] = b[a[i]];
38                 c[i] = 1 + c[a[i]];
39             }
40         }
41     } else {
42         int x;
43         cin >> x;
44         x--;
45         int y = -1, cnt = 0;
46         while (x < N) {
47             cnt += c[x];
48             y = b[x];
49             x = a[b[x]];
50         }
51         cout << y + 1 << " " << cnt << "\n";
52     }
53 }
54 return 0;
55 }
```

### 817: M-tree

- Time limit: 2.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A rooted tree is called good if every vertex of the tree either is a leaf (a vertex with no children) or has exactly  $m$  children.

For a good tree, each leaf  $u$  has a positive integer  $c_u$  written on it, and we define the value of the leaf as  $c_u + \text{dep}_u$ , where  $\text{dep}_u$  represents the number of edges of the path from vertex  $u$  to the root (also known as the depth of  $u$ ). The value of a good tree is the maximum value of all its leaves.

Now, you are given an array of  $n$  integers  $a_1, a_2, \dots, a_n$ , which are the integers that should be written on the leaves. You need to construct a good tree with  $n$  leaves and write the integers from the array  $a$  to all the leaves. Formally, you should assign each leaf  $u$  an index  $b_u$ , where  $b$  is a permutation of

length  $n$ , and represent that the integer written on leaf  $u$  is  $c_u = a_{b_u}$ . Under these constraints, you need to minimize the value of the good tree.

You have  $q$  queries. Each query gives you  $x, y$  and changes  $a_x$  to  $y$ , and after that, you should output the minimum value of a good tree based on the current array  $a$ .

A permutation of length  $n$  is an array consisting of  $n$  distinct integers from 1 to  $n$  in arbitrary order. For example,  $[2, 3, 1, 5, 4]$  is a permutation, but  $[1, 2, 2]$  is not a permutation (2 appears twice in the array), and  $[1, 3, 4]$  is also not a permutation ( $n = 3$  but there is 4 in the array).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m, q;
6     cin >> n >> m >> q;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    map<int, int> f;
12    f[0] = 0;
13    auto split = [&](int x) {
14        auto it = prev(f.upper_bound(x));
15        f[x] = it->second;
16    };
17    auto add = [&](int x) {
18        split(x);
19        auto it = f.find(x);
20        while (it->second == m - 1) {
21            it = f.erase(it);
22        }
23        if (it->first > x) {
24            f[x] = 0;
25        }
26        split(it->first + 1);
27        it->second += 1;
28    };
29    auto del = [&](int x) {
30        split(x);
31        auto it = f.find(x);
32        while (it->second == 0) {
33            it = f.erase(it);
34        }
35        if (it->first > x) {
36            f[x] = m - 1;
37        }
38        split(it->first + 1);
39        it->second -= 1;
40    };
41    for (int i = 0; i < n; i++) {
42        add(a[i]);
43    }
44    while (q--) {
45        int x, y;
46        cin >> x >> y;

```

```

47         x--;
48         del(a[x]);
49         add(a[x] = y);
50         del(0);
51         auto it = prev(f.end());
52         while (it != f.begin() && prev(it)->second == 0) {
53             it--;
54             f.erase(next(it));
55         }
56         cout << it->first << " \n"[q == 0];
57         add(0);
58     }
59 }
60 int main() {
61     ios::sync_with_stdio(false);
62     cin.tie(nullptr);
63     int t;
64     cin >> t;
65     while (t--) {
66         solve();
67     }
68     return 0;
69 }
```

### 818: Gasoline prices

- Time limit: 3.5 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

There is one gas station in each city of Berland. Gas stations have special pricing, and for each gas station there is a fixed range of prices for which they are ready to sell gasoline. A gas station in the city with the number  $i$  is ready to sell gasoline at any price from  $l_i$  to  $r_i$  inclusive.

The King of Berland - is an exemplary family man, and for  $m$  years, two sons were born to him every year. The king's children have been involved in public affairs since early childhood, and at the end of each year they check the honesty of gasoline prices. From birth, the king's children, who are born in the year  $i$ , are responsible for checking gasoline prices on the ways from the city of  $a_i$  to the city of  $b_i$  and from the city of  $c_i$  to the city of  $d_i$ , respectively.

The check is as follows: both children simultaneously start their journey from the cities  $a_i$  and  $c_i$ , respectively. The first son of the king, born in the year  $i$ , moves along the path from the city  $a_i$  to the city  $b_i$ , and the second - from the city  $c_i$  to the city  $d_i$ . Children check that the price of gasoline in the city of  $a_i$  coincides with the price of gasoline in the city of  $c_i$ . Next, they check that the price of gasoline in the second city on the way from  $a_i$  to  $b_i$  coincides with the price in the second city on the way from  $c_i$  to  $d_i$ . Then they repeat the same thing for a couple of third cities on their paths and so on. At the end, they check that the price of gasoline in the city of  $b_i$  coincides with the price of gasoline in the city

of  $d_i$ . It is guaranteed that the length of the path from the city  $a_i$  to the city  $b_i$  coincides with the length of the path from the city  $c_i$  to the city  $d_i$ .

Gas stations must strictly obey the laws, and therefore all checks of gasoline prices should not reveal violations. Help Berland gas stations find out how many ways they can set gasoline prices for  $m$  years. In other words, for each  $i$  from 1 to  $m$ , calculate how many ways you can set gasoline prices at all gas stations so that after the birth of the first  $i$  pairs of the king's children, all their checks did not reveal violations, and at any gas station the price was in the acceptable price range. Since the number of such methods can be large, calculate the answer modulo  $10^9 + 7$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 template<int P>
15 # struct MInt;
16 template<int V, int P>
17 constexpr MInt<P> CInv = MInt<P>(V).inv();
18 constexpr int P = 1000000007;
19 using Z = MInt<P>;
20 constexpr i64 M = i64(1E18) + 9;
21 # struct Fenwick;
22 # struct HLD;
23 i64 mul(i64 a, i64 b) {
24     i64 res = a * b - i64(1.L * a * b / M + .5L) * M;
25     res %= M;
26     if (res < 0) {
27         res += M;
28     }
29     return res;
30 }
31 i64 power(i64 a, i64 b) {
32     i64 res = 1;
33     for (; b; b /= 2, a = mul(a, a)) {
34         if (b % 2) {
35             res = mul(res, a);
36         }
37     }
38     return res;
39 }
40 int main() {
41     ios::sync_with_stdio(false);
42     cin.tie(nullptr);
43     int n;
44     cin >> n;
45     vector<int> p(n, -1);

```

```

46     HLD t(n);
47     for (int i = 1; i < n; i++) {
48         cin >> p[i];
49         p[i] -= 1;
50         t.addEdge(p[i], i);
51     }
52     t.work();
53     vector<int> l(n), r(n);
54     for (int i = 0; i < n; i++) {
55         cin >> l[i] >> r[i];
56         l[i] -= 1;
57     }
58     mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
59     const i64 B = 1000000000 + rng() % 1000000000;
60     vector<i64> pw(n + 1);
61     pw[0] = 1;
62     for (int i = 1; i <= n; i++) {
63         pw[i] = mul(pw[i - 1], B);
64     }
65     auto inv = power(B, M - 2);
66     vector<i64> invpw(n + 1);
67     invpw[0] = 1;
68     for (int i = 1; i <= n; i++) {
69         invpw[i] = mul(invpw[i - 1], inv);
70     }
71     int m;
72     cin >> m;
73     vector<int> bel(n);
74     vector<vector<int>> arr(n);
75     Z ans = 1;
76     for (int i = 0; i < n; i++) {
77         bel[i] = i;
78         arr[i] = {i};
79         ans *= (r[i] - l[i]);
80     }
81     Fenwick f1(n), f2(n);
82     for (int i = 0; i < n; i++) {
83         f1.add(t.in[i], mul(bel[i], pw[t.in[i]]));
84         f2.add(t.in[i], mul(bel[i], invpw[t.in[i]]));
85     }
86     auto getPath = [&](int a, int b) {
87         vector<array<int, 3>> l, r;
88         while (t.top[a] != t.top[b]) {
89             if (t.dep[t.top[a]] > t.dep[t.top[b]]) {
90                 l.push_back({t.in[t.top[a]], t.in[a] + 1, 0});
91                 a = t.parent[t.top[a]];
92             } else {
93                 r.push_back({t.in[t.top[b]], t.in[b] + 1, 1});
94                 b = t.parent[t.top[b]];
95             }
96         }
97         if (t.dep[a] < t.dep[b]) {
98             r.push_back({t.in[a], t.in[b] + 1, 1});
99         } else {
100             l.push_back({t.in[b], t.in[a] + 1, 0});
101         }
102         reverse(r.begin(), r.end());
103         l.insert(l.end(), r.begin(), r.end());
104         return l;
105     };
106     auto getHash = [&](int l, int r, int t, int len) {
107         // cerr << "query " << l << " " << r << " " << t << " " << len << "\n";
108         if (t == 0) {
109             l = r - len;

```

```

110         return mul(f1.rangeSum(l, r), invpw[l]);
111     } else {
112         r = l + len;
113         return mul(f2.rangeSum(l, r), pw[r - 1]);
114     }
115 }
116 auto merge = [&](int x, int y) {
117     x = bel[x];
118     y = bel[y];
119     // cerr << "(" << x << ", " << y << ")\n";
120     if (arr[x].size() < arr[y].size()) {
121         swap(x, y);
122     }
123     for (auto i : arr[y]) {
124         i64 dif = x - y;
125         if (dif < 0) {
126             dif += M;
127         }
128         f1.add(t.in[i], mul(dif, pw[t.in[i]]));
129         f2.add(t.in[i], mul(dif, invpw[t.in[i]]));
130         // f1.add(t.in[i], mul(x, pw[t.in[i]]));
131         // f2.add(t.in[i], mul(x, invpw[t.in[i]]));
132         // f1.add(t.in[i], mul(M - y, pw[t.in[i]]));
133         // f2.add(t.in[i], mul(M - y, invpw[t.in[i]]));
134         // cerr << i << " " << y << " => " << x << "\n";
135         bel[i] = x;
136     }
137     if (l[x] < r[x]) {
138         ans /= r[x] - l[x];
139     }
140     if (l[y] < r[y]) {
141         ans /= r[y] - l[y];
142     }
143     l[x] = max(l[x], l[y]);
144     r[x] = min(r[x], r[y]);
145     ans *= max(0, r[x] - l[x]);
146     arr[x].insert(arr[x].end(), arr[y].begin(), arr[y].end());
147     arr[y].clear();
148 };
149 for (int i = 0; i < m; i++) {
150     int a, b, c, d;
151     cin >> a >> b >> c >> d;
152     a--, b--, c--, d--;
153     auto u = getPath(a, b);
154     auto v = getPath(c, d);
155     int x = 0, y = 0;
156     while (x < u.size()) {
157         int len = min(u[x][1] - u[x][0], v[y][1] - v[y][0]);
158         auto hu = getHash(u[x][0], u[x][1], u[x][2], len);
159         auto hv = getHash(v[y][0], v[y][1], v[y][2], len);
160         if (hu != hv) {
161             // cerr << hu << " != " << hv << "\n";
162             int lo = 0, hi = len - 1;
163             while (lo < hi) {
164                 int m = (lo + hi + 1) / 2;
165                 auto hu = getHash(u[x][0], u[x][1], u[x][2], m);
166                 auto hv = getHash(v[y][0], v[y][1], v[y][2], m);
167                 if (hu == hv) {
168                     lo = m;
169                 } else {
170                     hi = m - 1;
171                 }
172             }
173             int idu, idv;

```

```

174         if (u[x][2] == 0) {
175             idu = u[x][1] - lo - 1;
176         } else {
177             idu = u[x][0] + lo;
178         }
179         if (v[y][2] == 0) {
180             idv = v[y][1] - lo - 1;
181         } else {
182             idv = v[y][0] + lo;
183         }
184         // cerr << "(" << t.seq[idu] << ", " << t.seq[idv] << ")\\n";
185         merge(t.seq[idu], t.seq[idv]);
186         len = lo;
187     }
188     if (u[x][2] == 0) {
189         u[x][1] -= len;
190     } else {
191         u[x][0] += len;
192     }
193     if (v[y][2] == 0) {
194         v[y][1] -= len;
195     } else {
196         v[y][0] += len;
197     }
198     x += (u[x][0] == u[x][1]);
199     y += (v[y][0] == v[y][1]);
200 }
201 cout << ans << "\\n";
202 }
203 return 0;
204 }
```

### 819: Magician and Pigs (Hard Version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. The only difference between the two versions is the constraint on  $n$  and  $x$ . You can make hacks only if both versions of the problem are solved.

Little09 has been interested in magic for a long time, and it's so lucky that he meets a magician! The magician will perform  $n$  operations, each of them is one of the following three:

1  $x$ : Create a pig with  $x$  Health Points.

2  $x$ : Reduce the Health Point of all living pigs by  $x$ .

3: Repeat all previous operations. Formally, assuming that this is the  $i$ -th operation in the operation sequence, perform the first  $i - 1$  operations (including “Repeat” operations involved) in turn.

A pig will die when its Health Point is less than or equal to 0.

Little09 wants to know how many living pigs there are after all the operations. Please, print the answer modulo 998 244 353.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 template<int P>
15 # struct MInt;
16 template<int V, int P>
17 constexpr MInt<P> CInv = MInt<P>(V).inv();
18 constexpr int P = 998244353;
19 using Z = MInt<P>;
20 constexpr int inf = 1E9;
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     int n;
25     cin >> n;
26     vector<array<int, 2>> op(n);
27     for (int i = 0; i < n; i++) {
28         int t;
29         cin >> t;
30         if (t == 1) {
31             int x;
32             cin >> x;
33             op[i] = {t, x};
34         } else if (t == 2) {
35             int x;
36             cin >> x;
37             op[i] = {t, x};
38         } else {
39             op[i] = {t};
40         }
41     }
42     vector<int> pre(n + 1);
43     for (int i = 0; i < n; i++) {
44         if (op[i][0] == 2) {
45             pre[i + 1] = min(inf, pre[i] + op[i][1]);
46         } else if (op[i][0] == 3) {
47             pre[i + 1] = min(inf, pre[i] * 2);
48         } else {
49             pre[i + 1] = pre[i];
50         }
51     }
52     int c0 = 0;
53     vector<int> a;
54     i64 sum = 0;
55     for (int i = 0; i < n; i++) {
56         if (op[i][0] == 3) {

```

```

57         if (pre[i] == 0) {
58             c0++;
59         } else {
60             a.push_back(i);
61         }
62     } else if (op[i][0] == 2) {
63         sum += op[i][1];
64     }
65 }
66 Z ans = 0;
67 for (int i = 0; i < n; i++) {
68     if (op[i][0] == 3) {
69         c0 -= (pre[i] == 0);
70     } else if (op[i][0] == 2) {
71         sum -= op[i][1];
72     } else {
73         int x = op[i][1];
74         if (sum >= x) {
75             continue;
76         }
77         x -= sum;
78         int j = lower_bound(a.begin(), a.end(), x, [&](int i, int x) {
79             return pre[i] < x;
80         }) - a.begin() - 1;
81         Z res = 0;
82         int v = x;
83         while (j >= 0 && a[j] > i) {
84             res *= 2;
85             if (pre[a[j]] < v) {
86                 res += 1;
87                 v -= pre[a[j]];
88             }
89             j--;
90         }
91         res += 1;
92         res *= power(Z(2), c0);
93         ans += res;
94     }
95 }
96 cout << ans << "\n";
97 return 0;
98 }
```

## 820: Colorful Tree Again

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

An edge-weighted tree of  $n$  nodes is given with each edge colored in some color. Each node of this tree can be blocked or unblocked, all nodes are unblocked initially.

A simple path is a path in a graph that does not have repeating nodes. The length of a path is defined as the sum of weights of all edges on the path.

A path is good when it is a simple path consisting of edges of the same color  $c$ , all edges of color  $c$  are on this path, and every node on the path is unblocked.

You need to operate 2 kinds of queries:

block a node,

unblock a node.

After each query, print the maximum length among all good paths. If there are no good paths, print 0.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, q;
8     cin >> n >> q;
9     vector<vector<tuple<int, int, int>>> adj(n);
10    for (int i = 1; i < n; i++) {
11        int u, v, w, c;
12        cin >> u >> v >> w >> c;
13        u--, v--, c--;
14        adj[u].emplace_back(v, w, c);
15        adj[v].emplace_back(u, w, c);
16    }
17    vector<int> parent(n, -1), col(n), cntend(n), cnt(n);
18    vector<i64> weight(n);
19    vector<bool> bad(n);
20    auto dfs = [&](auto self, int x) -> void {
21        for (auto [y, w, c] : adj[x]) {
22            if (y == parent[x]) continue;
23            weight[c] += w;
24            col[y] = c;
25            parent[y] = x;
26            self(self, y);
27        }
28        for (auto [y, w, c] : adj[x]) {
29            cnt[c]++;
30            if (cnt[c] > 2) bad[c] = true;
31        }
32        for (auto [y, w, c] : adj[x]) {
33            if (cnt[c] == 1) {
34                cntend[c]++;
35            }
36            cnt[c] = 0;
37        }
38    };
39    dfs(dfs, 0);
40    for (int i = 0; i < n; i++) {
41        if (cntend[i] != 2) {
42            bad[i] = true;
43        }
44    }
45    vector<int> lca(n, -1);

```

```

46     for (int i = 1; i < n; i++) {
47         if (!bad[col[i]] && lca[col[i]] == -1) {
48             int j = i;
49             while (j > 0 && col[j] == col[i]) {
50                 j = parent[j];
51             }
52             lca[col[i]] = j;
53         }
54     }
55     vector<multiset<i64>> s(n);
56     for (int i = 0; i < n; i++) {
57         if (!bad[i]) {
58             s[lca[i]].insert(weight[i]);
59             // cerr << i+1 << " " << lca[i] + 1 << " " << weight[i] << "\n";
60         }
61     }
62     for (int i = 0; i < n; i++) {
63         s[i].insert(0);
64     }
65     multiset<i64> as{0};
66     vector<int> ban(n);
67     for (int i = 0; i < n; i++) {
68         as.insert(*s[i].rbegin());
69     }
70     vector<bool> good(n, true);
71     while (q--) {
72         int o, x;
73         cin >> o >> x;
74         x--;
75         if (o == 0) {
76             good[x] = false;
77             as.erase(as.find(*s[x].rbegin()));
78             if (x > 0 && !bad[col[x]]) {
79                 if (ban[col[x]]++ == 0) {
80                     int l = lca[col[x]];
81                     if (good[l]) {
82                         as.erase(as.find(*s[l].rbegin()));
83                     }
84                     s[l].erase(s[l].find(weight[col[x]]));
85                     if (good[l]) {
86                         as.insert(*s[l].rbegin());
87                     }
88                 }
89             }
90         } else {
91             good[x] = true;
92             as.insert(*s[x].rbegin());
93             if (x > 0 && !bad[col[x]]) {
94                 if (--ban[col[x]] == 0) {
95                     int l = lca[col[x]];
96                     if (good[l]) {
97                         as.erase(as.find(*s[l].rbegin()));
98                     }
99                     s[l].insert(weight[col[x]]);
100                    if (good[l]) {
101                        as.insert(*s[l].rbegin());
102                    }
103                }
104            }
105        }
106        auto ans = *as.rbegin();
107        cout << ans << "\n";
108    }
109    return 0;

```

```
110 }
```

## 821: Weighed Tree Radius

- Time limit: 6 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given a tree of  $n$  vertices and  $n - 1$  edges. The  $i$ -th vertex has an initial weight  $a_i$ .

Let the distance  $d_v(u)$  from vertex  $v$  to vertex  $u$  be the number of edges on the path from  $v$  to  $u$ . Note that  $d_v(u) = d_u(v)$  and  $d_v(v) = 0$ .

Let the weighted distance  $w_v(u)$  from  $v$  to  $u$  be  $w_v(u) = d_v(u) + a_u$ . Note that  $w_v(v) = a_v$  and  $w_v(u) \neq w_u(v)$  if  $a_u \neq a_v$ .

Analogically to usual distance, let's define the eccentricity  $e(v)$  of vertex  $v$  as the greatest weighted distance from  $v$  to any other vertex (including  $v$  itself), or  $e(v) = \max_{1 \leq u \leq n} w_v(u)$ .

Finally, let's define the radius  $r$  of the tree as the minimum eccentricity of any vertex, or  $r = \min_{1 \leq v \leq n} e(v)$ .

You need to perform  $m$  queries of the following form:

$v_j \ x_j$  - assign  $a_{v_j} = x_j$ .

After performing each query, print the radius  $r$  of the current tree.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class Info,
5          class Merge = plus<Info>>
6 # struct SegmentTree;
7 constexpr int inf = 1E9;
8 # struct Info;
9 Info operator+(const Info &a, const Info &b) {
10     Info c;
11     c.sid = max(a.sid, b.sid);
12     c.mid = max(a.mid, b.mid);
13     c.l = max({a.l, b.l, array{a.sid[0] + b.mid, a.sid[1]}});
14     c.r = max({a.r, b.r, array{b.sid[0] + a.mid, b.sid[1]}});
15     c.ans = max({a.ans, b.ans, array{a.l[0] + b.sid[0], a.l[1], b.sid[1]}, array{a.sid[0]
+ b.r[0], a.sid[1], b.r[1]}});
```

```
16     return c;
17 }
18 int main() {
19     ios::sync_with_stdio(false);
20     cin.tie(nullptr);
21     int n;
22     cin >> n;
23     vector<int> a(n);
24     for (int i = 0; i < n; i++) {
25         cin >> a[i];
26     }
27     vector<vector<int>> adj(n);
28     for (int i = 1; i < n; i++) {
29         int u, v;
30         cin >> u >> v;
31         u--;
32         v--;
33         adj[u].push_back(v);
34         adj[v].push_back(u);
35     }
36     vector<int> euler, teul(n), dep(n), parent(n, -1);
37     int tm = 0;
38     auto dfs = [&](auto dfs, int x) -> void {
39         euler.push_back(x);
40         teul[x] = tm++;
41         for (auto y : adj[x]) {
42             if (y == parent[x]) continue;
43             parent[y] = x;
44             dep[y] = dep[x] + 1;
45             dfs(dfs, y);
46             euler.push_back(x);
47             tm++;
48         }
49     };
50     dfs(dfs, 0);
51     vector<Info> ini(2 * n - 1);
52     for (int i = 0; i < 2 * n - 1; i++) {
53         int x = euler[i];
54         if (i == teul[x]) {
55             ini[i] = Info(dep[x], a[x], x);
56         } else {
57             ini[i] = Info(dep[x], 0, x);
58         }
59     }
60     SegmentTree<Info> seg(ini);
61     int m;
62     cin >> m;
63     while (m--) {
64         int x, y;
65         cin >> x >> y;
66         x--;
67         a[x] = y;
68         seg.modify(teul[x], Info(dep[x], a[x], x));
69         auto [d, u, v] = seg.info[1].ans;
70         if (dep[u] + a[u] < dep[v] + a[v]) swap(u, v);
71         int ans = max({a[u], a[v], (d + 1) / 2});
72         cout << ans << "\n";
73     }
74 }
```

**dp****822: Xenia and String Problem**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Xenia the coder went to The Olympiad of Informatics and got a string problem. Unfortunately, Xenia isn't fabulous in string algorithms. Help her solve the problem.

String  $s$  is a sequence of characters  $s_1s_2\dots s_{|s|}$ , where record  $|s|$  shows the length of the string.

Substring  $s[i\dots j]$  of string  $s$  is string  $s_i s_{i+1} \dots s_j$ .

String  $s$  is a Gray string, if it meets the conditions:

the length of string  $|s|$  is odd;

character occurs exactly once in the string;

either  $|s|=1$ , or substrings and are the same and are Gray strings.

For example, strings "abacaba", "xzx", "g" are Gray strings and strings "aaa", "xz", "abaxcbc" are not.

The beauty of string  $p$  is the sum of the squares of the lengths of all substrings of string  $p$  that are Gray strings. In other words, consider all pairs of values  $i, j$  ( $1 \leq i \leq j \leq |p|$ ). If substring  $p[i\dots j]$  is a Gray string, you should add  $(j - i + 1)^2$  to the beauty.

Xenia has got string  $t$  consisting of lowercase English letters. She is allowed to replace at most one letter of the string by any other English letter. The task is to get a string of maximum beauty.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     string s;
8     cin >> s;
9     int n = s.size();
10    const int logn = __lg(n + 1);
11    vector<vector<int>> nxt(logn, vector<int>(n));
12    for (int i = 0; i < logn; i++) {
13        for (int j = n - 1; j >= 0; j--) {
14            int k = j + (2 << i);
15            nxt[i][j] = k < n && s[j] == s[k] ? nxt[i][k] : k;
16        }
}

```

```

17     }
18     i64 ans = -1E18;
19     i64 res = n;
20     vector<array<i64, 26>> val(n);
21     vector<i64> d(n);
22     for (int i = n - 1; i >= 0; i--) {
23         for (int j = 2; i + (1 << j) - 1 <= n; j++) {
24             int bad = -1;
25             int badk = -1;
26             char badch;
27             bool ok = true;
28             int L = i, R = i + (1 << j) - 1;
29             bool vis[26] {};
30             for (int k = 0; k < j - 2; k++) {
31                 int a = i + (1 << k) - 1;
32                 char ch = s[a];
33                 if (nxt[k][a] < R) {
34                     if (bad != -1) {
35                         ok = false;
36                         break;
37                     }
38                     bad = nxt[k][a];
39                     badk = k;
40                     if (bad + (2 << k) < R && (s[bad + (2 << k)] != ch || nxt[k][bad + (2
41                         << k)] < R)) {
42                         if (nxt[k][a + (2 << k)] >= R) {
43                             bad = a;
44                             ch = s[a + (2 << k)];
45                         } else {
46                             ok = false;
47                             break;
48                         }
49                     }
50                     if (vis[ch - 'a']) {
51                         ok = false;
52                     }
53                     vis[ch - 'a'] = true;
54                     if (badk == k) {
55                         badch = ch;
56                     }
57                 }
58                 if (!ok) {
59                     continue;
60                 }
61 // cerr << L << " " << R << " " << s.substr(L, R - L) << " " << bad << "\n";
62                 int pl = L - 1 + (R - L + 1) / 4;
63                 int pr = L - 1 + (R - L + 1) / 4 * 3;
64                 int pm = (L - 1 + R) / 2;
65                 char sl = s[pl];
66                 char sr = s[pr];
67                 char sm = s[pm];
68                 if (sl != sr && bad != -1) {
69                     continue;
70                 }
71                 if (sl == sr && vis[sl - 'a']) {
72                     continue;
73                 }
74                 int chg = 0;
75                 if (sl != sr) {
76                     chg++;
77                 } else {
78                     vis[sl - 'a'] = true;
79                 }

```

```

80         if (vis[sm - 'a']) {
81             chg++;
82         }
83         if (bad != -1) {
84             chg++;
85         }
86         if (chg > 1) {
87             continue;
88         }
89         i64 cost = 1LL * (R - L) * (R - L);
90         if (sl != sr) {
91             vis[sm - 'a'] = true;
92             if (!vis[sl - 'a']) {
93                 val[pr][sl - 'a'] += cost;
94                 // cerr << "R ";
95             }
96             if (!vis[sr - 'a']) {
97                 val[pl][sr - 'a'] += cost;
98                 // cerr << "L ";
99             }
100        } else if (bad != -1) {
101            val[bad][badch - 'a'] += cost;
102            // cerr << "B ";
103        } else if (vis[sm - 'a']) {
104            for (int x = 0; x < 26; x++) {
105                if (!vis[x]) {
106                    val[pm][x] += cost;
107                }
108            }
109            // cerr << "M ";
110        } else {
111            res += cost;
112            assert(chg == 0);
113            d[L] -= cost;
114            if (R < n) {
115                d[R] += cost;
116            }
117            for (int x = 0; x < 26; x++) {
118                if (!vis[x]) {
119                    val[pm][x] += cost;
120                }
121            }
122            // cerr << L << " " << R << " " << s.substr(L, R - L) << " " << bad << "\n";
123        }
124    }
125    for (int i = 1; i < n; i++) {
126        d[i] += d[i - 1];
127    }
128    for (int i = 0; i < n; i++) {
129        for (int x = 0; x < 26; x++) {
130            ans = max(ans, res + val[i][x] + d[i]);
131        }
132    }
133    cout << ans << "\n";
134    return 0;
135 }
136 }
```

**823: Difficult Mountain**

- Time limit: 2 seconds

- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

A group of  $n$  alpinists has just reached the foot of the mountain. The initial difficulty of climbing this mountain can be described as an integer  $d$ .

Each alpinist can be described by two integers  $s$  and  $a$ , where  $s$  is his skill of climbing mountains and  $a$  is his neatness.

An alpinist of skill level  $s$  is able to climb a mountain of difficulty  $p$  only if  $p \leq s$ . As an alpinist climbs a mountain, they affect the path and thus may change mountain difficulty. Specifically, if an alpinist of neatness  $a$  climbs a mountain of difficulty  $p$  the difficulty of this mountain becomes  $\max(p, a)$ .

Alpinists will climb the mountain one by one. And before the start, they wonder, what is the maximum number of alpinists who will be able to climb the mountain if they choose the right order. As you are the only person in the group who does programming, you are to answer the question.

Note that after the order is chosen, each alpinist who can climb the mountain, must climb the mountain at that time.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class Info, class Tag>
5 # struct LazySegmentTree;
6 # struct Tag;
7 # struct Info;
8 Info operator+(Info a, Info b) {
9     return {max(a.max, b.max)};
10 }
11 int main() {
12     ios::sync_with_stdio(false);
13     cin.tie(nullptr);
14     int n, d;
15     cin >> n >> d;
16     vector<int> s(n), a(n);
17     vector<int> v;
18     v.reserve(2 * n + 1);
19     v.push_back(d);
20     for (int i = 0; i < n; i++) {
21         cin >> s[i] >> a[i];
22         v.push_back(s[i]);
23         v.push_back(a[i]);
24     }
25     sort(v.begin(), v.end());
26     d = lower_bound(v.begin(), v.end(), d) - v.begin();
27     for (int i = 0; i < n; i++) {
28         s[i] = lower_bound(v.begin(), v.end(), s[i]) - v.begin();
29         a[i] = lower_bound(v.begin(), v.end(), a[i]) - v.begin();
30     }

```

```

31     vector<int> p(n);
32     iota(p.begin(), p.end(), 0);
33     sort(p.begin(), p.end(),
34           [&](int i, int j) {
35             return s[i] + a[i] < s[j] + a[j];
36           });
37     LazySegmentTree<Info, Tag> seg(2 * n + 1);
38     seg.modify(d, {0});
39     for (auto i : p) {
40       if (s[i] <= a[i]) {
41         auto v = seg.rangeQuery(0, s[i] + 1).max + 1;
42         seg.modify(a[i], {v});
43       } else {
44         auto v = seg.rangeQuery(0, a[i] + 1).max + 1;
45         seg.modify(a[i], {v});
46         seg.rangeApply(a[i] + 1, s[i] + 1, {1});
47       }
48     }
49     int ans = seg.rangeQuery(0, 2 * n + 1).max;
50     cout << ans << "\n";
51   }
52 }
```

## 825: Elevators of Tamem

- Time limit: 5 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

There is a building named Taman Membeku (shortened as Tamem). The building has  $N$  floors numbered from 1 to  $N$  from bottom to top. The only way to move between floors in the building is to use elevators. There are 3 elevators available in Tamem, namely elevators 1, 2, and 3.

Pak Chanek works as an elevator operator in Tamem. Pak Chanek will work for  $Q$  days. Initially, each elevator is in floor 1 and all elevators are on. On each day, exactly one of the following will happen:

1  $x$   $y$  - There is a person currently in floor  $x$  who wants to go to floor  $y$ . ( $1 \leq x, y \leq N$ ;  $x \neq y$ )

2  $p$  - Elevator  $p$  changes state at the start of the day. If previously it is on, then it will turn off. If previously it is off, then it will turn on. ( $1 \leq p \leq 3$ )

For each day, Pak Chanek can control the movement of all elevators as he pleases. However, for each day where there is a person currently in floor  $x$  who wants to go to floor  $y$ , among all elevator movements done by Pak Chanek, the following must happen:

One elevator moves to floor  $x$ .

The person gets into the elevator.

The elevator moves to floor  $y$ .

The person gets out of the elevator.

For each day, Pak Chanek can only move the elevators that are currently on. Note that, since a change in state happens at the start of the day, this means that an elevator that turns off on some day starts becoming unusable from that day itself. Conversely, an elevator that turns on on some day starts becoming usable from that day itself.

It is known that the electricity fee for each day is different. More specifically, on the  $j$ -th day, the fee needed to move one elevator up or down by one floor is  $A_j$ .

From the start, Pak Chanek already knows the electricity fees and the sequence of events that will happen on the  $Q$  days, so Pak Chanek can operate the elevators strategically. What is the minimum possible electricity fee to fulfill all needs of the people who want to move between floors in Tamem? Note: in the end, each elevator does not have to go back to floor 1.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 1E18;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int N, Q;
9     cin >> N >> Q;
10    vector<int> A(Q);
11    for (int i = 0; i < Q; i++) {
12        cin >> A[i];
13    }
14    vector<array<int, 3>> work(Q), a;
15    work[0] = {1, 1, 1};
16    a.push_back({0, 1, 1});
17    for (int i = 0; i < Q; i++) {
18        int o;
19        cin >> o;
20        if (i) {
21            work[i] = work[i - 1];
22        }
23        if (o == 2) {
24            int x;
25            cin >> x;
26            x--;
27            work[i][x] ^= 1;
28        } else {
29            int x, y;
30            cin >> x >> y;
31            a.push_back({i, x, y});
32        }
33    }
34    N = a.size();
35    vector val(3, vector(Q, vector<int>(Q)));
36    for (int k = 0; k < 3; k++) {
37        for (int r = 0; r < Q; r++) {
38            int v = A[r];
39            for (int l = r; l >= 0; l--) {

```

```

40             if (work[l][k]) {
41                 v = min(v, A[l]);
42             }
43             val[k][l][r] = v;
44         }
45     }
46 }
47 vector<int> dp(N, vector<int>(N, vector<int>(N, inf)));
48 dp[0][0][0] = 0;
49 i64 ans = inf;
50 for (int i = 0; i < N; i++) {
51     for (int j = 0; j < N; j++) {
52         for (int k = 0; k < N; k++) {
53             int l = max({i, j, k}) + 1;
54             if (l == N) {
55                 ans = min(ans, dp[i][j][k]);
56                 continue;
57             }
58             auto [t, x, y] = a[l];
59             if (work[t][0]) {
60                 int tm = a[i][0];
61                 int e = a[i][2];
62                 dp[l][j][k] = min(dp[l][j][k], dp[i][j][k] + 1LL * abs(e - x) * val
63                                 [0][tm][t] + 1LL * abs(x - y) * A[t]);
64             }
65             if (work[t][1]) {
66                 int tm = a[j][0];
67                 int e = a[j][2];
68                 dp[i][l][k] = min(dp[i][l][k], dp[i][j][k] + 1LL * abs(e - x) * val
69                                 [1][tm][t] + 1LL * abs(x - y) * A[t]);
70             }
71             if (work[t][2]) {
72                 int tm = a[k][0];
73                 int e = a[k][2];
74                 dp[i][j][l] = min(dp[i][j][l], dp[i][j][k] + 1LL * abs(e - x) * val
75                                 [2][tm][t] + 1LL * abs(x - y) * A[t]);
76             }
77         }
78     }
79 }
79 cout << ans << "\n";
80 return 0;
81 }
```

## 826: Jackets and Packets

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Pak Chanek has  $N$  jackets that are stored in a wardrobe. Pak Chanek's wardrobe has enough room for two stacks of jackets, namely the left stack and the right stack. Initially, all  $N$  jackets are in the left stack, while the right stack is empty. Initially, the  $i$ -th jacket from the top of the left stack has colour  $C_i$ .

Pak Chanek wants to pack all of those jackets into some packets, such that the jackets in the same packet has the same colour. However, it is possible for two jackets with the same colour to be in different packets.

Pak Chanek can do two kinds of operations:

Pak Chanek can pick any number of jackets from the top from one stack as long as the colour is the same, then take those jackets out of the stack and pack them into a packet. This packing operation takes  $X$  minutes, no matter how many jackets are packed.

Pak Chanek can move one topmost jacket from one stack to the top of the other stack (possibly right to left or left to right). This moving operation takes  $Y$  minutes.

Determine the minimum time to pack all jackets!

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 1E18;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int N, X, Y;
9     cin >> N >> X >> Y;
10    vector<int> C(N);
11    for (int i = 0; i < N; i++) {
12        cin >> C[i];
13        C[i]--;
14    }
15    vector<i64> dp(N + 1, vector<i64>(N + 1));
16    vector<i64> g(N + 1, vector<i64>(N + 1));
17    vector<i64> h(N + 1, vector<i64>(N + 1));
18    for (int r = 1; r <= N; r++) {
19        for (int l = r - 1; l >= 0; l--) {
20            dp[l][r] = inf;
21            for (int i = l + 1; i < r; i++) {
22                dp[l][r] = min(dp[l][r], dp[l][i] + dp[i][r]);
23            }
24            g[l][r] = inf;
25            h[l][r] = inf;
26            if (C[l] == C[r - 1]) {
27                g[l][r] = min(g[l][r], g[l][r - 1] + Y);
28            }
29            if (C[l] == C[r - 1]) {
30                h[l][r] = min(h[l][r], h[l][r - 1] + 2 * Y);
31            }
32            for (int i = l + 1; i < r; i++) {
33                g[l][r] = min(g[l][r], g[l][i] + dp[i][r]);
34                h[l][r] = min(h[l][r], h[l][i] + dp[i][r]);
35            }
36            for (int i = r - 1; i >= l; i--) {
37                if (C[i] != C[l]) {
38                    break;
39                }
40                dp[l][r] = min(dp[l][r], h[l][i] + X);
}

```

```

41         }
42         dp[l][r] = min(dp[l][r], g[l][r] + x);
43     }
44     cout << dp[0][N] << "\n";
45     return 0;
46 }

```

## 827: e-Government

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The best programmers of Embezzland compete to develop a part of the project called “e-Government” - the system of automated statistic collecting and press analysis.

We know that any of the  $k$  citizens can become a member of the Embezzland government. The citizens’ surnames are  $a_1, a_2, \dots, a_k$ . All surnames are different. Initially all  $k$  citizens from this list are members of the government. The system should support the following options:

Include citizen  $a_i$  to the government.

Exclude citizen  $a_i$  from the government.

Given a newspaper article text, calculate how politicized it is. To do this, for every active government member the system counts the number of times his surname occurs in the text as a substring. All occurrences are taken into consideration, including the intersecting ones. The degree of politicization of a text is defined as the sum of these values for all active government members.

Implement this system.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int N = 1 << 20;
5 int trie[N][26];
6 int fail[N];
7 int cnt = 1;
8 int in[N], out[N];
9 int cur = 0;
10 int fen[N];
11 vector<int> adj[N];
12 void add(int x, int y) {

```

```

13     for (int i = x + 1; i <= N; i += i & -i) {
14         fen[i - 1] += y;
15     }
16 }
17 int sum(int x) {
18     int ans = 0;
19     for (int i = x; i; i &= i - 1) {
20         ans += fen[i - 1];
21     }
22     return ans;
23 }
24 int main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
27     fill(trie[0], trie[0] + 26, 1);
28     int n, k;
29     cin >> n >> k;
30     vector<string> a(k);
31     vector<int> pos(k);
32     for (int i = 0; i < k; i++) {
33         cin >> a[i];
34         int p = 1;
35         for (auto c : a[i]) {
36             int &q = trie[p][c - 'a'];
37             if (!q) {
38                 q = ++cnt;
39             }
40             p = q;
41         }
42         pos[i] = p;
43     }
44     queue<int> q;
45     q.push(1);
46     while (!q.empty()) {
47         int x = q.front();
48         q.pop();
49         for (int i = 0; i < 26; i++) {
50             if (trie[x][i]) {
51                 fail[trie[x][i]] = trie[fail[x]][i];
52                 q.push(trie[x][i]);
53             } else {
54                 trie[x][i] = trie[fail[x]][i];
55             }
56         }
57     }
58     for (int i = 2; i <= cnt; i++) {
59         adj[fail[i]].push_back(i);
60     }
61     auto dfs = [&](auto self, int x) -> void {
62         in[x] = cur++;
63         for (auto y : adj[x]) {
64             self(self, y);
65         }
66         out[x] = cur;
67     };
68     dfs(dfs, 1);
69     for (int i = 0; i < k; i++) {
70         add(in[pos[i]], 1);
71         add(out[pos[i]], -1);
72     }
73     vector<int> isin(k, 1);
74     for (int _ = 0; _ < n; _++) {
75         char op;
76         cin >> op;

```

```

77         if (op == '+') {
78             int x;
79             cin >> x;
80             x--;
81             if (!isin[x]) {
82                 add(in[pos[x]], 1);
83                 add(out[pos[x]], -1);
84                 isin[x] = 1;
85             }
86         } else if (op == '-') {
87             int x;
88             cin >> x;
89             x--;
90             if (isin[x]) {
91                 add(in[pos[x]], -1);
92                 add(out[pos[x]], 1);
93                 isin[x] = 0;
94             }
95         } else {
96             string s;
97             cin >> s;
98             int p = 1;
99             i64 ans = 0;
100            for (auto c : s) {
101                p = trie[p][c - 'a'];
102                ans += sum(in[p] + 1);
103            }
104            cout << ans << "\n";
105        }
106    }
107    return 0;
108 }
```

## 828: Miriany and Matchstick

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Miriany's matchstick is a  $2 \times n$  grid that needs to be filled with characters A or B.

He has already filled in the first row of the grid and would like you to fill in the second row. You must do so in a way such that the number of adjacent pairs of cells with different characters<sup>†</sup> is equal to  $k$ . If it is impossible, report so.

<sup>†</sup> An adjacent pair of cells with different characters is a pair of cells  $(r_1, c_1)$  and  $(r_2, c_2)$  ( $1 \leq r_1, r_2 \leq 2$ ,  $1 \leq c_1, c_2 \leq n$ ) such that  $|r_1 - r_2| + |c_1 - c_2| = 1$  and the characters in  $(r_1, c_1)$  and  $(r_2, c_2)$  are different.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, k;
6     cin >> n >> k;
7     string s;
8     cin >> s;
9     for (int i = 1; i < n; i++) {
10         k -= (s[i] != s[i - 1]);
11     }
12     if (k < 0) {
13         cout << "NO\n";
14         return;
15     }
16     vector dp(2, vector<vector<pair<int, int>>>(n));
17     dp[0][0].emplace_back(s[0] != 'A', s[0] != 'A');
18     dp[1][0].emplace_back(s[0] != 'B', s[0] != 'B');
19     for (int i = 1; i < n; i++) {
20         for (int t = 0; t < 2; t++) {
21             for (auto [l, r] : dp[t][i - 1]) {
22                 dp[t][i].emplace_back(l, r);
23                 dp[!t][i].emplace_back(l + 1, r + 1);
24             }
25         }
26         for (int t = 0; t < 2; t++) {
27             vector<pair<int, int>> f;
28             sort(dp[t][i].begin(), dp[t][i].end());
29             int d = (s[i] != 'A' + t);
30             for (auto [l, r] : dp[t][i]) {
31                 l += d;
32                 r += d;
33                 if (!f.empty() && l <= f.back().second + 1) {
34                     f.back().second = max(f.back().second, r);
35                 } else {
36                     f.emplace_back(l, r);
37                 }
38             }
39             dp[t][i] = f;
40         }
41     }
42     auto get = [&](auto &f, int x) {
43         for (auto [l, r] : f) {
44             if (l <= x && x <= r) {
45                 return true;
46             }
47         }
48         return false;
49     };
50     for (int t = 0; t < 2; t++) {
51         if (get(dp[t][n - 1], k)) {
52             cout << "YES\n";
53             auto ans = s;
54             for (int i = n - 1; i > 0; i--) {
55                 k -= (s[i] != 'A' + t);
56                 ans[i] = 'A' + t;
57                 if (!get(dp[t][i - 1], k)) {
58                     t = !t;
59                     k--;
60                 }
61             }
62             ans[0] = 'A' + t;
63         }
64     }
65 }
```

```

63         cout << ans << "\n";
64     }
65 }
66 cout << "NO\n";
67 }
68 int main() {
69     ios::sync_with_stdio(false);
70     cin.tie(nullptr);
71     int t;
72     cin >> t;
73     while (t--) {
74         solve();
75     }
76     return 0;
77 }
```

## 829: Two Subsequences

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

On an IT lesson Valera studied data compression. The teacher told about a new method, which we shall now describe to you.

Let  $\{a_1, a_2, \dots, a_n\}$  be the given sequence of lines needed to be compressed. Here and below we shall assume that all lines are of the same length and consist only of the digits 0 and 1. Let's define the compression function:

$f(\text{empty sequence}) = \text{empty string}$

$f(s) = s$ .

$f(s_1, s_2) =$  the smallest in length string, which has one of the prefixes equal to  $s_1$  and one of the suffixes equal to  $s_2$ . For example,  $f(001, 011) = 0011$ ,  $f(111, 011) = 111011$ .

$f(a_1, a_2, \dots, a_n) = f(f(a_1, a_2, a_{n-1}), a_n)$ . For example,  $f(000, 000, 111) = f(f(000, 000), 111) = f(000, 111) = 000111$ .

Valera faces a real challenge: he should divide the given sequence  $\{a_1, a_2, \dots, a_n\}$  into two subsequences  $\{b_1, b_2, \dots, b_k\}$  and  $\{c_1, c_2, \dots, c_m\}$ ,  $m + k = n$ , so that the value of  $S = |f(b_1, b_2, \dots, b_k)| + |f(c_1, c_2, \dots, c_m)|$  took the minimum possible value. Here  $|p|$  denotes the length of the string  $p$ .

Note that it is not allowed to change the relative order of lines in the subsequences. It is allowed to make one of the subsequences empty. Each string from the initial sequence should belong to exactly one subsequence. Elements of subsequences  $b$  and  $c$  don't have to be consecutive in the original sequence  $a$ , i. e. elements of  $b$  and  $c$  can alternate in  $a$  (see samples 2 and 3).

Help Valera to find the minimum possible value of S.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<string> s(n);
10    vector<int> a(n);
11    for (int i = 0; i < n; i++) {
12        cin >> s[i];
13        for (int j = 0; j < s[i].size(); j++) {
14            a[i] |= (s[i][j] - '0') << j;
15        }
16    }
17    int m = s[0].size();
18    vector<int> dp(1 << (m + 1), -1E9);
19    dp[1] = 0;
20    int add = 0;
21    for (int i = 1; i < n; i++) {
22        int f = 0;
23        for (int len = 0; len <= m; len++) {
24            f = max(f, add + dp[1 << len | (a[i] & ((1 << len) - 1)) + len]);
25        }
26        int mxlen = 0;
27        for (int len = m; len > 0; len--) {
28            if ((a[i] & ((1 << len) - 1)) == (a[i - 1] >> (m - len))) {
29                mxlen = len;
30                break;
31            }
32        }
33        add += mxlen;
34        for (int len = 0; len <= m; len++) {
35            int &res = dp[1 << len | (a[i - 1] >> (m - len))];
36            res = max(res, f - add);
37        }
38    }
39    int ans = n * m;
40    ans -= *max_element(dp.begin(), dp.end()) + add;
41    cout << ans << "\n";
42    return 0;
43 }
```

### 830: Mex Tree

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a tree with  $n$  nodes. For each node, you either color it in 0 or 1.

The value of a path  $(u, v)$  is equal to the MEX<sup>†</sup> of the colors of the nodes from the shortest path between  $u$  and  $v$ .

The value of a coloring is equal to the sum of values of all paths  $(u, v)$  such that  $1 \leq u \leq v \leq n$ .

What is the maximum possible value of any coloring of the tree?

<sup>†</sup> The MEX (minimum excluded) of an array is the smallest non-negative integer that does not belong to the array. For instance:

The MEX of  $[2, 2, 1]$  is 0, because 0 does not belong to the array.

The MEX of  $[3, 1, 0, 1]$  is 2, because 0 and 1 belong to the array, but 2 does not.

The MEX of  $[0, 3, 1, 2]$  is 4 because 0, 1, 2, and 3 belong to the array, but 4 does not.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int inf = 1E9;
5 void solve() {
6     int n;
7     cin >> n;
8     vector<vector<int>> adj(n);
9     // mt19937 rng;
10    for (int i = 1; i < n; i++) {
11        int u, v;
12        cin >> u >> v;
13        // u = i + 1;
14        // v = rng() % i + 1;
15        u--, v--;
16        adj[u].push_back(v);
17        adj[v].push_back(u);
18    }
19    i64 ans = 1LL * n * (n + 1);
20    vector dp(n, vector(2, vector<int>{}));
21    function<void(int, int)> dfs = [&](int x, int p) {
22        dp[x][0] = dp[x][1] = {0, 0};
23        for (auto y : adj[x]) {
24            if (y == p) {
25                continue;
26            }
27            dfs(y, x);
28            for (int i = 0; i < 2; i++) {
29                vector<int> g(dp[x][i].size() + dp[y][i].size() - 1, inf);
30                for (int a = 1; a < dp[x][i].size(); a++) {
31                    for (int b = 0; b < dp[y][i].size(); b++) {
32                        g[a + b] = min(g[a + b], dp[x][i][a] + dp[y][i][b]);
33                    }
34                }
35                int s = g.size() - 1;
36                while (s > 1 && g[s] + 1LL * s * (s + 1) / 2 * (i + 1) >= g[s - 1] + 1LL *
37                    (s - 1) * s / 2 * (i + 1)) {
38                    g.pop_back();
39                    s--;
40                }
41            }
42        }
43    }
44 }
```

```

40             dp[x][i] = move(g);
41         }
42     }
43     for (int i = 0; i < 2; i++) {
44         i64 res = 1E18;
45         for (int j = 1; j < dp[x][!i].size(); j++) {
46             res = min(res, dp[x][!i][j] + 1LL * j * (j + 1) / 2 * (!i + 1));
47         }
48         dp[x][i][0] = res;
49     }
50 };
51 dfs(0, -1);
52 ans -= min(dp[0][0][0], dp[0][1][0]);
53 cout << ans << "\n";
54 }
55 int main() {
56     ios::sync_with_stdio(false);
57     cin.tie(nullptr);
58     int t;
59     cin >> t;
60     while (t--) {
61         solve();
62     }
63     return 0;
64 }
```

## 831: Sequence Transformation

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You've got a non-decreasing sequence  $x_1, x_2, \dots, x_n$  ( $1 \leq x_1 \leq x_2 \leq \dots \leq x_n \leq q$ ). You've also got two integers  $a$  and  $b$  ( $a \leq b$ ;  $a \cdot (n - 1) < q$ ).

Your task is to transform sequence  $x_1, x_2, \dots, x_n$  into some sequence  $y_1, y_2, \dots, y_n$  ( $1 \leq y_i \leq q$ ;  $a \leq y_i + 1 - y_i \leq b$ ). The transformation price is the following sum: . Your task is to choose such sequence  $y$  that minimizes the described transformation price.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr double eps = 1E-8;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, q, a, b;
```

```

9      cin >> n >> q >> a >> b;
10     vector<int> x(n);
11     for (int i = 0; i < n; i++) {
12         cin >> x[i];
13         x[i] -= a * i + 1;
14     }
15     q -= a * (n - 1) + 1;
16     b -= a;
17     vector<array<double, 2>> f;
18     f.push_back({0, 0});
19     if (q) {
20         f.push_back({q, 0});
21     }
22     cout << fixed << setprecision(10);
23     double f0 = 0;
24     vector<double> g(n);
25     for (int i = 0; i < n; i++) {
26         f0 += 1. * x[i] * x[i];
27         for (auto &[X, y] : f) {
28             y += 2 * (X - x[i]);
29         }
30         int pos = f.size();
31         for (int j = 0; j < f.size(); j++) {
32             if (f[j][1] > 0) {
33                 if (j == 0 || f[j][0] - f[j - 1][0] < eps) {
34                     g[i] = f[j][0];
35                 } else {
36                     double x = (f[j][0] - f[j - 1][0]) / (f[j][1] - f[j - 1][1]) * -f[j - 1][1] + f[j - 1][0];
37                     g[i] = x;
38                 }
39                 pos = j;
40                 break;
41             } else {
42                 g[i] = f[j][0];
43             }
44         }
45         // cerr << "g : " << g[i] << "\n";
46         if (b && i < n - 1) {
47             bool ok = false;
48             f.insert(f.begin() + pos, array{g[i], 0.});
49             f.insert(f.begin() + pos, array{g[i], 0.});
50             for (int j = pos + 1; j < f.size(); j++) {
51                 f[j][0] += b;
52             }
53             while (f.size() > 1 && f.back()[0] > q) {
54                 if (f[f.size() - 2][0] > q) {
55                     f.pop_back();
56                 } else {
57                     auto a = f[f.size() - 2];
58                     auto b = f.back();
59                     auto k = (b[1] - a[1]) / (b[0] - a[0]);
60                     f.back() = {q, a[1] + (q - a[0]) * k};
61                     break;
62                 }
63             }
64         }
65         // for (auto [x, y] : f) {
66         //     cerr << "(" << x << ", " << y << ") ";
67         // }
68         // cerr << "\n";
69     }
70     for (int i = n - 2; i >= 0; i--) {
71         if (g[i] > g[i + 1]) {

```

```

72         g[i] = g[i + 1];
73     } else if (g[i] < g[i + 1] - b) {
74         g[i] = g[i + 1] - b;
75     }
76 }
77 double ans = 0;
78 for (int i = 0; i < n; i++) {
79     ans += (g[i] - x[i]) * (g[i] - x[i]);
80     g[i] += 1 + a * i;
81     cout << g[i] << "\n"[i == n - 1];
82 }
83 cout << ans << "\n";
84 return 0;
85 }
```

## 832: Misha and Apples

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Schoolboy Misha got tired of doing sports programming, so he decided to quit everything and go to the magical forest to sell magic apples.

His friend Danya came to the magical forest to visit Misha. What was his surprise when he found out that Misha found a lot of friends there, the same former sports programmers. And all of them, like Misha, have their own shop where they sell magic apples. To support his friends, who have changed their lives so drastically, he decided to buy up their entire assortment.

The buying process works as follows: in total there are  $n$  stalls, numbered with integers from 1 to  $n$ , and  $m$  kinds of magic apples, numbered with integers from 1 to  $m$ . Each shop sells some number of kinds of apples. Danya visits all the shops in order of increasing number, starting with the first one. Upon entering the shop he buys one magic apple of each kind sold in that shop and puts them in his backpack.

However, magical apples wouldn't be magical if they were all right. The point is that when two apples of the same type end up together in the backpack, all of the apples in it magically disappear. Importantly, the disappearance happens after Danya has put the apples in the backpack and left the shop.

Upon returning home, Danya realized that somewhere in the forest he had managed to lose his backpack. Unfortunately, for some shops Danya had forgotten what assortment of apples there was. Remembering only for some shops, what kinds of magical apples were sold in them, he wants to know what is the maximum number of apples he could have in his backpack after all his purchases at best.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<vector<int>> a(n);
8     for (int i = 0; i < n; i++) {
9         int k;
10        cin >> k;
11        a[i].resize(k);
12        for (int j = 0; j < k; j++) {
13            cin >> a[i][j];
14            a[i][j]--;
15        }
16    }
17    vector<int> nxt(n + 1);
18    nxt[n] = n;
19    for (int i = n - 1; i >= 0; i--) {
20        nxt[i] = a[i].empty() ? i : nxt[i + 1];
21    }
22    int ans = 0;
23    vector<int> dp(n + 1);
24    dp[0] = 1;
25    dp[1] = -1;
26    int bad = 0;
27    map<int, int> cnt;
28    int sum = 0;
29    for (int i = 0, j = 0; i <= n; i++) {
30        if (i) {
31            dp[i] += dp[i - 1];
32        }
33        if (i == n) {
34            break;
35        }
36        while (!bad && j < n) {
37            for (auto x : a[j]) {
38                bad += ++cnt[x] == 2;
39            }
40            sum += a[j].size();
41            j++;
42        }
43        if (dp[i]) {
44            if (bad) {
45                dp[j] += 1;
46                if (j < n) {
47                    dp[j + 1] -= 1;
48                }
49            }
50            int nj = j - (bad > 0);
51            if (nxt[i] < nj) {
52                dp[nxt[i] + 1] += 1;
53                if (nj < n) {
54                    dp[nj + 1] -= 1;
55                }
56            }
57            if (nj == n) {
58                ans = max(ans, sum);
59                if (nxt[i] < n) {
60                    ans = m;
61                }
62            }
63        }
64    }
65}
```

```

63         }
64         for (auto x : a[i]) {
65             bad -= --cnt[x] == 1;
66         }
67         sum -= a[i].size();
68     }
69     cout << ans << "\n";
70 }
71 int main() {
72     ios::sync_with_stdio(false);
73     cin.tie(nullptr);
74     int t;
75     cin >> t;
76     while (t--) {
77         solve();
78     }
79     return 0;
80 }
```

### 833: Forward, march!

- Time limit: 1 second
- Memory limit: 64 megabytes
- Input file: standard input
- Output file: standard output

Jack has become a soldier now. Unfortunately, he has trouble with the drill. Instead of marching beginning with the left foot and then changing legs with each step, as ordered, he keeps repeating a sequence of steps, in which he sometimes makes the wrong steps or - horror of horrors! - stops for a while. For example, if Jack uses the sequence ‘right, left, break’, when the sergeant yells: ‘Left! Right! Left! Right! Left! Right!’, Jack first makes a step with the right foot, then one with the left foot, then he is confused and stops for a moment, then again - this time according to the order - starts with the right foot, then uses the left foot, then - to the sergeant’s irritation - he stops to catch his breath, to incorrectly start with the right foot again... Marching this way, Jack will make the step that he is supposed to in the given moment in only one third of cases.

When the officers convinced him he should do something about it, Jack decided to modify the basic sequence of steps that he repeats. However, in order not to get too tired, he has decided that the only thing he’ll do is adding any number of breaks in any positions of the original sequence (a break corresponds to stopping for the duration of one step). Of course, Jack can’t make a step on the same foot twice in a row, if there is no pause between these steps. It is, however, not impossible that the sequence of steps he used so far is incorrect (it would explain a lot, actually).

Help Private Jack! Given the sequence of steps he keeps repeating, calculate the maximal percentage of time that he can spend marching correctly after adding some breaks to his scheme.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 i64 get(string s) {
5     int n = s.size();
6     i64 lo = 0, hi = 1000000000LL;
7     while (lo < hi) {
8         i64 x = (lo + hi + 1) / 2;
9         constexpr i64 inf = 1E18;
10        constexpr i64 V = 1E8;
11        array<i64, 2> dp{0LL, -inf};
12        for (int i = 0; i < n; i++) {
13            array<i64, 2> g{-inf, -inf};
14            if (s[i] == 'L') {
15                g[0] = dp[1] - x;
16                g[1] = max(dp[1] - 2 * x + V, dp[0] - x + V);
17            } else if (s[i] == 'R') {
18                g[1] = dp[0] - x;
19                g[0] = max(dp[0] - 2 * x + V, dp[1] - x + V);
20            } else {
21                g[0] = dp[1] - x;
22                g[1] = dp[0] - x;
23            }
24            dp = g;
25        }
26        if (max(dp[0], dp[1] - x) >= 0) {
27            lo = x;
28        } else {
29            hi = x - 1;
30        }
31    }
32    return lo;
33 }
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     string s;
38     cin >> s;
39     string ns;
40     for (auto c : s) {
41         if (!ns.empty() && c != 'X' && c == ns.back()) {
42             ns += 'X';
43         }
44         ns += c;
45     }
46     i64 ans;
47     if (s[0] == s.back() && s[0] != 'X') {
48         ans = max(get('X' + ns), get(ns + 'X'));
49     } else {
50         ans = get(ns);
51     }
52     auto t = to_string(ans);
53     while (t.size() < 7) {
54         t = '0' + t;
55     }
56     t.insert(t.end() - 6, '.');
57     cout << t << "\n";
58     return 0;
59 }
```

### 834: Minibuses on Venus (hard version)

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. The only difference between the three versions is the constraints on  $n$  and  $k$ . You can make hacks only if all versions of the problem are solved.

Maxim is a minibus driver on Venus.

To ride on Maxim's minibus, you need a ticket. Each ticket has a number consisting of  $n$  digits. However, as we know, the residents of Venus use a numeral system with base  $k$ , rather than the decimal system. Therefore, the ticket number can be considered as a sequence of  $n$  integers from 0 to  $k - 1$ , inclusive.

The residents of Venus consider a ticket to be lucky if there is a digit on it that is equal to the sum of the remaining digits, modulo  $k$ . For example, if  $k = 10$ , then the ticket 7135 is lucky because  $7 + 1 + 5 \equiv 3 \pmod{10}$ . On the other hand, the ticket 7136 is not lucky because no digit is equal to the sum of the others modulo 10.

Once, while on a trip, Maxim wondered: how many lucky tickets exist? At the same time, Maxim understands that this number can be very large, so he is interested only in the answer modulo some prime number  $m$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
```

```

25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 using Z = MInt<0>;
33 vector<Z> operator*(const vector<Z> &a, const vector<Z> &b) {
34     int n = a.size();
35     vector<Z> c(n);
36     vector<i64> d(n);
37     for (int i = 0; i < n; i++) {
38         for (int j = 0; j < n; j++) {
39             int k = (i + j) % n;
40             if (d[k] > 8E18) {
41                 c[k] += d[k];
42                 d[k] = 0;
43             }
44             d[k] += 1LL * int(a[i]) * int(b[j]);
45         }
46     }
47     for (int i = 0; i < n; i++) {
48         c[i] += d[i];
49     }
50     return c;
51 }
52 vector<Z> power(vector<Z> a, i64 n) {
53     vector<Z> res(a.size());
54     res[0] = 1;
55     for (; n; n /= 2, a = a * a) {
56         if (n & 1) {
57             res = res * a;
58         }
59     }
60     return res;
61 }
62 int main() {
63     ios::sync_with_stdio(false);
64     cin.tie(nullptr);
65     i64 n;
66     int k, m;
67     cin >> n >> k >> m;
68     Z::setMod(m);
69     Z ans = 0;
70     if (k % 2 == 1) {
71         vector<Z> a(k, 1);
72         a[0] = 0;
73         auto g = power(a, n);
74         for (int s = 0; s < k; s++) {
75             ans += g[(s - n % k * (1LL * s * (k + 1) / 2) % k + k) % k];
76         }
77     } else {
78         vector<Z> a(k, 1);
79         a[0] = a[k / 2] = 0;
80         auto g = power(a, n);
81         for (int s = 0; s < k; s++) {
82             if (s % 2 == 1) {
83                 ans += power(Z(k), n - 1);
84             } else {
85                 ans += g[(s - n % k * (s / 2) % k + k) % k];
86             }
87         }
88     }
}

```

```

89     ans = power(Z(k), n) - ans;
90     cout << ans << "\n";
91     return 0;
92 }
```

## 835: Prediction

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Consider a tournament with  $n$  participants. The rating of the  $i$ -th participant is  $a_i$ .

The tournament will be organized as follows. First of all, organizers will assign each participant an index from 1 to  $n$ . All indices will be unique. Let  $p_i$  be the participant who gets the index  $i$ .

Then,  $n - 1$  games will be held. In the first game, participants  $p_1$  and  $p_2$  will play. In the second game, the winner of the first game will play against  $p_3$ . In the third game, the winner of the second game will play against  $p_4$ , and so on - in the last game, the winner of the  $(n - 2)$ -th game will play against  $p_n$ .

Monocarp wants to predict the results of all  $n - 1$  games (of course, he will do the prediction only after the indices of the participants are assigned). He knows for sure that, when two participants with ratings  $x$  and  $y$  play, and  $|x - y| > k$ , the participant with the higher rating wins. But if  $|x - y| \leq k$ , any of the two participants may win.

Among all  $n!$  ways to assign the indices to participants, calculate the number of ways to do this so that Monocarp can predict the results of all  $n - 1$  games. Since the answer can be large, print it modulo 998244353.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
```

```

15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<OLL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int n, k;
39     cin >> n >> k;
40     vector<int> a(n);
41     for (int i = 0; i < n; i++) {
42         cin >> a[i];
43     }
44     vector<int> p(n);
45     for (int i = 0, j = 0; i < n; i++) {
46         while (a[i] - a[j] > k) {
47             j++;
48         }
49         p[i] = j;
50     }
51     vector<Z> dp(n);
52     vector<Z> sum(n + 1);
53     for (int i = 0; i < n; i++) {
54         Z val = 0;
55         val += comb.fac(n - 1) * comb.invfac(n - p[i] - 1);
56         val += sum[p[i]] * comb.invfac(n - p[i] - 1);
57         if (p[i] < i) {
58             if (i == n - 1) {
59                 val = 0;
60             } else {
61                 val *= comb.fac(n - p[i] - 2) * comb.invfac(n - i - 2);
62             }
63         }
64         dp[i] = val;
65         if (i < n - 1) {
66             sum[i + 1] = sum[i] + dp[i] * comb.fac(n - i - 2);
67         }
68     }
69     cout << dp[n - 1] << "\n";
70     return 0;
71 }

```

**836: R3D3s Summer Adventure**

- Time limit: 1 second

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

R3D3 spent some time on an internship in MDCS. After earning enough money, he decided to go on a holiday somewhere far, far away. He enjoyed suntanning, drinking alcohol-free cocktails and going to concerts of popular local bands. While listening to “The White Buttons” and their hit song “Dacan the Baker”, he met another robot for whom he was sure is the love of his life. Well, his summer, at least. Anyway, R3D3 was too shy to approach his potential soulmate, so he decided to write her a love letter. However, he stumbled upon a problem. Due to a terrorist threat, the Intergalactic Space Police was monitoring all letters sent in the area. Thus, R3D3 decided to invent his own alphabet, for which he was sure his love would be able to decipher.

There are  $n$  letters in R3D3's alphabet, and he wants to represent each letter as a sequence of '0' and '1', so that no letters sequence is a prefix of another letter's sequence. Since the Intergalactic Space Communications Service has lately introduced a tax for invented alphabets, R3D3 must pay a certain amount of money for each bit in his alphabets code (check the sample test for clarifications). He is too lovestruck to think clearly, so he asked you for help.

Given the costs  $c_0$  and  $c_1$  for each '0' and '1' in R3D3's alphabet, respectively, you should come up with a coding for the alphabet (with properties as above) with minimum total cost.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int N, c0, c1;
8     cin >> N >> c0 >> c1;
9     if (c1 < c0) {
10         swap(c0, c1);
11     }
12     if (c0 == 0) {
13         i64 ans = 1LL * (N - 1) * c1;
14         cout << ans << "\n";
15         return 0;
16     }
17     i64 lo = 0, hi = 1E10;
18     auto get = [&](i64 x, bool prec = false) {
19         i64 cnt = 1;
20         i64 sum = 0;
21         for (i64 i = 0; i * c1 <= x; i++) {
22             i64 j = (x - i * c1) / c0;
23             i64 v = 1;
24             i64 n = i + j + 1, m = j;

```

```

25         if (m > n - m) {
26             m = n - m;
27         }
28         for (i64 j = 1; j <= m; j++) {
29             v = v * (n - j + 1) / j;
30             if (v > N && !prec) {
31                 break;
32             }
33         }
34         cnt += v;
35         sum += v * c1 * i;
36         if (j > 0) {
37             v = 1;
38             n = i + j + 1, m = j - 1;
39             if (m > n - m) {
40                 m = n - m;
41             }
42             for (i64 j = 1; j <= m; j++) {
43                 v = v * (n - j + 1) / j;
44                 if (v > N && !prec) {
45                     break;
46                 }
47             }
48             sum += v * c0 * (i + 1);
49         }
50         if (cnt > N && !prec) {
51             break;
52         }
53     }
54     sum += 1LL * (N - 1) * (c0 + c1);
55     return pair(cnt, sum);
56 };
57 while (lo < hi) {
58     i64 x = (lo + hi) / 2;
59     if (get(x).first >= N) {
60         hi = x;
61     } else {
62         lo = x + 1;
63     }
64 }
auto [cnt, sum] = get(lo, true);
// cerr << lo << " " << cnt << " " << sum << "\n";
sum -= (cnt - N) * lo;
cout << sum << "\n";
return 0;
}

```

### 837: Another n-dimensional chocolate bar

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Vasya and his friends want to cut a chocolate bar to get at least  $k$  pieces, while Vasya wants to maximize the volume of the smallest of them. It is possible to cut the chocolate bar only at the junction of the lobules, and each incision must pass through the entire chocolate bar along some hyperplane involved

in the formation of lobules. Only after making all the cuts, Vasya disassembles the chocolate into pieces.

More formally, Vasya wants to choose the numbers  $b_1, b_2, \dots, b_n$  ( $1 \leq b_i \leq a_i$ ) - the number of parts into which Vasya will cut the chocolate bar along each dimension. The condition  $b_1 \cdot b_2 \cdot \dots \cdot b_n \geq k$  must be met to get at least  $k$  pieces after all cuts. It can be noted that with optimal cutting with such parameters, the minimum piece will contain  $\lfloor \frac{a_1}{b_1} \rfloor \cdots \lfloor \frac{a_n}{b_n} \rfloor$  slices, and its volume will be equal to  $\lfloor \frac{a_1}{b_1} \rfloor \cdots \lfloor \frac{a_n}{b_n} \rfloor \cdot \frac{1}{a_1 a_2 \cdots a_n}$ .

Vasya wants to get the maximum possible value of the volume of the minimum piece multiplied by  $k$ , that is, he wants to maximize the number of  $\lfloor \frac{a_1}{b_1} \rfloor \cdots \lfloor \frac{a_n}{b_n} \rfloor \cdot \frac{1}{a_1 a_2 \cdots a_n} \cdot k$ . Help him with this.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     cout << fixed << setprecision(10);
8     int n, k;
9     cin >> n >> k;
10    k -= 1;
11    vector<int> a(n);
12    for (int i = 0; i < n; i++) {
13        cin >> a[i];
14    }
15    vector<int> b;
16    for (int l = 1, r; l <= k; l = r + 1) {
17        int v = k / l;
18        r = k / v;
19        b.push_back(v);
20    }
21    b.push_back(0);
22    int m = b.size();
23    vector<int> id(k + 1);
24    for (int i = 0; i < m; i++) {
25        id[b[i]] = i;
26    }
27    vector<double> dp(m);
28    dp[0] = k + 1;
29    for (auto x : a) {
30        for (int i = m - 1; i >= 0; i--) {
31            double v = dp[i];
32            int q = b[i];
33            for (int l = 1, r; l <= q; l = r + 1) {
34                int t = q / l;
35                r = q / t;
36                dp[id[t]] = max(dp[id[t]], v * (x / l) / x);
37            }
38            dp[m - 1] = max(dp[m - 1], v * (x / (q + 1)) / x);
39        }
40    }
41    cout << dp[m - 1] << "\n";
42    return 0;
43 }
```

### 838: Halve or Subtract

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have an array of positive integers  $a_1, a_2, \dots, a_n$ , of length  $n$ . You are also given a positive integer  $b$ .

You are allowed to perform the following operations (possibly several) times in any order:

Choose some  $1 \leq i \leq n$ , and replace  $a_i$  with  $\lceil \frac{a_i}{2} \rceil$ . Here,  $\lceil x \rceil$  denotes the smallest integer not less than  $x$ .

Choose some  $1 \leq i \leq n$ , and replace  $a_i$  with  $\max(a_i - b, 0)$ .

However, you must also follow these rules:

You can perform at most  $k_1$  operations of type 1 in total.

You can perform at most  $k_2$  operations of type 2 in total.

For all  $1 \leq i \leq n$ , you can perform at most 1 operation of type 1 on element  $a_i$ .

For all  $1 \leq i \leq n$ , you can perform at most 1 operation of type 2 on element  $a_i$ .

The cost of an array is the sum of its elements. Find the minimum cost of  $a$  you can achieve by performing these operations.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, b, k1, k2;
6     cin >> n >> b >> k1 >> k2;
7     i64 sum = 0;
8     vector<int> a(n);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11         sum += a[i];
12     }
13     i64 ans = 0;
14     sort(a.begin(), a.end(), greater());
15     vector<i64> f(k1 + 1, vector<i64>(k2 + 1));
16     int mid = 0;
17     while (mid < n && a[mid] >= b) {
18         mid++;
19     }
20     i64 res = 0;
21     for (int i = 0; i <= k1; i++) {

```

```

22         if (i > 0 && i <= mid) {
23             res += a[i - 1] / 2;
24         }
25         int t = min(i, mid);
26         i64 v = res;
27         for (int j = 0; j <= k2; j++) {
28             if (j > 0) {
29                 if (t + j <= mid) {
30                     v += b;
31                 } else if (j <= mid) {
32                     v += min(b, (a[t + j - mid - 1] + 1) / 2);
33                 }
34             }
35             f[i][j] = v;
36         }
37     }
38     res = 0;
39     for (int i = 0; i <= min(k2, n - mid); i++) {
40         if (i > 0) {
41             res += a[mid + i - 1];
42         }
43         i64 v = res;
44         for (int j = 0; j <= min(k1, n - i - mid); j++) {
45             if (j > 0) {
46                 v += a[mid + i + j - 1] / 2;
47             }
48             ans = max(ans, v + f[k1 - j][k2 - i]);
49         }
50     }
51     ans = sum - ans;
52     cout << ans << "\n";
53 }
54 int main() {
55     ios::sync_with_stdio(false);
56     cin.tie(nullptr);
57     int t;
58     cin >> t;
59     while (t--) {
60         solve();
61     }
62     return 0;
63 }
```

### 839: Serval and Brain Power

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Serval loves Brain Power and his brain power problem.

Serval defines that a string  $T$  is powerful iff  $T$  can be obtained by concatenating some string  $T'$  multiple

times. Formally speaking,  $T$  is powerful iff there exist a string  $T'$  and an integer  $k \geq 2$  such that

$$T = \underbrace{T' + T' + \cdots + T'}_{k \text{ times}}$$

For example, gogogo is powerful because it can be obtained by concatenating go three times, but power is not powerful.

Serval has a string  $S$  consists of lowercase English letters. He is curious about the longest powerful subsequence of  $S$ , and he only needs you to find out the length of it. If all the non-empty subsequences of  $S$  is not powerful, the answer is considered to be 0.

A string  $a$  is a subsequence of a string  $b$  if  $a$  can be obtained from  $b$  by the deletion of several (possibly, zero or all) characters.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void update(int &a, int b) {
5     if (a < b) {
6         a = b;
7     }
8 }
9 int main() {
10    ios::sync_with_stdio(false);
11    cin.tie(nullptr);
12    string s;
13    cin >> s;
14    int n = s.size();
15    int ans = 0;
16    array<int, 6> p;
17    for (int i = 0; i <= 5; i++) {
18        p[i] = n * i / 5;
19    }
20    for (int k = 0; k < 5; k++) {
21        int gap = p[k + 1] - p[k];
22        for (int mask = 1; mask < (1 << gap); mask++) {
23            string t;
24            for (int i = 0; i < gap; i++) {
25                if (mask >> i & 1) {
26                    t += s[p[k] + i];
27                }
28            }
29            int len = 0;
30            for (int i = 0; i < n; i++) {
31                if (s[i] == t[len % t.size()]) {
32                    len++;
33                }
34            }
35            if (len < 2 * t.size()) {
36                continue;
37            }
38            ans = max(ans, len - int(len % t.size())));
}

```

```

39         }
40     }
41     for (int x = 1; x < n; x++) {
42         vector<vector<int>> dp(x + 1, vector<int>(n - x + 1));
43         for (int i = 0; i <= x; i++) {
44             for (int j = 0; j <= n - x; j++) {
45                 if (i < x) {
46                     update(dp[i + 1][j], dp[i][j]);
47                 }
48                 if (j < n - x) {
49                     update(dp[i][j + 1], dp[i][j]);
50                 }
51                 if (i < x && j < n - x && s[i] == s[x + j]) {
52                     update(dp[i + 1][j + 1], dp[i][j] + 1);
53                 }
54             }
55         }
56         ans = max(ans, dp[x][n - x] * 2);
57     }
58     for (int x = 1; x < n; x++) {
59         for (int y = x + 1; y < n; y++) {
60             vector<vector<vector<int>>> dp(x + 1, vector<vector<int>>(y - x + 1, vector<int>(n - y + 1)));
61             for (int i = 0; i <= x; i++) {
62                 for (int j = 0; j <= y - x; j++) {
63                     for (int k = 0; k <= n - y; k++) {
64                         if (i < x) {
65                             update(dp[i + 1][j][k], dp[i][j][k]);
66                         }
67                         if (j < y - x) {
68                             update(dp[i][j + 1][k], dp[i][j][k]);
69                         }
70                         if (k < n - y) {
71                             update(dp[i][j][k + 1], dp[i][j][k]);
72                         }
73                         if (i < x && j < y - x && k < n - y && s[i] == s[x + j] && s[i] == s[y + k]) {
74                             update(dp[i + 1][j + 1][k + 1], dp[i][j][k] + 1);
75                         }
76                     }
77                 }
78             }
79             ans = max(ans, dp[x][y - x][n - y] * 3);
80         }
81     }
82     cout << ans << "\n";
83     return 0;
84 }
```

### 840: Parmigiana With Seafood

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The “Parmigiana di melanzane” is a typical Italian dish. Alessandro and Bianca have very different tastes when it comes to it: Alessandro loves to eat Parmigiana with seafood, but Bianca thinks it is

an atrocity! To decide which ingredients to include in the dish they prepare, they play the following game.

There are  $n$  possible ingredients, labeled from 1 to  $n$ . The higher the label, the closer the ingredient is to being seafood. The ingredients are connected by  $n - 1$  edges, in such a way as to form a tree. Alessandro and Bianca take turns, with Alessandro going first. They alternately choose a terminal ingredient  $x$ , that is an ingredient currently connected to at most one other ingredient, and remove it from the tree. If the terminal ingredient  $x$  was chosen by Alessandro, it goes in the recipe; if it was chosen by Bianca, it is discarded.

The taste of the Parmigiana is measured as the maximum label of an ingredient in the recipe. Alessandro wants to maximize the taste, while Bianca wants to minimize the taste. If both play optimally, what is the taste of the Parmigiana?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<vector<int>> adj(n);
10    for (int i = 1; i < n; i++) {
11        int u, v;
12        cin >> u >> v;
13        u--, v--;
14        adj[u].push_back(v);
15        adj[v].push_back(u);
16    }
17    if (n % 2 == 0) {
18        cout << n << "\n";
19        return 0;
20    }
21    int lo = 0, hi = n - 1;
22    auto check = [&](int v) {
23        for (int x = 0; x < n; x++) {
24            if (adj[x].size() == 1 && x >= v) {
25                return true;
26            }
27        }
28        bool have[2] {};
29        vector<int> dep(n), siz(n);
30        bool ok = false;
31        auto dfs = [&](auto self, int x, int p) -> void {
32            siz[x] = (x >= v);
33            if (x >= v) {
34                have[dep[x] % 2] = true;
35            }
36            for (auto y : adj[x]) {
37                if (y == p) {
38                    continue;
39                }

```

```

40         dep[y] = dep[x] + 1;
41         self(self, y, x);
42         siz[x] += siz[y];
43     }
44 }
45 dfs(dfs, 0, -1);
46 if (have[0] && have[1]) {
47     return true;
48 }
49 auto dfs1 = [&](auto self, int x, int p) -> void {
50     int cnt = 0;
51     for (auto y : adj[x]) {
52         if (y == p) {
53             continue;
54         }
55         self(self, y, x);
56         cnt += (siz[y] > 0);
57     }
58     cnt += (siz[0] - siz[x] > 0);
59     if (cnt >= 3 && have[dep[x] % 2]) {
60         ok = true;
61     }
62 };
63 dfs1(dfs1, 0, -1);
64 return ok;
65 };
66 while (lo < hi) {
67     int x = (lo + hi + 1) / 2;
68     if (!check(x)) {
69         hi = x - 1;
70     } else {
71         lo = x;
72     }
73 }
74 int ans = lo + 1;
75 cout << ans << "\n";
76 return 0;
77 }

```

### 841: Edge Queries

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an undirected, connected graph of  $n$  nodes and  $m$  edges. All nodes  $u$  of the graph satisfy the following:

Let  $S_u$  be the set of vertices in the longest simple cycle starting and ending at  $u$ .

Let  $C_u$  be the union of the sets of vertices in any simple cycle starting and ending at  $u$ .

$$S_u = C_u.$$

You need to answer  $q$  queries.

For each query, you will be given node  $a$  and node  $b$ . Out of all the edges that belong to any simple path from  $a$  to  $b$ , count the number of edges such that if you remove that edge,  $a$  and  $b$  are reachable from each other.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct BlockCutTree;
5 # struct HLD;
6 int main() {
7     ios::sync_with_stdio(false);
8     cin.tie(nullptr);
9     int n, m;
10    cin >> n >> m;
11    BlockCutTree g(n);
12    vector<pair<int, int>> e;
13    for (int i = 0; i < m; i++) {
14        int u, v;
15        cin >> u >> v;
16        u--, v--;
17        g.addEdge(u, v);
18        e.emplace_back(u, v);
19    }
20    auto [cnt, edges] = g.work();
21    vector<int> siz(n + cnt);
22    HLD t(n + cnt);
23    for (auto [x, y] : edges) {
24        t.addEdge(x, y);
25    }
26    t.work();
27    for (auto [x, y] : e) {
28        if (t.dep[x] > t.dep[y]) {
29            siz[t.parent[x]]++;
30        } else {
31            siz[t.parent[y]]++;
32        }
33    }
34    for (int i = n; i < n + cnt; i++) {
35        if (siz[i] == 1) {
36            siz[i] = 0;
37        }
38    }
39    auto dfs = [&](auto self, int x) -> void {
40        for (auto y : t.adj[x]) {
41            siz[y] += siz[x];
42            self(self, y);
43        }
44    };
45    dfs(dfs, 0);
46    int q;
47    cin >> q;
48    for (int i = 0; i < q; i++) {
49        int a, b;
50        cin >> a >> b;
51        a--, b--;
52        int ans = siz[a] + siz[b];
53        int l = t.lca(a, b);
54        ans -= siz[l];

```

```

55         if (l > 0) {
56             ans -= siz[t.parent[l]];
57         }
58         cout << ans << "\n";
59     }
60     return 0;
61 }
```

**greedy****842: Group Division**

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In the 31st lyceum, there were two groups of olympiad participants: computer science and mathematics. The number of computer scientists was  $n_1$ , and the number of mathematicians was  $n_2$ . It is not known for certain who belonged to which group, but it is known that there were friendly connections between some pairs of people (these connections could exist between a pair of people from the same group or from different groups).

The connections were so strong that even if one person is removed along with all their friendly connections, any pair of people still remains acquainted either directly or through mutual friends.

<sup>†</sup> More formally, two people  $(x, y)$  are acquainted in the following case: there are people  $a_1, a_2, \dots, a_n$  ( $1 \leq a_i \leq n_1 + n_2$ ) such that the following conditions are simultaneously met:

- Person  $x$  is directly acquainted with  $a_1$ .
- Person  $a_n$  is directly acquainted with  $y$ .
- Person  $a_i$  is directly acquainted with  $a_{i+1}$  for any ( $1 \leq i \leq n - 1$ ).

The teachers were dissatisfied with the fact that computer scientists were friends with mathematicians and vice versa, so they decided to divide the students into two groups in such a way that the following two conditions are met:

- There were  $n_1$  people in the computer science group, and  $n_2$  people in the mathematics group.
- Any pair of computer scientists should be acquainted (acquaintance involving mutual friends, who must be from the same group as the people in the pair, is allowed), the same should be true for mathematicians.

Help them solve this problem and find out who belongs to which group.

## Problem: [link](#)

## Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n1, n2, m;
6     cin >> n1 >> n2 >> m;
7     int n = n1 + n2;
8     vector<vector<int>> adj(n);
9     for (int i = 0; i < m; i++) {
10         int u, v;
11         cin >> u >> v;
12         u--, v--;
13         adj[u].push_back(v);
14         adj[v].push_back(u);
15     }
16     int s = 0, t = n - 1;
17     vector<int> pre(n), low(n), p(n);
18     int cur = 1;
19     pre[s] = 1;
20     vector<int> preorder;
21     auto dfs = [&](auto self, int x) -> void {
22         pre[x] = ++cur;
23         low[x] = x;
24         for (auto y : adj[x]) {
25             if (pre[y] == 0) {
26                 preorder.push_back(y);
27                 self(self, y);
28                 p[y] = x;
29                 if (pre[low[y]] < pre[low[x]]) {
30                     low[x] = low[y];
31                 }
32             } else if (pre[y] != 0 && pre[y] < pre[low[x]]) {
33                 low[x] = y;
34             }
35         }
36     };
37     dfs(dfs, t);
38     vector<int> sign(n, -1);
39     vector<int> l(n), r(n);
40     r[s] = t;
41     l[t] = s;
42     for (auto v : preorder) {
43         if (sign[low[v]] == -1) {
44             l[v] = l[p[v]];
45             r[l[v]] = v;
46             l[p[v]] = v;
47             r[v] = p[v];
48             sign[p[v]] = 1;
49         } else {
50             r[v] = r[p[v]];
51             l[r[v]] = v;
52             r[p[v]] = v;
53             l[v] = p[v];
54             sign[p[v]] = -1;
55         }
56     }
57     vector<int> a;
58     for (int i = 0, x = s; i < n; x = r[x], i++) {

```

```

59         a.push_back(x);
60         cout << x + 1 << " \n"[i == n1 - 1 || i == n - 1];
61     }
62 }
63 int main() {
64     ios::sync_with_stdio(false);
65     cin.tie(nullptr);
66     int t;
67     cin >> t;
68     while (t--) {
69         solve();
70     }
71     return 0;
72 }
```

### 843: Alice and Recoloring 2

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The difference between the versions is in the costs of operations. Solution for one version won't work for another!

Alice has a grid of size  $n \times m$ , initially all its cells are colored white. The cell on the intersection of  $i$ -th row and  $j$ -th column is denoted as  $(i, j)$ . Alice can do the following operations with this grid:

Choose any subrectangle containing cell  $(1, 1)$ , and flip the colors of all its cells. (Flipping means changing its color from white to black or from black to white). This operation costs 1 coin.

Choose any subrectangle containing cell  $(1, 1)$ , and flip the colors of all its cells. (Flipping means changing its color from white to black or from black to white).

This operation costs 1 coin.

Choose any subrectangle containing cell  $(n, 1)$ , and flip the colors of all its cells. This operation costs 3 coins.

Choose any subrectangle containing cell  $(n, 1)$ , and flip the colors of all its cells.

This operation costs 3 coins.

Choose any subrectangle containing cell  $(1, m)$ , and flip the colors of all its cells. This operation costs 4 coins.

Choose any subrectangle containing cell  $(1, m)$ , and flip the colors of all its cells.

This operation costs 4 coins.

Choose any subrectangle containing cell  $(n, m)$ , and flip the colors of all its cells. This operation costs 2 coins.

Choose any subrectangle containing cell  $(n, m)$ , and flip the colors of all its cells.

This operation costs 2 coins.

As a reminder, subrectangle is a set of all cells  $(x, y)$  with  $x_1 \leq x \leq x_2, y_1 \leq y \leq y_2$  for some  $1 \leq x_1 \leq x_2 \leq n, 1 \leq y_1 \leq y_2 \leq m$ .

Alice wants to obtain her favorite coloring with these operations. What's the smallest number of coins that she would have to spend? It can be shown that it's always possible to transform the initial grid into any other.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m;
8     cin >> n >> m;
9     vector<string> s(n);
10    for (int i = 0; i < n; i++) {
11        cin >> s[i];
12    }
13    vector a(n, vector<int>(m));
14    for (int i = 0; i < n; i++) {
15        for (int j = 0; j < m; j++) {
16            a[i][j] = (s[i][j] == 'B');
17        }
18    }
19    for (int i = 0; i < n; i++) {
20        for (int j = 0; j < m - 1; j++) {
21            a[i][j] ^= a[i][j + 1];
22        }
23    }
24    for (int i = 0; i < n - 1; i++) {
25        for (int j = 0; j < m; j++) {
26            a[i][j] ^= a[i + 1][j];
27        }
28    }
29    int ans = 0;
30    for (int i = 0; i < n; i++) {
31        for (int j = 0; j < m; j++) {
32            ans += a[i][j];
33        }
34    }
35    int matching = 0;
36    vector<vector<int>> adj(n);
37    for (int i = 0; i < n - 1; i++) {
38        for (int j = 0; j < m - 1; j++) {
39            if (a[i][j] && a[i][m - 1] && a[n - 1][j]) {
40                adj[i].push_back(j);
41            }
42        }

```

```

43     }
44     if (a[n - 1][m - 1]) {
45         adj[n - 1].push_back(m - 1);
46     }
47     vector<int> yx(m, -1);
48     vector<bool> vis(n, false);
49     auto find = [&](auto self, int x) -> bool {
50         vis[x] = true;
51         for (auto y : adj[x]) {
52             if (yx[y] == -1 || (!vis[yx[y]] && self(self, yx[y]))) {
53                 yx[y] = x;
54                 return true;
55             }
56         }
57         return false;
58     };
59     for (int i = 0; i < n; i++) {
60         if (find(find, i)) {
61             matching += 1;
62             vis.assign(n, false);
63         }
64     }
65     ans -= matching / 2 * 2;
66     cout << ans << "\n";
67     return 0;
68 }
```

## 844: Most Different Tree

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Given a tree with  $n$  vertices rooted at vertex 1, denote it as  $G$ . Also denote  $P(G)$  as the multiset of subtrees of all vertices in tree  $G$ . You need to find a tree  $G'$  of size  $n$  rooted at vertex 1 such that the number of subtrees in  $P(G')$  that are isomorphic to any subtree in  $P(G)$  is minimized.

A subtree of vertex  $v$  is a graph that contains all vertices for which vertex  $v$  lies on the path from the root of the tree to itself, as well as all edges between these vertices.

Two rooted trees are considered isomorphic if it is possible to relabel the vertices of one of them so that it becomes equal to the other, with the root of the first tree receiving the number of the root of the second tree.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 map<vector<int>, int> tree;
```

```
5  vector<int> siz;
6  vector<vector<int>> vec;
7  int f(const vector<int> &a) {
8      if (tree.count(a)) {
9          return tree[a];
10     }
11     int v = tree.size();
12     int s = 1;
13     for (auto x : a) {
14         s += siz[x];
15     }
16     siz.push_back(s);
17     vec.push_back(a);
18     return tree[a] = v;
19 }
20 int main() {
21     ios::sync_with_stdio(false);
22     cin.tie(nullptr);
23     int n;
24     cin >> n;
25     vector<vector<int>> adj(n);
26     for (int i = 1; i < n; i++) {
27         int u, v;
28         cin >> u >> v;
29         u--, v--;
30         adj[u].push_back(v);
31         adj[v].push_back(u);
32     }
33     set<int> s;
34     vector<int> g(n);
35     auto dfs = [&](auto self, int x, int p) -> void {
36         vector<int> a;
37         for (auto y : adj[x]) {
38             if (y == p) {
39                 continue;
40             }
41             self(self, y, x);
42             a.push_back(g[y]);
43         }
44         sort(a.begin(), a.end());
45         g[x] = f(a);
46         s.insert(g[x]);
47     };
48     dfs(dfs, 0, -1);
49     vector<int> a;
50     for (int cnt = 1; cnt <= n; cnt++) {
51         vector<int> b, c;
52         int x = -1;
53         auto dfs = [&](auto self, int sum, int lst) {
54             if (sum == 0) {
55                 int v = f(b);
56                 c.push_back(v);
57                 if (!s.count(v)) {
58                     x = v;
59                 }
60                 return;
61             }
62             for (int i = lst; i < a.size(); i++) {
63                 int y = a[i];
64                 if (sum >= siz[y]) {
65                     b.push_back(y);
66                     self(self, sum - siz[y], i);
67                     b.pop_back();
68                     if (x != -1) {
69                         if (sum - siz[y] >= 0) {
70                             c.push_back(sum - siz[y]);
71                         }
72                     }
73                 }
74             }
75         };
76         dfs(dfs, 0, -1);
77         a = b;
78     }
79 }
```

```

69             return;
70         }
71     }
72 }
73 };
74 dfs(dfs, cnt - 1, 0);
75 if (x != -1) {
76     int cur = 1;
77     for (int i = n; i > cnt; i--) {
78         cout << cur << " " << cur + 1 << "\n";
79         cur++;
80     }
81     auto dfs = [&](auto self, int x, int y) -> void {
82         for (auto v : vec[y]) {
83             int u = ++cur;
84             cout << x << " " << u << "\n";
85             self(self, u, v);
86         }
87     };
88     dfs(dfs, cur, x);
89     return 0;
90 }
91 a.insert(a.end(), c.begin(), c.end());
92 sort(a.begin(), a.end());
93 }
94 for (int i = 0; i < n; i++) {
95     for (auto j : adj[i]) {
96         if (j > i) {
97             cout << i + 1 << " " << j + 1 << "\n";
98         }
99     }
100 }
101 return 0;
102 }
```

### 845: Flower-like Pseudotree

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

A pseudotree is a connected graph which has exactly one cycle and no self-loops. Note that a pseudotree may contain multiple-edges. It can be shown that a pseudotree with  $n$  vertices always contains  $n$  edges.

After deleting all edges on the cycle in the pseudotree, a forest<sup>†</sup> will be formed. It can be shown that each tree in the forest will contain exactly one vertex which is on cycle before removing the edges. If all trees in the forest have the same depth<sup>‡</sup> when picking the vertex on cycle as root, we call the original pseudotree flower-like.

Our friend sszcdjr, had a flower-like pseudotree with  $n$  vertices and  $n$  edges. However, he forgot all the edges in the pseudotree. Fortunately, he still remembers the degrees of vertices. Specifically, the

degree of the  $i$ -th vertex is  $d_i$ .

You have to help sszcdjr construct a possible flower-like pseudotree with  $n$  vertices, where the degree of the  $i$ -th vertex is exactly  $d_i$ , or tell him that it is impossible.

$\dagger$  A forest is a graph in which all connectivity components are trees. A connected graph without cycles and self-loops is called a tree.

$\ddagger$  The depth of a tree with a root is the maximum distance from the root to the vertex of this tree.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> d(n);
8     for (int i = 0; i < n; i++) {
9         cin >> d[i];
10    }
11    if (accumulate(d.begin(), d.end(), 0LL) != 2 * n) {
12        cout << "No\n";
13        return;
14    }
15    vector<pair<int, int>> edges;
16    if (count(d.begin(), d.end(), 2) == n) {
17        for (int i = 0; i < n; i++) {
18            edges.emplace_back(i, (i + 1) % n);
19        }
20    } else if (count(d.begin(), d.end(), 2) == 0) {
21        vector<int> a;
22        for (int i = 0; i < n; i++) {
23            if (d[i] > 1) {
24                a.push_back(i);
25            }
26        }
27        if (a.size() == 1) {
28            cout << "No\n";
29            return;
30        }
31        for (int i = 0, x = 0; i < a.size(); i++) {
32            edges.emplace_back(a[i], a[(i + 1) % a.size()]);
33            for (int j = 2; j < d[a[i]]; j++) {
34                while (d[x] != 1) {
35                    x++;
36                }
37                edges.emplace_back(a[i], x);
38                x++;
39            }
40        }
41    } else {
42        vector<int> p(n);
43        iota(p.begin(), p.end(), 0);
44        sort(p.begin(), p.end(),
45              [&](int i, int j) {
46                  return d[i] > d[j];
47              });

```

```

48     for (int i = 0; i < 4; i++) {
49         if (i >= n || d[p[i]] < (i < 2 ? 3 : 2)) {
50             cout << "No\n";
51             return;
52         }
53     }
54     int leaf = 0;
55     while (d[p[leaf]] != 1) {
56         leaf++;
57     }
58     edges.emplace_back(p[0], p[1]);
59     edges.emplace_back(p[0], p[1]);
60     d[p[0]] -= 2;
61     d[p[1]] -= 2;
62     vector<int> u[2];
63     for (int i = 2; i < leaf; i++) {
64         d[p[i]] -= 1;
65         u[i % 2].push_back(p[i]);
66     }
67     vector<int> a;
68     for (int t = 0; t < 2; t++) {
69         int i = 0;
70         if (t == 0 && u[0].size() > u[1].size()) {
71             if (d[p[0]] >= 2) {
72                 edges.emplace_back(p[0], u[0][0]);
73                 edges.emplace_back(p[0], u[0][1]);
74                 d[p[0]] -= 2;
75                 i = 1;
76             } else if (d[u[0][0]] >= 2) {
77                 edges.emplace_back(p[0], u[0][0]);
78                 d[p[0]]--;
79                 if (u[0].size() <= 2) {
80                     cout << "No\n";
81                     return;
82                 }
83                 edges.emplace_back(u[0][0], u[0][1]);
84                 edges.emplace_back(u[0][0], u[0][2]);
85                 d[u[0][0]] -= 2;
86                 i = 2;
87             } else {
88                 cout << "No\n";
89                 return;
90             }
91         } else {
92             edges.emplace_back(p[t], u[t][0]);
93             d[p[t]]--;
94         }
95         while (i + 1 < u[t].size()) {
96             edges.emplace_back(u[t][i], u[t][i + 1]);
97             d[u[t][i]]--;
98             i++;
99         }
100    }
101    for (int i = 0; i < leaf; i++) {
102        for (int j = 0; j < d[p[i]]; j++) {
103            a.push_back(p[i]);
104        }
105    }
106    assert(a.size() == n - leaf);
107    for (int i = 0; i < a.size(); i++) {
108        edges.emplace_back(a[i], p[leaf + i]);
109    }
110 }
111 cout << "Yes\n";

```

```

112     for (auto [x, y] : edges) {
113         cout << x + 1 << " " << y + 1 << "\n";
114     }
115 }
116 int main() {
117     ios::sync_with_stdio(false);
118     cin.tie(nullptr);
119     int t;
120     cin >> t;
121     while (t--) {
122         solve();
123     }
124     return 0;
125 }
```

## 846: Three Swaps

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Xenia the horse breeder has  $n$  ( $n > 1$ ) horses that stand in a row. Each horse has its own unique number. Initially, the  $i$ -th left horse has number  $i$ . That is, the sequence of numbers of horses in a row looks as follows (from left to right):  $1, 2, 3, \dots, n$ .

Xenia trains horses before the performance. During the practice sessions, she consistently gives them commands. Each command is a pair of numbers  $l, r$  ( $1 \leq l < r \leq n$ ). The command  $l, r$  means that the horses that are on the  $l$ -th,  $(l+1)$ -th,  $(l+2)$ -th, ...,  $r$ -th places from the left must be rearranged. The horses that initially stand on the  $l$ -th and  $r$ -th places will swap. The horses on the  $(l+1)$ -th and  $(r-1)$ -th places will swap. The horses on the  $(l+2)$ -th and  $(r-2)$ -th places will swap and so on. In other words, the horses that were on the segment  $[l, r]$  change their order to the reverse one.

For example, if Xenia commanded  $l = 2, r = 5$ , and the sequence of numbers of horses before the command looked as  $(2, 1, 3, 4, 5, 6)$ , then after the command the sequence will be  $(2, 5, 4, 3, 1, 6)$ .

We know that during the practice Xenia gave at most three commands of the described form. You have got the final sequence of numbers of horses by the end of the practice. Find what commands Xenia gave during the practice. Note that you do not need to minimize the number of commands in the solution, find any valid sequence of at most three commands.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
```

```

2  using namespace std;
3  using i64 = long long;
4  int main() {
5      ios::sync_with_stdio(false);
6      cin.tie(nullptr);
7      int n;
8      cin >> n;
9      vector<int> p(n);
10     for (int i = 0; i < n; i++) {
11         cin >> p[i];
12         p[i]--;
13     }
14     vector<int> goal(n);
15     iota(goal.begin(), goal.end(), 0);
16     vector<pair<int, int>> ans;
17     function<void(int, vector<int>)> dfs = [&](int t, vector<int> p) {
18         if (t == 3) {
19             if (p == goal) {
20                 cout << ans.size() << "\n";
21                 reverse(ans.begin(), ans.end());
22                 for (auto [l, r] : ans) {
23                     cout << l << " " << r << "\n";
24                 }
25                 exit(0);
26             }
27             return;
28         }
29         vector<int> invp(n);
30         for (int i = 0; i < n; i++) {
31             invp[p[i]] = i;
32         }
33         vector<int> e;
34         int cnt = 0;
35         for (int i = 0; i <= n; i++) {
36             if (!i || i == n || abs(p[i] - p[i - 1]) != 1) {
37                 cnt++;
38                 e.push_back(i);
39                 if (i) {
40                     e.push_back(i - 1);
41                 }
42                 if (i < n) {
43                     e.push_back(i + 1);
44                 }
45                 for (auto j : {i - 1, i}) {
46                     if (0 <= j && j < n) {
47                         for (auto k : {p[j] - 1, p[j] + 1}) {
48                             if (0 <= k && k < n) {
49                                 if (invp[k] < j) {
50                                     e.push_back(invp[k]);
51                                 } else {
52                                     e.push_back(invp[k] + 1);
53                                 }
54                             }
55                         }
56                     }
57                 }
58             }
59         }
60         if (cnt > (4 - t) * 2) {
61             return;
62         }
63         sort(e.begin(), e.end());
64         e.erase(unique(e.begin(), e.end()), e.end());
65         int m = e.size();

```

```

66     for (int l = 0; l < m; l++) {
67         for (int r = l; r < m; r++) {
68             auto q = p;
69             if (e[l] + 1 < e[r]) {
70                 reverse(q.begin() + e[l], q.begin() + e[r]);
71                 ans.emplace_back(e[l] + 1, e[r]);
72             }
73             dfs(t + 1, q);
74             if (e[l] + 1 < e[r]) {
75                 ans.pop_back();
76             }
77         }
78     };
79 }
80 dfs(0, p);
81 return 0;
82 }
```

**847: GCD Master (hard version)**

- Time limit: 3 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. The only difference between the two versions is the constraint on  $m$ . You can make hacks only if both versions of the problem are solved.

You are given an array  $a$  of length  $n$  and two integers  $m$  and  $k$ . Each element in  $a$  satisfies  $1 \leq a_i \leq m$ .

In one operation, you choose two indices  $i$  and  $j$  such that  $1 \leq i < j \leq |a|$ , then append  $\gcd(a_i, a_j)$  to the back of the array and delete  $a_i$  and  $a_j$  from the array. Note that the length of the array decreases by one after this operation.

Find the maximum possible sum of the array after performing exactly  $k$  operations.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 using i128 = __int128;
5 ostream &operator<<(ostream &os, i128 n) {
6     string s;
7     while (n) {
8         s += '0' + n % 10;
9         n /= 10;
10    }
11    reverse(s.begin(), s.end());
12    return os << s;
13 }
14 void solve() {
```

```

15     int n, k;
16     i64 m;
17     cin >> n >> m >> k;
18     vector<i64> a(n);
19     for (int i = 0; i < n; i++) {
20         cin >> a[i];
21     }
22     sort(a.begin(), a.end());
23     vector<int> suf(n + 1);
24     vector<i128> ssuf(n + 1);
25     vector<i128> sum(n + 1);
26     for (int i = n - 1; i >= 0; i--) {
27         suf[i] = suf[i + 1] + (i == n - 1 || a[i] != a[i + 1]);
28         ssuf[i] = ssuf[i + 1] + (i == n - 1 || a[i] != a[i + 1]) * a[i];
29         sum[i] = sum[i + 1] + a[i];
30     }
31     i128 ans = 0;
32     i64 g = 0;
33     auto gg = a;
34     for (int i = 0; i < n; i++) {
35         if (i > 0 && a[i] % g != 0) {
36             i64 last = 1;
37             for (int j = i; j < n; j++) {
38                 i64 ng = gcd(g, gg[j]);
39                 gg[j] = ng;
40                 if (a[j] - a[i] > ng || last >= ng) {
41                     continue;
42                 }
43                 last = ng;
44                 int L = lower_bound(a.begin(), a.end(), a[j]) - a.begin();
45                 int R = upper_bound(a.begin(), a.end(), a[j]) - a.begin();
46                 int v = suf[i] - (suf[L] - suf[R]);
47                 i128 res = ng + ssuf[i] - (ssuf[L] - ssuf[R]);
48                 if (v <= n - k - 1) {
49                     int t = n - k - 1 - v;
50                     int lo = i + 1, hi = n;
51                     while (lo < hi) {
52                         int x = (lo + hi) / 2;
53                         int c = n - x - suf[x];
54                         if (x <= L) {
55                             c -= R - L - (suf[L] - suf[R]);
56                         } else if (x < R) {
57                             c -= R - x - (suf[x] - suf[R]);
58                         }
59                         if (c <= t) {
60                             hi = x;
61                         } else {
62                             lo = x + 1;
63                         }
64                     }
65                     res += sum[lo] - ssuf[lo];
66                     if (lo <= L) {
67                         res -= sum[L] - sum[R] - (ssuf[L] - ssuf[R]);
68                     } else if (lo < R) {
69                         res -= sum[lo] - sum[R] - (ssuf[lo] - ssuf[R]);
70                     }
71                     ans = max(ans, res);
72                 }
73             }
74         }
75         g = gcd(g, a[i]);
76         int v = suf[i + 1];
77         if (v <= n - k - 1) {
78             i128 res = g + ssuf[i + 1];

```

```

79         int t = n - k - 1 - v;
80         int lo = i + 1, hi = n;
81         while (lo < hi) {
82             int x = (lo + hi) / 2;
83             int c = n - x - suf[x];
84             if (c <= t) {
85                 hi = x;
86             } else {
87                 lo = x + 1;
88             }
89         }
90         res += sum[lo] - ssuf[lo];
91         ans = max(ans, res);
92     }
93 }
94 cout << ans << "\n";
95 }
96 int main() {
97     ios::sync_with_stdio(false);
98     cin.tie(nullptr);
99     int t;
100    cin >> t;
101    while (t--) {
102        solve();
103    }
104    return 0;
105 }
```

**848: GCD Master (easy version)**

- Time limit: 3 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

This is the easy version of the problem. The only difference between the two versions is the constraint on  $m$ . You can make hacks only if both versions of the problem are solved.

You are given an array  $a$  of length  $n$  and two integers  $m$  and  $k$ . Each element in  $a$  satisfies  $1 \leq a_i \leq m$ .

In one operation, you choose two indices  $i$  and  $j$  such that  $1 \leq i < j \leq |a|$ , then append  $\gcd(a_i, a_j)$  to the back of the array and delete  $a_i$  and  $a_j$  from the array. Note that the length of the array decreases by one after this operation.

Find the maximum possible sum of the array after performing exactly  $k$  operations.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 using i128 = __int128;
```

```

5  ostream &operator<<(ostream &os, i128 n) {
6      string s;
7      while (n) {
8          s += '0' + n % 10;
9          n /= 10;
10     }
11     reverse(s.begin(), s.end());
12     return os << s;
13 }
14 void solve() {
15     int n, k;
16     i64 m;
17     cin >> n >> m >> k;
18     vector<i64> a(n);
19     for (int i = 0; i < n; i++) {
20         cin >> a[i];
21     }
22     sort(a.begin(), a.end());
23     vector<int> suf(n + 1);
24     vector<i128> ssuf(n + 1);
25     vector<i128> sum(n + 1);
26     for (int i = n - 1; i >= 0; i--) {
27         suf[i] = suf[i + 1] + (i == n - 1 || a[i] != a[i + 1]);
28         ssuf[i] = ssuf[i + 1] + (i == n - 1 || a[i] != a[i + 1]) * a[i];
29         sum[i] = sum[i + 1] + a[i];
30     }
31     i128 ans = 0;
32     i64 g = 0;
33     for (int i = 0; i < n; i++) {
34         if (i > 0 && a[i] % g != 0) {
35             i64 last = 1;
36             for (int j = i; j < n; j++) {
37                 i64 ng = gcd(g, a[j]);
38                 if (a[j] - a[i] > ng || last >= ng) {
39                     continue;
40                 }
41                 last = ng;
42                 int L = lower_bound(a.begin(), a.end(), a[j]) - a.begin();
43                 int R = upper_bound(a.begin(), a.end(), a[j]) - a.begin();
44                 int v = suf[i] - (suf[L] - suf[R]);
45                 i128 res = ng + ssuf[i] - (ssuf[L] - ssuf[R]);
46                 if (v <= n - k - 1) {
47                     int t = n - k - 1 - v;
48                     int lo = i + 1, hi = n;
49                     while (lo < hi) {
50                         int x = (lo + hi) / 2;
51                         int c = n - x - suf[x];
52                         if (x <= L) {
53                             c -= R - L - (suf[L] - suf[R]);
54                         } else if (x < R) {
55                             c -= R - x - (suf[x] - suf[R]);
56                         }
57                         if (c <= t) {
58                             hi = x;
59                         } else {
60                             lo = x + 1;
61                         }
62                     }
63                     res += sum[lo] - ssuf[lo];
64                     if (lo <= L) {
65                         res -= sum[L] - sum[R] - (ssuf[L] - ssuf[R]);
66                     } else if (lo < R) {
67                         res -= sum[lo] - sum[R] - (ssuf[lo] - ssuf[R]);
68                     }
}

```

```

69             ans = max(ans, res);
70         }
71     }
72 }
73 g = gcd(g, a[i]);
74 int v = suf[i + 1];
75 if (v <= n - k - 1) {
76     i128 res = g + ssuf[i + 1];
77     int t = n - k - 1 - v;
78     int lo = i + 1, hi = n;
79     while (lo < hi) {
80         int x = (lo + hi) / 2;
81         int c = n - x - suf[x];
82         if (c <= t) {
83             hi = x;
84         } else {
85             lo = x + 1;
86         }
87     }
88     res += sum[lo] - ssuf[lo];
89     ans = max(ans, res);
90 }
91 cout << ans << "\n";
92 }
93 int main() {
94     ios::sync_with_stdio(false);
95     cin.tie(nullptr);
96     int t;
97     cin >> t;
98     while (t--) {
99         solve();
100    }
101 }
102 return 0;
103 }
```

## 849: Teamwork

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

As soon as SWERC starts, your experienced 3-person team immediately realizes that the contest features  $a$  easy problems,  $b$  medium problems, and  $c$  hard problems. Solving a problem will take any of you 2, 3, or 4 time units, depending on whether the problem is easy, medium, or hard. Regardless of the difficulty of the problem, the last time unit spent to solve it has to be spent using your shared computer.

You organize your efforts so that each of you starts (and ends) solving problems at integer time units. Any given problem can be solved by only one contestant; it requires a contiguous amount of time (which depends on the difficulty of the problem). None of the 3 of you can solve more than one problem at a time, but you can start solving a new problem immediately after finishing one. Similarly, the shared

computer cannot be used by more than one of you at a time, but any of you can start using the computer (to complete the problem currently being solved) immediately after someone else stops using it.

Given that the contest lasts  $l$  time units, find the maximum number of problems that your team can solve. Additionally, find one way to solve the maximum number of problems.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int a, b, c, l;
8     cin >> a >> b >> c >> l;
9     int lo = 0, hi = a + b + c;
10    vector<array<int, 3>> ans;
11    auto check = [&](int x) {
12        int cnt[5] {};
13        cnt[2] = a;
14        cnt[3] = b;
15        cnt[4] = c;
16        int tot = x;
17        for (int i = 2; i <= 4; i++) {
18            int v = min(cnt[i], tot);
19            tot -= v;
20            cnt[i] = v;
21        }
22        ans.clear();
23        vector<int> f(l);
24        tot = 0;
25        for (int q = 2; q <= l; q++) {
26            int maxlen = 2;
27            while (maxlen < min(q, 4) && f[q - maxlen - 1] < 3) {
28                maxlen++;
29            }
30            vector<int> cand{4, 2, 3};
31            if (q == 3) {
32                cand = {3, 2};
33            }
34            for (auto i : cand) {
35                if (cnt[i] > 0 && i <= maxlen) {
36                    tot++;
37                    cnt[i]--;
38                    for (int x = q - i; x < q; x++) {
39                        f[x]++;
40                    }
41                    ans.push_back({0, q - i, q});
42                    break;
43                }
44            }
45        }
46        int time[3] = {l, l, l};
47        for (int i = int(ans.size()) - 1; i >= 0; i--) {
48            int t = -1;
49            for (int j = 0; j < 3; j++) {
50                if (time[j] >= ans[i][2]) {
51                    t = j;
52                    break;
```

```

53             }
54         }
55         time[t] = ans[i][1];
56         ans[i][0] = t + 1;
57     }
58     return tot == x;
59 };
60     while (lo < hi) {
61         int x = (lo + hi + 1) / 2;
62         if (check(x)) {
63             lo = x;
64         } else {
65             hi = x - 1;
66         }
67     }
68     cout << lo << "\n";
69     check(lo);
70     for (auto [x, p, q] : ans) {
71         cout << x << " " << p << " " << q << "\n";
72     }
73     return 0;
74 }
```

## 850: Wonderful Jump

- Time limit: 4 seconds
- Memory limit: 128 megabytes
- Input file: standard input
- Output file: standard output

You are given an array of positive integers  $a_1, a_2, \dots, a_n$  of length  $n$ .

In one operation you can jump from index  $i$  to index  $j$  ( $1 \leq i \leq j \leq n$ ) by paying  $\min(a_i, a_{i+1}, \dots, a_j) \cdot (j - i)^2$  eris.

For all  $k$  from 1 to  $n$ , find the minimum number of eris needed to get from index 1 to index  $k$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 1E18;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n;
9     cin >> n;
10    vector<int> a(n);
11    for (int i = 0; i < n; i++) {
12        cin >> a[i];
13    }
14    vector<int> last(500, -1), next(500, -1);
15    vector<int> rep(n, -1);
```

```

16     for (int i = n - 1; i >= 0; i--) {
17         if (a[i] < 500) {
18             rep[i] = next[a[i]];
19             next[a[i]] = i;
20         }
21     }
22     vector<i64> dp(n, inf);
23     dp[0] = 0;
24     for (int i = 0; i < n; i++) {
25         for (int j = max(0, i - 500); j < i; j++) {
26             dp[i] = min(dp[i], min(a[i], a[j]) * 1LL * (i - j) * (i - j) + dp[j]);
27         }
28         for (int j = 1; j < 500; j++) {
29             if (last[j] != -1) {
30                 dp[i] = min(dp[i], dp[last[j]] + 1LL * j * (i - last[j]) * (i - last[j]));
31             }
32         }
33         if (a[i] < 500) {
34             last[a[i]] = i;
35             next[a[i]] = rep[i];
36         }
37         for (int j = 1; j < 500; j++) {
38             if (next[j] != -1) {
39                 dp[next[j]] = min(dp[next[j]], dp[i] + 1LL * j * (i - next[j]) * (i - next[j]));
40             }
41         }
42     }
43     for (int i = 0; i < n; i++) {
44         cout << dp[i] << " \n"[i == n - 1];
45     }
46     return 0;
47 }
```

## math

### 851: Matrix Rank (Hard Version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. The only differences between the two versions of this problem are the constraints on  $k$ . You can make hacks only if all versions of the problem are solved.

You are given integers  $n, p$  and  $k$ .  $p$  is guaranteed to be a prime number.

For each  $r$  from 0 to  $k$ , find the number of  $n \times n$  matrices  $A$  of the field<sup>†</sup> of integers modulo  $p$  such that the rank<sup>‡</sup> of  $A$  is exactly  $r$ . Since these values are big, you are only required to output them modulo 998 244 353.

<sup>†</sup> [https://en.wikipedia.org/wiki/Field\\_\(mathematics\)](https://en.wikipedia.org/wiki/Field_(mathematics))

<sup>‡</sup> [https://en.wikipedia.org/wiki/Rank\\_\(linear\\_algebra\)](https://en.wikipedia.org/wiki/Rank_(linear_algebra))

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     i64 n;
38     int p;
39     int k;
40     cin >> n >> p >> k;
41     int r = P - 1;
42     for (int i = 2; i * i <= r; i++) {
43         while (r % i == 0 && power(Z(p), r / i).val() == 1) {
44             r /= i;
45         }
46         while (r > i && r % i == 0 && power(Z(p), i).val() == 1) {
47             r = i;
48         }
49     }
50     Z inv = Z(1 - p).inv();
51     Z ans = 1;
52     int v = 0;
53     auto mul = [&](i64 n, int t) {
54         if (n % r == 0) {
55             n /= r;

```

```

56         while (n % P == 0) {
57             n /= P;
58             v += t;
59         }
60         if (t == 1) {
61             ans *= n;
62         } else {
63             ans /= n;
64         }
65     } else {
66         Z val = (1 - power(Z(p), n)) * inv;
67         if (t == 1) {
68             ans *= val;
69         } else {
70             ans /= val;
71         }
72     }
73 };
74 for (int i = 0; i <= k; i++) {
75     if (i > 0 && i <= n) {
76         mul(n - i + 1, 1);
77         mul(i, -1);
78         ans *= power(Z(p), n) - power(Z(p), i - 1);
79     }
80     Z res = ans;
81     if (i > n || v > 0 || n / r != i / r + (n - i) / r) {
82         res = 0;
83     }
84     cout << res << " \n"[i == k];
85 }
86 return 0;
87 }
```

## 852: Matrix Rank (Easy Version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the easy version of the problem. The only differences between the two versions of this problem are the constraints on  $k$ . You can make hacks only if all versions of the problem are solved.

You are given integers  $n, p$  and  $k$ .  $p$  is guaranteed to be a prime number.

For each  $r$  from 0 to  $k$ , find the number of  $n \times n$  matrices  $A$  of the field<sup>†</sup> of integers modulo  $p$  such that the rank<sup>‡</sup> of  $A$  is exactly  $r$ . Since these values are big, you are only required to output them modulo 998 244 353.

<sup>†</sup> [https://en.wikipedia.org/wiki/Field\\_\(mathematics\)](https://en.wikipedia.org/wiki/Field_(mathematics))

<sup>‡</sup> [https://en.wikipedia.org/wiki/Rank\\_\(linear\\_algebra\)](https://en.wikipedia.org/wiki/Rank_(linear_algebra))

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     i64 n;
38     int p;
39     int k;
40     cin >> n >> p >> k;
41     int r = P - 1;
42     for (int i = 2; i * i <= r; i++) {
43         while (r % i == 0 && power(Z(p), r / i).val() == 1) {
44             r /= i;
45         }
46         while (r > i && r % i == 0 && power(Z(p), i).val() == 1) {
47             r = i;
48         }
49     }
50     Z inv = Z(1 - p).inv();
51     Z ans = 1;
52     int v = 0;
53     auto mul = [&](i64 n, int t) {
54         if (n % r == 0) {
55             n /= r;
56             while (n % P == 0) {
57                 n /= P;
58                 v += t;
59             }
60             if (t == 1) {

```

```

61         ans *= n;
62     } else {
63         ans /= n;
64     }
65 } else {
66     Z val = (1 - power(Z(p), n)) * inv;
67     if (t == 1) {
68         ans *= val;
69     } else {
70         ans /= val;
71     }
72 }
73 };
74 for (int i = 0; i <= k; i++) {
75     if (i > 0 && i <= n) {
76         mul(n - i + 1, 1);
77         mul(i, -1);
78         ans *= power(Z(p), n) - power(Z(p), i - 1);
79     }
80     Z res = ans;
81     if (i > n || v > 0 || n / r != i / r + (n - i) / r) {
82         res = 0;
83     }
84     cout << res << " \n"[i == k];
85 }
86 return 0;
87 }
```

**853: Split**

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Let's call an array  $b_1, b_2, \dots, b_m$  ( $m \geq 2$ ) good if it can be split into two parts such that all elements in the left part are strictly smaller than all elements in the right part. In other words, there must exist an index  $1 \leq i < m$  such that every element from  $b_1, \dots, b_i$  is strictly smaller than every element from  $b_{i+1}, \dots, b_m$ .

Given an array  $a_1, a_2, \dots, a_n$  consisting of distinct integers from 1 to  $n$ . There are  $q$  queries. Each query consists of two numbers  $l$  and  $r$ . For each query, determine whether the array  $a_l, a_{l+1}, \dots, a_r$  is good.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template <typename T>
5 # struct Fenwick;
```

```

6  int main() {
7      ios::sync_with_stdio(false);
8      cin.tie(nullptr);
9      int n;
10     cin >> n;
11     vector<int> a(n), inva(n);
12     for (int i = 0; i < n; i++) {
13         cin >> a[i];
14         a[i]--;
15         inva[a[i]] = i;
16     }
17     int q;
18     cin >> q;
19     vector<int> ans(q);
20     vector<vector<pair<int, int>>> qry(n);
21     for (int i = 0; i < q; i++) {
22         int l, r;
23         cin >> l >> r;
24         l--, r--;
25         qry[l].emplace_back(r, i);
26     }
27     vector<int> L(n), R(n), RR(n);
28     set<int> s{-1, n};
29     for (int i = n - 1; i >= 0; i--) {
30         int x = inva[i];
31         int l = *prev(s.lower_bound(x));
32         int r = *s.lower_bound(x);
33         s.insert(x);
34         L[x] = l, R[x] = r;
35     }
36     s = {-1, n};
37     for (int i = 0; i < n; i++) {
38         int x = inva[i];
39         if (R[x] < n) {
40             RR[x] = *s.lower_bound(R[x]);
41         }
42         s.insert(x);
43     }
44     vector<vector<pair<int, int>>> add(n);
45     for (int i = 0; i < n; i++) {
46         if (R[i] < n) {
47             add[L[i] + 1].emplace_back(R[i], 1);
48             add[L[i] + 1].emplace_back(RR[i], -1);
49             add[i + 1].emplace_back(R[i], -1);
50             add[i + 1].emplace_back(RR[i], 1);
51         }
52     }
53     Fenwick<int> fen(n);
54     for (int i = 0; i < n; i++) {
55         for (auto [x, y] : add[i]) {
56             fen.add(x, y);
57         }
58         for (auto [r, j] : qry[i]) {
59             if (fen.sum(r + 1) > 0) {
60                 ans[j] = 1;
61             }
62         }
63     }
64     for (int i = 0; i < q; i++) {
65         cout << (ans[i] ? "Yes" : "No") << "\n";
66     }
67     return 0;
68 }

```

## 854: Hard Design

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Consider an array of integers  $b_0, b_1, \dots, b_{n-1}$ . Your goal is to make all its elements equal. To do so, you can perform the following operation several (possibly, zero) times:

Pick a pair of indices  $0 \leq l \leq r \leq n - 1$ , then for each  $l \leq i \leq r$  increase  $b_i$  by 1 (i. e. replace  $b_i$  with  $b_i + 1$ ).

After performing this operation you receive  $(r - l + 1)^2$  coins.

The value  $f(b)$  is defined as a pair of integers  $(cnt, cost)$ , where  $cnt$  is the smallest number of operations required to make all elements of the array equal, and  $cost$  is the largest total number of coins you can receive among all possible ways to make all elements equal within  $cnt$  operations. In other words, first, you need to minimize the number of operations, second, you need to maximize the total number of coins you receive.

You are given an array of integers  $a_0, a_1, \dots, a_{n-1}$ . Please, find the value of  $f$  for all cyclic shifts of  $a$ .

Formally, for each  $0 \leq i \leq n - 1$  you need to do the following:

Let  $c_j = a_{(j+i) \pmod n}$  for each  $0 \leq j \leq n - 1$ .

Find  $f(c)$ . Since  $cost$  can be very large, output it modulo  $(10^9 + 7)$ .

Please note that under a fixed  $cnt$  you need to maximize the total number of coins  $cost$ , not its remainder modulo  $(10^9 + 7)$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {

```

```

15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 void solve() {
35     int n;
36     cin >> n;
37     vector<int> a(n);
38     for (int i = 0; i < n; i++) {
39         cin >> a[i];
40     }
41     int x = max_element(a.begin(), a.end()) - a.begin();
42     vector<Z> f(n);
43     vector<i64> g(n);
44     Z sum0 = 0;
45     Z sum1 = 0;
46     Z ans = 0;
47     i64 ansg = 0;
48     vector<pair<int, int>> stk;
49     for (int i = 1; i < n; i++) {
50         int d = a[(x + i - 1) % n] - a[(x + i) % n];
51         if (d > 0) {
52             stk.emplace_back(i, d);
53             sum0 += d;
54             ansg += d;
55         } else {
56             d = -d;
57             while (d > 0) {
58                 auto &[x, y] = stk.back();
59                 int t = min(y, d);
60                 d -= t;
61                 y -= t;
62                 sum0 -= t;
63                 sum1 -= (i - x) * Z(t);
64                 if (y == 0) {
65                     stk.pop_back();
66                 }
67             }
68         }
69         ans += 2 * sum1 + sum0;
70         sum1 += sum0;
71         f[(x + i + 1) % n] += ans;
72         g[(x + i + 1) % n] += ansg;
73     }
74     stk.clear();
75     sum0 = sum1 = ans = ansg = 0;
76     for (int i = 1; i < n; i++) {
77         int d = a[(x - i + 1 + n) % n] - a[(x - i + n) % n];
78         if (d > 0) {

```

```

79         stk.emplace_back(i, d);
80         sum0 += d;
81         ansg += d;
82     } else {
83         d = -d;
84         while (d > 0) {
85             auto &[x, y] = stk.back();
86             int t = min(y, d);
87             d -= t;
88             y -= t;
89             sum0 -= t;
90             sum1 -= (i - x) * Z(t);
91             if (y == 0) {
92                 stk.pop_back();
93             }
94         }
95     }
96     ans += 2 * sum1 + sum0;
97     sum1 += sum0;
98     f[(x - i + n) % n] += ans;
99     g[(x - i + n) % n] += ansg;
100 }
101 for (int i = 0; i < n; i++) {
102     cout << g[i] << " " << f[i] << "\n";
103 }
104 }
105 int main() {
106     ios::sync_with_stdio(false);
107     cin.tie(nullptr);
108     int t;
109     cin >> t;
110     while (t--) {
111         solve();
112     }
113     return 0;
114 }
```

### 855: Ksusha and Square

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Ksusha is a vigorous mathematician. She is keen on absolutely incredible mathematical riddles.

Today Ksusha came across a convex polygon of non-zero area. She is now wondering: if she chooses a pair of distinct points uniformly among all integer points (points with integer coordinates) inside or on the border of the polygon and then draws a square with two opposite vertices lying in the chosen points, what will the expectation of this square's area be?

A pair of distinct points is chosen uniformly among all pairs of distinct points, located inside or on the border of the polygon. Pairs of points  $p, q$  ( $p \neq q$ ) and  $q, p$  are considered the same.

Help Ksusha! Count the required expectation.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int N = 1 << 21;
5 i64 ceilDiv(i64 n, i64 m) {
6     if (n >= 0) {
7         return (n + m - 1) / m;
8     } else {
9         return n / m;
10    }
11 }
12 i64 floorDiv(i64 n, i64 m) {
13     if (n >= 0) {
14         return n / m;
15     } else {
16         return (n - m + 1) / m;
17     }
18 }
19 int main() {
20     ios::sync_with_stdio(false);
21     cin.tie(nullptr);
22     int n;
23     cin >> n;
24     vector<int> x(n), y(n);
25     for (int i = 0; i < n; i++) {
26         cin >> x[i] >> y[i];
27     }
28     i64 area = 0;
29     for (int i = 0; i < n; i++) {
30         int j = (i + 1) % n;
31         area += 1LL * x[i] * y[j] - 1LL * x[j] * y[i];
32     }
33     if (area < 0) {
34         reverse(x.begin(), x.end());
35         reverse(y.begin(), y.end());
36     }
37     double ans = 0;
38     double tot = 0;
39     auto work = [&]() {
40         vector<int> l(N), r(N, N);
41         int minx = N, maxx = 0;
42         for (int i = 0; i < n; i++) {
43             minx = min(minx, x[i] + N / 2);
44             maxx = max(maxx, x[i] + N / 2);
45             int j = (i + 1) % n;
46             if (x[i] < x[j]) {
47                 for (int X = x[i]; X <= x[j]; X++) {
48                     l[X + N / 2] = max<i64>(l[X + N / 2], y[i] + ceilDiv(1LL * (y[j] - y[i]) * (X - x[i]), x[j] - x[i]) + N / 2);
49                 }
50             } else if (x[i] > x[j]) {
51                 for (int X = x[j]; X <= x[i]; X++) {
52                     r[X + N / 2] = min<i64>(r[X + N / 2], y[j] + floorDiv(1LL * (y[i] - y[j]) * (X - x[j]), x[i] - x[j]) + N / 2);
53                 }
54             }
55         }
56         double s0 = 0, s1 = 0, s2 = 0;
57         for (int x = minx; x <= maxx; x++) {
58             int cnt = r[x] - l[x] + 1;

```

```

59         s2 += s1 * 2 + s0;
60         s1 += s0;
61         ans += cnt * s2;
62         s0 += cnt;
63     }
64     tot = s0;
65 };
66 work();
67 reverse(x.begin(), x.end());
68 reverse(y.begin(), y.end());
69 for (int i = 0; i < n; i++) {
70     swap(x[i], y[i]);
71 }
72 work();
73 ans /= tot;
74 ans /= tot - 1;
75 cout << fixed << setprecision(10) << ans << "\n";
76 return 0;
77 }
```

## 856: Lazy Numbers

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given positive integers  $n$  and  $k$ . For each number from 1 to  $n$ , we write its representation in the number system with base  $k$  (without leading zeros) and then sort the resulting array in lexicographic order as strings. In the sorted array, we number the elements from 1 to  $n$  (i.e., indexing starts from 1). Find the number of values  $i$  such that the representation of number  $i$  is at the  $i$ -th position in the sorted array of representations.

Examples of representations: 1 in any number system is equal to 1, 7 with  $k = 3$  is written as 21, and 81 with  $k = 9$  is written as 100.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void gen(i64 n, i64 k, vector<i64> &d) {
5     d.clear();
6     while (n) {
7         d.push_back(n % k);
8         n /= k;
9     }
10    reverse(d.begin(), d.end());
11 }
```

```

12 void solve() {
13     i64 n, k;
14     cin >> n >> k;
15     vector<i64> dn, dx;
16     gen(n, k, dn);
17     const int d = dn.size();
18     vector<i64> pw(d);
19     pw[0] = 1;
20     for (int i = 1; i < d; i++) {
21         pw[i] = pw[i - 1] * k;
22     }
23     auto spw = pw;
24     for (int i = 1; i < d; i++) {
25         spw[i] += spw[i - 1];
26     }
27     auto get = [&](i64 x) {
28         gen(x, k, dx);
29         i64 ans = dx.size();
30         for (int i = 0; i < dx.size(); i++) {
31             i64 v = dx[i] - (i == 0);
32             if (d - 2 - i >= 0) {
33                 ans += v * spw[d - 2 - i];
34             }
35         }
36         if (dx > dn) {
37             ans += n - pw[d - 1] + 1;
38         } else {
39             ans += x * pw[d - dx.size()] - pw[d - 1];
40         }
41         return ans;
42     };
43     i64 ans = 0;
44     for (int i = 0; i < d; i++) {
45         i64 l = pw[i], r = i < d - 1 ? pw[i + 1] - 1 : n;
46         i64 lo = l, hi = r + 1;
47         while (lo < hi) {
48             i64 m = (lo + hi) / 2;
49             if (get(m) >= m) {
50                 hi = m;
51             } else {
52                 lo = m + 1;
53             }
54         }
55         i64 L = lo;
56         lo = l, hi = r + 1;
57         while (lo < hi) {
58             i64 m = (lo + hi) / 2;
59             if (get(m) > m) {
60                 hi = m;
61             } else {
62                 lo = m + 1;
63             }
64         }
65         ans += lo - L;
66     }
67     cout << ans << "\n";
68 }
69 int main() {
70     ios::sync_with_stdio(false);
71     cin.tie(nullptr);
72     int t;
73     cin >> t;
74     while (t--) {
75         solve();

```

```

76     }
77     return 0;
78 }
```

**857: Swaps**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array of integers  $a_1, a_2, \dots, a_n$  ( $1 \leq a_i \leq n$ ). You can perform the following operation several (possibly, zero) times:

pick an arbitrary  $i$  and perform  $\text{swap}(a_i, a_{a_i})$ .

How many distinct arrays is it possible to attain? Output the answer modulo  $(10^9 + 7)$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 int main() {
```

```

35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int n;
38     cin >> n;
39     vector<int> a(n), deg(n);
40     for (int i = 0; i < n; i++) {
41         cin >> a[i];
42         a[i]--;
43         deg[a[i]]++;
44     }
45     Z ans = 1;
46     vector<int> vis(n, -1);
47     vector<bool> cyc(n);
48     for (int i = 0; i < n; i++) {
49         if (vis[i] != -1) {
50             continue;
51         }
52         int j;
53         for (j = i; vis[j] == -1; j = a[j]) {
54             vis[j] = i;
55         }
56         if (vis[j] != i) {
57             continue;
58         }
59         vector<int> v;
60         int k = j;
61         do {
62             v.push_back(k);
63             cyc[k] = true;
64             k = a[k];
65         } while (k != j);
66         Z res = 1;
67         for (auto x : v) {
68             res *= (deg[x] + 1);
69         }
70         for (auto x : v) {
71             res -= deg[x];
72         }
73         ans *= res;
74     }
75     for (int i = 0; i < n; i++) {
76         if (!cyc[i]) {
77             ans *= deg[i] + 1;
78         }
79     }
80     cout << ans << "\n";
81     return 0;
82 }
```

**858: Nephren Runs a Cinema**

- Time limit: 2.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Lakhesh loves to make movies, so Nephren helps her run a cinema. We may call it No. 68 Cinema.

However, one day, the No. 68 Cinema runs out of changes (they don't have 50-yuan notes currently), but Nephren still wants to start their business. (Assume that yuan is a kind of currency in Regulu Ere.)

There are three types of customers: some of them bring exactly a 50-yuan note; some of them bring a 100-yuan note and Nephren needs to give a 50-yuan note back to him/her; some of them bring VIP cards so that they don't need to pay for the ticket.

Now  $n$  customers are waiting outside in queue. Nephren wants to know how many possible queues are there that they are able to run smoothly (i.e. every customer can receive his/her change), and that the number of 50-yuan notes they have after selling tickets to all these customers is between  $l$  and  $r$ , inclusive. Two queues are considered different if there exists a customer whose type is different in two queues. As the number can be large, please output the answer modulo  $p$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 vector<pair<int, int>> factorize(int n) {
5     vector<pair<int, int>> factors;
6     for (int i = 2; static_cast<long long>(i) * i <= n; i++) {
7         if (n % i == 0) {
8             int t = 0;
9             for (; n % i == 0; n /= i)
10                 ++t;
11             factors.emplace_back(i, t);
12         }
13     }
14     if (n > 1)
15         factors.emplace_back(n, 1);
16     return factors;
17 }
18 constexpr int power(int base, i64 exp) {
19     int res = 1;
20     for (; exp > 0; base *= base, exp /= 2) {
21         if (exp % 2 == 1) {
22             res *= base;
23         }
24     }
25     return res;
26 }
27 constexpr int power(int base, i64 exp, int mod) {
28     int res = 1 % mod;
29     for (; exp > 0; base = 1LL * base * base % mod, exp /= 2) {
30         if (exp % 2 == 1) {
31             res = 1LL * res * base % mod;
32         }
33     }
34     return res;
35 }
36 int inverse(int a, int m) {
37     int g = m, r = a, x = 0, y = 1;
38     while (r != 0) {
39         int q = g / r;
40         g %= r;
41         swap(g, r);

```

```

42         x -= q * y;
43         swap(x, y);
44     }
45     return x < 0 ? x + m : x;
46 }
47 int solveModuloEquations(const vector<pair<int, int>> &e) {
48     int m = 1;
49     for (size_t i = 0; i < e.size(); i++) {
50         m *= e[i].first;
51     }
52     int res = 0;
53     for (size_t i = 0; i < e.size(); i++) {
54         int p = e[i].first;
55         res = (res + 1LL * e[i].second * (m / p) * inverse(m / p, p)) % m;
56     }
57     return res;
58 }
59 constexpr int N = 1E5;
60 class Binomial {
61     const int mod;
62 private:
63     const vector<pair<int, int>> factors;
64     vector<int> pk;
65     vector<vector<int>> prod;
66     static constexpr i64 exponent(i64 n, int p) {
67         i64 res = 0;
68         for (n /= p; n > 0; n /= p) {
69             res += n;
70         }
71         return res;
72     }
73     int product(i64 n, size_t i) {
74         int res = 1;
75         int p = factors[i].first;
76         for (; n > 0; n /= p) {
77             res = 1LL * res * power(prod[i].back(), n / pk[i], pk[i]) % pk[i] * prod[i][n %
78                 pk[i]] % pk[i];
79         }
80         return res;
81     }
82 public:
83     Binomial(int mod) : mod(mod), factors(factorize(mod)) {
84         pk.resize(factors.size());
85         prod.resize(factors.size());
86         for (size_t i = 0; i < factors.size(); i++) {
87             int p = factors[i].first;
88             int k = factors[i].second;
89             pk[i] = power(p, k);
90             prod[i].resize(min(N + 1, pk[i]));
91             prod[i][0] = 1;
92             for (int j = 1; j < prod[i].size(); j++) {
93                 if (j % p == 0) {
94                     prod[i][j] = prod[i][j - 1];
95                 } else {
96                     prod[i][j] = 1LL * prod[i][j - 1] * j % pk[i];
97                 }
98             }
99         }
100     }
101     int operator()(i64 n, i64 m) {
102         if (n < m || m < 0) {
103             return 0;
104         }
105         vector<pair<int, int>> ans(factors.size());

```

```

105     for (int i = 0; i < factors.size(); i++) {
106         int p = factors[i].first;
107         int k = factors[i].second;
108         int e = exponent(n, p) - exponent(m, p) - exponent(n - m, p);
109         if (e >= k) {
110             ans[i] = make_pair(pk[i], 0);
111         } else {
112             int pn = product(n, i);
113             int pm = product(m, i);
114             int pd = product(n - m, i);
115             int res = 1LL * pn * inverse(pm, pk[i]) % pk[i] * inverse(pd, pk[i]) % pk[i] * power(p, e) % pk[i];
116             ans[i] = make_pair(pk[i], res);
117         }
118     }
119     return solveModuloEquations(ans);
120 }
121 };
122 int main() {
123     ios::sync_with_stdio(false);
124     cin.tie(nullptr);
125     int n, p, l, r;
126     cin >> n >> p >> l >> r;
127     Binomial binom(p);
128     int ans = 0;
129     for (int i = 0; i <= n; i++) {
130         int r1 = min(r, i);
131         int l1 = l;
132         if (r1 % 2 != i % 2) {
133             r1--;
134         }
135         if (l1 % 2 != i % 2) {
136             l1++;
137         }
138         if (l1 <= r1) {
139             int res = binom(i, (i - l1) / 2) - binom(i, (i - r1) / 2 - 1);
140             if (res < 0) {
141                 res += p;
142             }
143             ans = (ans + 1LL * res * binom(n, i)) % p;
144         }
145     }
146     cout << ans << "\n";
147     return 0;
148 }
```

## 859: Lust

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A false witness that speaketh lies!

You are given a sequence containing  $n$  integers. There is a variable  $res$  that is equal to 0 initially. The following process repeats  $k$  times.

Choose an index from 1 to n uniformly at random. Name it x. Add to res the multiply of all  $a_i$ 's such that  $1 \leq i \leq n$ , but  $i \neq x$ . Then, subtract  $a_x$  by 1.

You have to find expected value of res at the end of the process. It can be proved that the expected value of res can be represented as an irreducible fraction . You have to find .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int n, k;
38     cin >> n >> k;
39     Z inv = Z(n).inv();
40     vector<int> a(n);
41     for (int i = 0; i < n; i++) {
42         cin >> a[i];
43     }
44     vector<Z> f(n + 1);
45     f[0] = 1;
46     for (int i = 0; i < n; i++) {
47         for (int j = i; j >= 0; j--) {
48             f[j + 1] -= f[j] * inv;
49             f[j] *= a[i];
50         }
51     }
52     Z ans = 1;

```

```

53     for (int i = 0; i < n; i++) {
54         ans *= a[i];
55     }
56     Z perm = 1;
57     for (int i = 0; i <= min(n, k); i++) {
58         ans -= f[i] * perm;
59         perm *= k - i;
60     }
61     cout << ans << "\n";
62     return 0;
63 }
```

## 860: Evaluate RBS

- Time limit: 10 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given  $2n$  tuples of values  $(a, b, c)$ , where  $a$  and  $b$  are positive integers and  $c$  is a bracket ‘(’ or ‘)’. Exactly  $n$  tuples have  $c = '('$  and the other  $n$  tuples have  $c = ')'$ .

You are asked to choose two positive values  $x$  and  $y$  ( $x > 0$  and  $y > 0$ ; not necessarily integers) and sort the tuples in the increasing value of  $a \cdot x + b \cdot y$ . If several tuples have the same value, you can place them in any order among themselves.

Is it possible to choose such  $x$  and  $y$  that taking brackets  $c$  from the tuples in the resulting order produces a regular bracket sequence?

A regular bracket sequence is a bracket sequence that can be transformed into a correct arithmetic expression by inserting characters “1” and “+” between the original characters of the sequence.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 # struct Point;
6 template<class T>
7 T dot(Point<T> a, Point<T> b) {
8     return a.x * b.x + a.y * b.y;
9 }
10 template<class T>
11 T cross(Point<T> a, Point<T> b) {
12     return a.x * b.y - a.y * b.x;
13 }
14 template<class T>
```

```

15 T square(Point<T> p) {
16     return dot(p, p);
17 }
18 template<class T>
19 double length(Point<T> p) {
20     return sqrt(double(square(p)));
21 }
22 long double length(Point<long double> p) {
23     return sqrt(square(p));
24 }
25 template<class T>
26 Point<T> normalize(Point<T> p) {
27     return p / length(p);
28 }
29 template<class T>
30 # struct Line;
31 template<class T>
32 Point<T> rotate(Point<T> a) {
33     return Point(-a.y, a.x);
34 }
35 template<class T>
36 int sgn(Point<T> a) {
37     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
38 }
39 template<class T>
40 bool pointOnLineLeft(Point<T> p, Line<T> l) {
41     return cross(l.b - l.a, p - l.a) > 0;
42 }
43 template<class T>
44 Point<T> lineIntersection(Line<T> l1, Line<T> l2) {
45     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1
        .a - l1.b));
46 }
47 template<class T>
48 bool pointOnSegment(Point<T> p, Line<T> l) {
49     return cross(p - l.a, l.b - l.a) == 0 && min(l.a.x, l.b.x) <= p.x && p.x <= max(l.a.x,
        l.b.x)
50         && min(l.a.y, l.b.y) <= p.y && p.y <= max(l.a.y, l.b.y);
51 }
52 template<class T>
53 bool pointInPolygon(Point<T> a, vector<Point<T>> p) {
54     int n = p.size();
55     for (int i = 0; i < n; i++) {
56         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
57             return true;
58         }
59     }
60     int t = 0;
61     for (int i = 0; i < n; i++) {
62         auto u = p[i];
63         auto v = p[(i + 1) % n];
64         if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
65             t ^= 1;
66         }
67         if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
68             t ^= 1;
69         }
70     }
71     return t == 1;
72 }
73 // 0 : not intersect
74 // 1 : strictly intersect
75 // 2 : overlap
76 // 3 : intersect at endpoint

```

```

77  template<class T>
78  tuple<int, Point<T>, Point<T>> segmentIntersection(Line<T> l1, Line<T> l2) {
79      if (max(l1.a.x, l1.b.x) < min(l2.a.x, l2.b.x)) {
80          return {0, Point<T>(), Point<T>()};
81      }
82      if (min(l1.a.x, l1.b.x) > max(l2.a.x, l2.b.x)) {
83          return {0, Point<T>(), Point<T>()};
84      }
85      if (max(l1.a.y, l1.b.y) < min(l2.a.y, l2.b.y)) {
86          return {0, Point<T>(), Point<T>()};
87      }
88      if (min(l1.a.y, l1.b.y) > max(l2.a.y, l2.b.y)) {
89          return {0, Point<T>(), Point<T>()};
90      }
91      if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
92          if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
93              return {0, Point<T>(), Point<T>()};
94          } else {
95              auto maxx1 = max(l1.a.x, l1.b.x);
96              auto minx1 = min(l1.a.x, l1.b.x);
97              auto maxy1 = max(l1.a.y, l1.b.y);
98              auto miny1 = min(l1.a.y, l1.b.y);
99              auto maxx2 = max(l2.a.x, l2.b.x);
100             auto minx2 = min(l2.a.x, l2.b.x);
101             auto maxy2 = max(l2.a.y, l2.b.y);
102             auto miny2 = min(l2.a.y, l2.b.y);
103             Point<T> p1(max(minx1, minx2), max(miny1, miny2));
104             Point<T> p2(min(maxx1, maxx2), min(maxy1, maxy2));
105             if (!pointOnSegment(p1, l1)) {
106                 swap(p1.y, p2.y);
107             }
108             if (p1 == p2) {
109                 return {3, p1, p2};
110             } else {
111                 return {2, p1, p2};
112             }
113         }
114     }
115     auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
116     auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
117     auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
118     auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
119     if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) || (cp3 < 0
120         && cp4 < 0)) {
121         return {0, Point<T>(), Point<T>()};
122     }
123     Point p = lineIntersection(l1, l2);
124     if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
125         return {1, p, p};
126     } else {
127         return {3, p, p};
128     }
129 template<class T>
130 bool segmentInPolygon(Line<T> l, vector<Point<T>> p) {
131     int n = p.size();
132     if (!pointInPolygon(l.a, p)) {
133         return false;
134     }
135     if (!pointInPolygon(l.b, p)) {
136         return false;
137     }
138     for (int i = 0; i < n; i++) {
139         auto u = p[i];

```

```
140         auto v = p[(i + 1) % n];
141         auto w = p[(i + 2) % n];
142         auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
143         if (t == 1) {
144             return false;
145         }
146         if (t == 0) {
147             continue;
148         }
149         if (t == 2) {
150             if (pointOnSegment(v, l) && v != l.a && v != l.b) {
151                 if (cross(v - u, w - v) > 0) {
152                     return false;
153                 }
154             }
155         } else {
156             if (p1 != u && p1 != v) {
157                 if (pointOnLineLeft(l.a, Line(v, u))
158                     || pointOnLineLeft(l.b, Line(v, u))) {
159                     return false;
160                 }
161             } else if (p1 == v) {
162                 if (l.a == v) {
163                     if (pointOnLineLeft(u, l)) {
164                         if (pointOnLineLeft(w, l)
165                             && pointOnLineLeft(w, Line(u, v))) {
166                             return false;
167                         }
168                     } else {
169                         if (pointOnLineLeft(w, l)
170                             || pointOnLineLeft(w, Line(u, v))) {
171                             return false;
172                         }
173                     }
174                 } else if (l.b == v) {
175                     if (pointOnLineLeft(u, Line(l.b, l.a))) {
176                         if (pointOnLineLeft(w, Line(l.b, l.a))
177                             && pointOnLineLeft(w, Line(u, v))) {
178                             return false;
179                         }
180                     } else {
181                         if (pointOnLineLeft(w, Line(l.b, l.a))
182                             || pointOnLineLeft(w, Line(u, v))) {
183                             return false;
184                         }
185                     }
186                 } else {
187                     if (pointOnLineLeft(u, l)) {
188                         if (pointOnLineLeft(w, Line(l.b, l.a))
189                             || pointOnLineLeft(w, Line(u, v))) {
190                             return false;
191                         }
192                     } else {
193                         if (pointOnLineLeft(w, l)
194                             || pointOnLineLeft(w, Line(u, v))) {
195                             return false;
196                         }
197                     }
198                 }
199             }
200         }
201     }
202     return true;
203 }
```

```

204 template<class T>
205   vector<Point<T>> hp(vector<Line<T>> lines) {
206     sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
207       auto d1 = l1.b - l1.a;
208       auto d2 = l2.b - l2.a;
209       if (sgn(d1) != sgn(d2)) {
210         return sgn(d1) == 1;
211       }
212       return cross(d1, d2) > 0;
213     });
214     deque<Line<T>> ls;
215     deque<Point<T>> ps;
216     for (auto l : lines) {
217       if (ls.empty()) {
218         ls.push_back(l);
219         continue;
220       }
221       while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
222         ps.pop_back();
223         ls.pop_back();
224       }
225       while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
226         ps.pop_front();
227         ls.pop_front();
228       }
229       if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
230         if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
231           if (!pointOnLineLeft(ls.back().a, l)) {
232             assert(ls.size() == 1);
233             ls[0] = l;
234           }
235           continue;
236         }
237         return {};
238       }
239       ps.push_back(lineIntersection(ls.back(), l));
240       ls.push_back(l);
241     }
242     while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
243       ps.pop_back();
244       ls.pop_back();
245     }
246     if (ls.size() <= 2) {
247       return {};
248     }
249     ps.push_back(lineIntersection(ls[0], ls.back()));
250     return vector(ps.begin(), ps.end());
251   }
252   using P = Point<i64>;
253   void solve() {
254     int n;
255     cin >> n;
256     n *= 2;
257     vector<P> a;
258     vector<int> b;
259     a.reserve(n);
260     b.reserve(n);
261     map<array<int, 2>, int> ind;
262     for (int i = 0; i < n; i++) {
263       int x, y;
264       cin >> x >> y;
265       char p;
266       cin >> p;
267       int val = p == '(' ? 1 : -1;

```

```

268     if (ind.count({x, y})) {
269         b[ind[{x, y}]] += val;
270     } else {
271         ind[{x, y}] = a.size();
272         a.push_back({x, y});
273         b.push_back(val);
274     }
275 }
276 n = a.size();
277 vector<int> p(n), invp(n);
278 vector<int> s(n + 1);
279 iota(p.begin(), p.end(), 0);
280 sort(p.begin(), p.end(),
281     [&](int i, int j) {
282         return a[i].x < a[j].x || (a[i].x == a[j].x && a[i].y < a[j].y);
283     });
284 int bad = 0;
285 for (int i = 0; i < n; i++) {
286     invp[p[i]] = i;
287     s[i + 1] = s[i] + b[p[i]];
288     bad += (s[i + 1] < 0);
289 }
290 if (bad == 0) {
291     cout << "YES\n";
292     return;
293 }
294 vector<tuple<P, int, int>> e;
295 e.reserve(n * (n - 1) / 2);
296 for (int i = 0; i < n; i++) {
297     for (int j = 0; j < n; j++) {
298         if (a[i].x < a[j].x && a[i].y > a[j].y) {
299             P v = rotate(a[j] - a[i]);
300             e.push_back({v, i, j});
301         }
302     }
303 }
304 sort(e.begin(), e.end(),
305     [&](auto a, auto b) {
306         return cross(get<0>(a), get<0>(b)) > 0;
307     });
308 int E = e.size();
309 for (int l = 0, r = 0; l < E; l = r) {
310     while (r < E && cross(get<0>(e[l]), get<0>(e[r])) == 0) {
311         r++;
312     }
313     vector<array<int, 2>> seg;
314     seg.reserve(r - l);
315     for (int i = l; i < r; i++) {
316         auto [_, x, y] = e[i];
317         seg.push_back({invp[x], invp[y]});
318     }
319     sort(seg.begin(), seg.end());
320     auto tmp = move(seg);
321     for (auto [l, r] : tmp) {
322         if (!seg.empty() && l <= seg.back()[1]) {
323             seg.back()[1] = max(seg.back()[1], r);
324         } else {
325             seg.push_back({l, r});
326         }
327     }
328     for (auto [l, r] : seg) {
329         for (int i = l + 1; i <= r; i++) {
330             bad -= (s[i] < 0);
331         }
332     }
333 }

```

```

332         }
333     if (bad == 0) {
334         cout << "YES\n";
335         return;
336     }
337     for (auto [l, r] : seg) {
338         reverse(p.begin() + l, p.begin() + r + 1);
339         for (int i = l; i <= r; i++) {
340             invp[p[i]] = i;
341             s[i + 1] = s[i] + b[p[i]];
342         }
343         for (int i = l + 1; i <= r; i++) {
344             bad += (s[i] < 0);
345         }
346     }
347     cout << "NO\n";
348 }
349 int main() {
350     ios::sync_with_stdio(false);
351     cin.tie(nullptr);
352     int t;
353     cin >> t;
354     while (t--) {
355         solve();
356     }
357     return 0;
358 }
```

**861: Mod Mod Mod**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a sequence of integers  $a_1, a_2, \dots, a_n$ . Let  $\oplus$ , and for  $1 \leq i < n$ . Here, denotes the modulus operation. Find the maximum value of  $f(x, 1)$  over all nonnegative integers  $x$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<i64> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    map<i64, i64> f;
14    f[a[0]] = 0;
```

```

15     for (int i = 0; i < n; i++) {
16         i64 res = -1;
17         for (auto it = f.upper_bound(a[i]); it != f.end(); it = f.erase(it)) {
18             res = max(res, it->second + (it->first / a[i] - 1) * a[i] * i);
19             i64 t = it->first % a[i];
20             if (t > 0) {
21                 f[t] = max(f[t], it->second + it->first / a[i] * a[i] * i);
22             }
23         }
24         if (res >= 0) {
25             f[a[i]] = max(f[a[i]], res);
26         }
27     }
28     i64 ans = 0;
29     for (auto [x, y] : f) {
30         ans = max(ans, (x - 1) * n + y);
31     }
32     cout << ans << "\n";
33     return 0;
34 }
```

## 862: PermuTree (hard version)

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. The differences between the two versions are the constraint on  $n$  and the time limit. You can make hacks only if both versions of the problem are solved.

You are given a tree with  $n$  vertices rooted at vertex 1.

For some permutation<sup>†</sup>  $a$  of length  $n$ , let  $f(a)$  be the number of pairs of vertices  $(u, v)$  such that  $a_u < a_{\text{lca}(u,v)} < a_v$ . Here,  $\text{lca}(u, v)$  denotes the lowest common ancestor of vertices  $u$  and  $v$ .

Find the maximum possible value of  $f(a)$  over all permutations  $a$  of length  $n$ .

<sup>†</sup> A permutation of length  $n$  is an array consisting of  $n$  distinct integers from 1 to  $n$  in arbitrary order.

For example,  $[2, 3, 1, 5, 4]$  is a permutation, but  $[1, 2, 2]$  is not a permutation (2 appears twice in the array), and  $[1, 3, 4]$  is also not a permutation ( $n = 3$  but there is 4 in the array).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
```

```

8     cin >> n;
9     vector<int> p(n, -1), siz(n, 1);
10    for (int i = 1; i < n; i++) {
11        cin >> p[i];
12        p[i]--;
13    }
14    for (int i = n - 1; i; i--) {
15        siz[p[i]] += siz[i];
16    }
17    i64 ans = 0;
18    vector<vector<int>> a(n);
19    for (int i = 1; i < n; i++) {
20        a[p[i]].push_back(siz[i]);
21    }
22    for (int i = 0; i < n; i++) {
23        if (a[i].empty()) {
24            continue;
25        }
26        auto &v = a[i];
27        sort(v.begin(), v.end(), greater());
28        int sum = accumulate(v.begin() + 1, v.end(), 0);
29        vector<int> dp(sum + 1);
30        dp[0] = 1;
31        for (int i = 1, j = i; i < v.size(); i = j) {
32            while (j < v.size() && v[i] == v[j]) {
33                j++;
34            }
35            int cnt = j - i;
36            int k = 1;
37            while (k < cnt) {
38                int x = v[i] * k;
39                for (int j = sum; j >= x; j--) {
40                    dp[j] |= dp[j - x];
41                }
42                cnt -= k;
43                k *= 2;
44            }
45            int x = v[i] * cnt;
46            for (int j = sum; j >= x; j--) {
47                dp[j] |= dp[j - x];
48            }
49        }
50        i64 res = 0;
51        for (int j = 0; j <= sum; j++) {
52            if (dp[j]) {
53                res = max(res, 1LL * j * (siz[i] - 1 - j));
54                res = max(res, 1LL * (j + v[0]) * (siz[i] - 1 - (j + v[0])));
55            }
56        }
57        ans += res;
58    }
59    cout << ans << "\n";
60    return 0;
61 }

```

**863: Game Bundles**

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input

- Output file: standard output

Rishi is developing games in the 2D metaverse and wants to offer game bundles to his customers. Each game has an associated enjoyment value. A game bundle consists of a subset of games whose total enjoyment value adds up to 60.

Your task is to choose  $k$  games, where  $1 \leq k \leq 60$ , along with their respective enjoyment values  $a_1, a_2, \dots, a_k$ , in such a way that exactly  $m$  distinct game bundles can be formed.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     i64 m;
8     cin >> m;
9     if (m > 20000000) {
10         vector<i64> a;
11         for (int i = 0; i < 10; i++) {
12             a.push_back(1);
13         }
14         for (int i = 0; i < 16; i++) {
15             a.push_back(2);
16         }
17         for (int i = 0; i < 10; i++) {
18             a.push_back(3);
19         }
20         vector<i64> f(61);
21         f[0] = 1;
22         for (int i = 0; i < a.size(); i++) {
23             for (int j = 60; j >= a[i]; j--) {
24                 f[j] += f[j - a[i]];
25             }
26         }
27         m -= f[60];
28         int N = 20000000;
29         vector<int> g(N + 1, 0);
30         g[0] = 0;
31         for (int i = 0; i < 20; i++) {
32             if (f[i] <= N) {
33                 for (int j = f[i]; j <= N; j++) {
34                     g[j] = min(g[j], g[j - f[i]] + 1);
35                 }
36             }
37         }
38         // for (int i = 0; i <= 60; i++) {
39         //     cerr << i << " " << f[i] << "\n";
40         // }
41         // cerr << "-----\n";
42         // int mx = 0;
43         // for (int i = N; i; i--) {
44         //     if (g[i] > mx) {
45         //         mx = g[i];
46         //         cout << i << " " << g[i] << "\n";
47         //     }
48     }
49 }
```

```

49         // mx = 0;
50         // for (int s = 0; s < (1 << 21); s++) {
51         //     i64 lo = 0, hi = 1E10;
52         //     for (int i = 0; i <= 20; i++) {
53         //         if (s >> i & 1) {
54         //             lo += f[i + 16];
55         //             lo = max(lo, 1LL * N);
56         //             hi += f[i + 16];
57         //         } else {
58         //             hi = min(hi, f[i + 16] - 1);
59         //         }
60         //     }
61         //     hi = min(hi, i64(1E10));
62         //     if (lo <= hi) {
63         //         mx = max(mx, __builtin_popcount(s));
64         //     }
65     }
66     // cerr << mx << "\n";
67     for (int i = 36; i >= 0; i--) {
68         if (m > N && m - f[i] > 5E6) {
69             m -= f[i];
70             a.push_back(60 - i);
71             for (int j = 60; j >= a.back(); j--) {
72                 f[j] += f[j - a.back()];
73             }
74         }
75     }
76     for (int i = 19; i >= 0; i--) {
77         while (m >= f[i] && g[m - f[i]] == g[m] - 1) {
78             a.push_back(60 - i);
79             m -= f[i];
80         }
81     }
82     cout << a.size() << "\n";
83     for (int i = 0; i < a.size(); i++) {
84         cout << a[i] << " \n"[i == a.size() - 1];
85     }
86     f.assign(61, 0);
87     f[0] = 1;
88     for (int i = 0; i < a.size(); i++) {
89         for (int j = 60; j >= a[i]; j--) {
90             f[j] += f[j - a[i]];
91         }
92     }
93     cerr << f[60] << "\n";
94 } else {
95     vector<i64> a;
96     for (int i = 0; i < 10; i++) {
97         a.push_back(1);
98     }
99     for (int i = 0; i < 16; i++) {
100        a.push_back(2);
101    }
102    vector<i64> f(61);
103    f[0] = 1;
104    for (int i = 0; i < a.size(); i++) {
105        for (int j = 60; j >= a[i]; j--) {
106            f[j] += f[j - a[i]];
107        }
108    }
109    int N = 20000000;
110    vector<int> g(N + 1, N);
111    g[0] = 0;
112    for (int i = 0; i < 20; i++) {

```

```

113         if (f[i] <= N) {
114             for (int j = f[i]; j <= N; j++) {
115                 g[j] = min(g[j], g[j - f[i]] + 1);
116             }
117         }
118     }
119     for (int i = 19; i >= 0; i--) {
120         while (m >= f[i] && g[m - f[i]] == g[m] - 1) {
121             a.push_back(60 - i);
122             m -= f[i];
123         }
124     }
125     cout << a.size() << "\n";
126     for (int i = 0; i < a.size(); i++) {
127         cout << a[i] << " \n"[i == a.size() - 1];
128     }
129     f.assign(61, 0);
130     f[0] = 1;
131     for (int i = 0; i < a.size(); i++) {
132         for (int j = 60; j >= a[i]; j--) {
133             f[j] += f[j - a[i]];
134         }
135     }
136     cerr << f[60] << "\n";
137 }
138 return 0;
139 }
```

## 864: Triangle Platinum?

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an interactive problem.

Made in Heaven is a rather curious Stand. Of course, it is (arguably) the strongest Stand in existence, but it is also an ardent puzzle enjoyer. For example, it gave Qtaro the following problem recently:

Made in Heaven has  $n$  hidden integers  $a_1, a_2, \dots, a_n$  ( $3 \leq n \leq 5000, 1 \leq a_i \leq 4$ ). Qtaro must determine all the  $a_i$  by asking Made in Heaven some queries of the following form:

In one query Qtaro is allowed to give Made in Heaven three distinct indexes  $i, j$  and  $k$  ( $1 \leq i, j, k \leq n$ ).

If  $a_i, a_j, a_k$  form the sides of a non-degenerate triangle<sup>†</sup>, Made in Heaven will respond with the area of this triangle.

Otherwise, Made in Heaven will respond with 0.

By asking at most 5500 such questions, Qtaro must either tell Made in Heaven all the values of the  $a_i$ , or report that it is not possible to uniquely determine them.

Unfortunately due to the universe reboot, Qtaro is not as smart as Jotaro. Please help Qtaro solve Made In Heaven's problem.

<sup>†</sup> Three positive integers  $a, b, c$  are said to form the sides of a non-degenerate triangle if and only if all of the following three inequalities hold:

$$a + b > c,$$

$$b + c > a,$$

$$c + a > b.$$

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int area(int a, int b, int c) {
5     if (a + b <= c) {
6         return 0;
7     }
8     if (b + c <= a) {
9         return 0;
10    }
11    if (c + a <= b) {
12        return 0;
13    }
14    return (a + b + c) * (a + b - c) * (b + c - a) * (c + a - b);
15 }
16 int query(int i, int j, int k) {
17     array<int, 3> a{i, j, k};
18     sort(a.begin(), a.end());
19     cout << "? " << a[0] + 1 << " " << a[1] + 1 << " " << a[2] + 1 << endl;
20     int ans;
21     cin >> ans;
22     return ans;
23 }
24 int invarea(int a, int b, int c, int x) {
25     for (int i = 1; i <= 4; i++) {
26         if (area(a > 0 ? a : i, b > 0 ? b : i, c > 0 ? c : i) == x) {
27             return i;
28         }
29     }
30     return 0;
31 }
32 int main() {
33     ios::sync_with_stdio(false);
34     cin.tie(nullptr);
35     int n;
36     cin >> n;
37     if (n < 9) {
38         vector<int> q;
39         for (int i = 0; i < n; i++) {
40             for (int j = i + 1; j < n; j++) {
41                 for (int k = j + 1; k < n; k++) {
42                     q.push_back(query(i, j, k));
43                 }
44             }
45         }
46     }
47 }
```

```

44         }
45     }
46     int res = -1;
47     for (int s = 0; s < (1 << (2 * n)); s++) {
48         vector<int> a;
49         for (int i = 0; i < n; i++) {
50             for (int j = i + 1; j < n; j++) {
51                 for (int k = j + 1; k < n; k++) {
52                     int x = (s >> (2 * i) & 3) + 1;
53                     int y = (s >> (2 * j) & 3) + 1;
54                     int z = (s >> (2 * k) & 3) + 1;
55                     a.push_back(area(x, y, z));
56                 }
57             }
58         }
59         if (a == q) {
60             if (res != -1) {
61                 cout << "!" << -1 << endl;
62                 return 0;
63             }
64             res = s;
65         }
66     }
67     cout << "!";
68     for (int i = 0; i < n; i++) {
69         int a = (res >> (2 * i) & 3) + 1;
70         cout << " " << a;
71     }
72     cout << endl;
73     return 0;
74 }
75 int i1, j1, k1;
76 int C;
77 [&]() {
78     for (int i = 0; i < 9; i++) {
79         for (int j = i + 1; j < 9; j++) {
80             for (int k = j + 1; k < 9; k++) {
81                 int res = query(i, j, k);
82                 int a = invarea(0, 0, 0, res);
83                 if (a != 0) {
84                     C = a;
85                     i1 = i, j1 = j, k1 = k;
86                     return;
87                 }
88             }
89         }
90     }
91 }();
92 vector<int> ans(n);
93 ans[i1] = ans[j1] = ans[k1] = C;
94 if (C >= 2) {
95     for (int i = 0; i < n; i++) {
96         if (ans[i] == 0) {
97             int res = query(i1, j1, i);
98             ans[i] = invarea(C, C, 0, res);
99         }
100    }
101    cout << "!";
102    for (int i = 0; i < n; i++) {
103        cout << " " << ans[i];
104    }
105    cout << endl;
106    return 0;
107 }

```

```

108     int i2, j2;
109     int D;
110     vector<int> t;
111     for (int i = 0; i < n && t.size() < 4; i++) {
112         if (ans[i] == 0) {
113             if (query(i1, j1, i) != 0) {
114                 ans[i] = 1;
115             } else {
116                 t.push_back(i);
117             }
118         }
119     }
120     if (t.empty()) {
121         cout << "!";
122         for (int i = 0; i < n; i++) {
123             cout << " " << ans[i];
124         }
125         cout << endl;
126         return 0;
127     }
128     [&]() {
129         for (int i = 0; i < t.size(); i++) {
130             for (int j = 0; j < i; j++) {
131                 int res = query(i1, t[i], t[j]);
132                 if (res == 0 || res == area(1, 1, 1)) {
133                     continue;
134                 }
135                 i2 = t[i], j2 = t[j];
136                 D = invarea(C, 0, 0, res);
137                 return;
138             }
139         }
140         cout << "!" << -1 << endl;
141         exit(0);
142     }();
143     ans[i2] = ans[j2] = D;
144     for (int i = 0; i < n; i++) {
145         if (ans[i] == 0) {
146             int res = query(i2, j2, i);
147             ans[i] = invarea(D, D, 0, res);
148         }
149     }
150     cout << "!";
151     for (int i = 0; i < n; i++) {
152         cout << " " << ans[i];
153     }
154     cout << endl;
155     return 0;
156 }

```

## 865: Tree Weights

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a tree with  $n$  nodes labelled  $1, 2, \dots, n$ . The  $i$ -th edge connects nodes  $u_i$  and  $v_i$  and has

an unknown positive integer weight  $w_i$ . To help you figure out these weights, you are also given the distance  $d_i$  between the nodes  $i$  and  $i + 1$  for all  $1 \leq i \leq n - 1$  (the sum of the weights of the edges on the simple path between the nodes  $i$  and  $i + 1$  in the tree).

Find the weight of each edge. If there are multiple solutions, print any of them. If there are no weights  $w_i$  consistent with the information, print a single integer  $-1$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct HLD;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n;
9     cin >> n;
10    HLD t(n);
11    vector<int> u(n - 1), v(n - 1);
12    for (int i = 0; i < n - 1; i++) {
13        cin >> u[i] >> v[i];
14        u[i]--, v[i]--;
15        t.addEdge(u[i], v[i]);
16    }
17    t.work();
18    vector<i64> d(n - 1);
19    for (int i = 0; i < n - 1; i++) {
20        cin >> d[i];
21    }
22    for (int i = 0; i < n - 1; i++) {
23        if (t.parent[u[i]] == v[i]) {
24            swap(u[i], v[i]);
25        }
26    }
27    vector<i64> w(n - 1);
28    vector<int> lca(n - 1);
29    for (int i = 0; i < n - 1; i++) {
30        lca[i] = t.lca(i, i + 1);
31    }
32    for (int l = 0; l < 40; l++) {
33        vector<int> a(n), b(n);
34        for (int i = 0; i < n - 1; i++) {
35            a[i + 1] = a[i] ^ (d[i] & 1);
36        }
37        for (int i = 1; i < n; i++) {
38            b[i] = a[i] ^ a[t.parent[i]];
39        }
40        for (int i = 0; i < n - 1; i++) {
41            w[i] += (1LL * b[v[i]]) << l;
42        }
43        auto dfs = [&](auto self, int x) -> void {
44            for (auto y : t.adj[x]) {
45                b[y] += b[x];
46                self(self, y);
47            }
48        };
49        dfs(dfs, 0);
50        for (int i = 0; i < n - 1; i++) {

```

```

51         i64 val = b[i] + b[i + 1] - 2 * b[lca[i]];
52         d[i] -= val;
53         if (d[i] < 0 || d[i] % 2 != 0) {
54             cout << -1 << "\n";
55             return 0;
56         }
57         d[i] /= 2;
58     }
59 }
60 if (count(w.begin(), w.end(), 0) > 0) {
61     cout << -1 << "\n";
62     return 0;
63 }
64 for (int i = 0; i < n - 1; i++) {
65     cout << w[i] << "\n";
66 }
67 return 0;
68 }
```

**866: Min Cost Permutation (Hard Version)**

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The only difference between this problem and the easy version is the constraints on  $t$  and  $n$ .

You are given an array of  $n$  positive integers  $a_1, \dots, a_n$ , and a (possibly negative) integer  $c$ .

Across all permutations  $b_1, \dots, b_n$  of the array  $a_1, \dots, a_n$ , consider the minimum possible value of

$$\sum_{i=1}^{n-1} |b_{i+1} - b_i - c|.$$

Find the lexicographically smallest permutation  $b$  of the array  $a$  that achieves this minimum.

A sequence  $x$  is lexicographically smaller than a sequence  $y$  if and only if one of the following holds:

$x$  is a prefix of  $y$ , but  $x \neq y$ ;

in the first position where  $x$  and  $y$  differ, the sequence  $x$  has a smaller element than the corresponding element in  $y$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
```

```
5     int n, c;
6     cin >> n >> c;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    if (c >= 0) {
12        sort(a.begin(), a.end());
13        for (int i = 0; i < n; i++) {
14            cout << a[i] << " \n"[i == n - 1];
15        }
16        return;
17    }
18    sort(a.begin(), a.end(), greater());
19    c = -c;
20    vector<int> ans;
21    for (int i = 1, j = 0; i <= n; i++) {
22        if (i == n || a[i - 1] - a[i] > c) {
23            set<pair<int, int>> s, v;
24            for (int k = j; k < i; k++) {
25                s.emplace(a[k], k);
26                v.emplace(a[k], k);
27            }
28            auto x = *s.rbegin();
29            ans.push_back(x.first);
30            s.extract(x);
31            int lst = x.first;
32            v.erase(v.begin());
33            while (!s.empty()) {
34                bool ok = false;
35                while (true) {
36                    auto a = v.lower_bound(pair(lst - c, 0));
37                    if (a == v.end()) {
38                        break;
39                    }
40                    auto x = *a;
41                    auto it = s.find(x);
42                    v.erase(a);
43                    assert(it != s.begin());
44                    if (it == s.end()) {
45                        continue;
46                    }
47                    auto r = next(it);
48                    if (r == s.end()) {
49                        continue;
50                    }
51                    auto l = prev(it);
52                    if (r->first - l->first > c) {
53                        continue;
54                    }
55                    ok = true;
56                    ans.push_back(it->first);
57                    lst = it->first;
58                    s.erase(it);
59                    break;
60                }
61                if (!ok) {
62                    auto x = *s.rbegin();
63                    ans.push_back(x.first);
64                    lst = x.first;
65                    s.extract(x);
66                }
67            }
68            j = i;
```

```

69      }
70  }
71  for (int i = 0; i < n; i++) {
72    cout << ans[i] << " \n"[i == n - 1];
73  }
74 }
75 int main() {
76   ios::sync_with_stdio(false);
77   cin.tie(nullptr);
78   int t;
79   cin >> t;
80   while (t--) {
81     solve();
82   }
83   return 0;
84 }
```

## 867: Swimmers in the Pool

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

There is a pool of length  $l$  where  $n$  swimmers plan to swim. People start swimming at the same time (at the time moment 0), but you can assume that they take different lanes, so they don't interfere with each other.

Each person swims along the following route: they start at point 0 and swim to point  $l$  with constant speed (which is equal to  $v_i$  units per second for the  $i$ -th swimmer). After reaching the point  $l$ , the swimmer instantly (in negligible time) turns back and starts swimming to the point 0 with the same constant speed. After returning to the point 0, the swimmer starts swimming to the point  $l$ , and so on.

Let's say that some real moment of time is a meeting moment if there are at least two swimmers that are in the same point of the pool at that moment of time (that point may be 0 or  $l$  as well as any other real point inside the pool).

The pool will be open for  $t$  seconds. You have to calculate the number of meeting moments while the pool is open. Since the answer may be very large, print it modulo  $10^9 + 7$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
```

```

3  using i64 = long long;
4  template<class T>
5  constexpr T power(T a, i64 b) {
6      T res = 1;
7      for (; b; b /= 2, a *= a) {
8          if (b % 2) {
9              res *= a;
10         }
11     }
12     return res;
13 }
14 template<int P>
15 # struct MInt;
16 template<>
17 int MInt<0>::Mod = 1;
18 template<int V, int P>
19 constexpr MInt<P> CInv = MInt<P>(V).inv();
20 constexpr int P = 1000000007;
21 using Z = MInt<P>;
22 vector<int> rev;
23 template<int P>
24 vector<MInt<P>> roots{0, 1};
25 template<int P>
26 constexpr MInt<P> findPrimitiveRoot() {
27     MInt<P> i = 2;
28     int k = __builtin_ctz(P - 1);
29     while (true) {
30         if (power(i, (P - 1) / 2) != 1) {
31             break;
32         }
33         i += 1;
34     }
35     return power(i, (P - 1) >> k);
36 }
37 template<int P>
38 constexpr MInt<P> primitiveRoot = findPrimitiveRoot<P>();
39 template<>
40 constexpr MInt<998244353> primitiveRoot<998244353> {31};
41 template<int P>
42 constexpr void dft(vector<MInt<P>> &a) {
43     int n = a.size();
44     if (int(rev.size()) != n) {
45         int k = __builtin_ctz(n) - 1;
46         rev.resize(n);
47         for (int i = 0; i < n; i++) {
48             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
49         }
50     }
51     for (int i = 0; i < n; i++) {
52         if (rev[i] < i) {
53             swap(a[i], a[rev[i]]);
54         }
55     }
56     if (roots<P>.size() < n) {
57         int k = __builtin_ctz(roots<P>.size());
58         roots<P>.resize(n);
59         while ((1 << k) < n) {
60             auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k - 1));
61             for (int i = 1 << (k - 1); i < (1 << k); i++) {
62                 roots<P>[2 * i] = roots<P>[i];
63                 roots<P>[2 * i + 1] = roots<P>[i] * e;
64             }
65             k++;
66         }
67     }
68 }

```

```

67     }
68     for (int k = 1; k < n; k *= 2) {
69         for (int i = 0; i < n; i += 2 * k) {
70             for (int j = 0; j < k; j++) {
71                 MInt<P> u = a[i + j];
72                 MInt<P> v = a[i + j + k] * roots<P>[k + j];
73                 a[i + j] = u + v;
74                 a[i + j + k] = u - v;
75             }
76         }
77     }
78 }
79 template<int P>
80 constexpr void idft(vector<MInt<P>> &a) {
81     int n = a.size();
82     reverse(a.begin() + 1, a.end());
83     dft(a);
84     MInt<P> inv = (1 - P) / n;
85     for (int i = 0; i < n; i++) {
86         a[i] *= inv;
87     }
88 }
89 template<int P = 998244353>
90 # struct Poly;
91 template<int P = 998244353>
92 Poly<P> berlekampMassey(const Poly<P> &s) {
93     Poly<P> c;
94     Poly<P> oldC;
95     int f = -1;
96     for (int i = 0; i < s.size(); i++) {
97         auto delta = s[i];
98         for (int j = 1; j <= c.size(); j++) {
99             delta -= c[j - 1] * s[i - j];
100        }
101        if (delta == 0) {
102            continue;
103        }
104        if (f == -1) {
105            c.resize(i + 1);
106            f = i;
107        } else {
108            auto d = oldC;
109            d *= -1;
110            d.insert(d.begin(), 1);
111            MInt<P> df1 = 0;
112            for (int j = 1; j <= d.size(); j++) {
113                df1 += d[j - 1] * s[f + 1 - j];
114            }
115            assert(df1 != 0);
116            auto coef = delta / df1;
117            d *= coef;
118            Poly<P> zeros(i - f - 1);
119            zeros.insert(zeros.end(), d.begin(), d.end());
120            d = zeros;
121            auto temp = c;
122            c += d;
123            if (i - temp.size() > f - oldC.size()) {
124                oldC = temp;
125                f = i;
126            }
127        }
128    }
129    c *= -1;
130    c.insert(c.begin(), 1);

```

```
131     return c;
132 }
133 template<int P = 998244353>
134 MInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
135     int m = q.size() - 1;
136     while (n > 0) {
137         auto newq = q;
138         for (int i = 1; i <= m; i += 2) {
139             newq[i] *= -1;
140         }
141         auto newp = p * newq;
142         newq = q * newq;
143         for (int i = 0; i < m; i++) {
144             p[i] = newp[i * 2 + n % 2];
145         }
146         for (int i = 0; i <= m; i++) {
147             q[i] = newq[i * 2];
148         }
149         n /= 2;
150     }
151     return p[0] / q[0];
152 }
153 int main() {
154     ios::sync_with_stdio(false);
155     cin.tie(nullptr);
156     int l, t;
157     cin >> l >> t;
158     int n;
159     cin >> n;
160     vector<int> v(n);
161     for (int i = 0; i < n; i++) {
162         cin >> v[i];
163     }
164     int m = *max_element(v.begin(), v.end());
165     Poly<998244353> f(m + 1);
166     for (int i = 0; i < n; i++) {
167         f[v[i]] = 1;
168     }
169     vector<bool> g(2 * m + 1);
170     auto sum = f * f;
171     for (int i = 0; i < n; i++) {
172         sum[2 * v[i]] -= 1;
173     }
174     for (int i = 1; i <= 2 * m; i++) {
175         if (sum[i] != 0) {
176             g[i] = true;
177         }
178     }
179     auto dif = f.mult(f);
180     for (int i = 1; i <= m; i++) {
181         if (dif[i] != 0) {
182             g[i] = true;
183         }
184     }
185     m *= 2;
186     for (int i = 1; i <= m; i++) {
187         for (int j = i; j <= m; j += i) {
188             if (g[j]) {
189                 g[i] = true;
190             }
191         }
192     }
193     vector<Z> h(m + 1);
194     for (int i = 1; i <= m; i++) {
```

```

195     h[i] = 1LL * i * t / (2 * l);
196 }
197 for (int i = 1; i <= m; i++) {
198     for (int j = 2 * i; j <= m; j += i) {
199         h[j] -= h[i];
200     }
201 }
202 Z ans = 0;
203 for (int i = 1; i <= m; i++) {
204     if (g[i]) {
205         ans += h[i];
206     }
207 }
208 cout << ans << "\n";
209 return 0;
210 }
```

## 868: Tenzing and Random Operations

- Time limit: 2 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

Tenzing has an array  $a$  of length  $n$  and an integer  $v$ .

Tenzing will perform the following operation  $m$  times:

Choose an integer  $i$  such that  $1 \leq i \leq n$  uniformly at random.

For all  $j$  such that  $i \leq j \leq n$ , set  $a_j := a_j + v$ .

Tenzing wants to know the expected value of  $\prod_{i=1}^n a_i$  after performing the  $m$  operations, modulo  $10^9 + 7$ .

Formally, let  $M = 10^9 + 7$ . It can be shown that the answer can be expressed as an irreducible fraction  $\frac{p}{q}$ , where  $p$  and  $q$  are integers and  $q \not\equiv 0 \pmod{M}$ . Output the integer equal to  $p \cdot q^{-1} \pmod{M}$ . In other words, output the integer  $x$  that  $0 \leq x < M$  and  $x \cdot q \equiv p \pmod{M}$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
```

```

11     }
12     return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int n, m;
39     Z v;
40     cin >> n >> m >> v;
41     vector<Z> f{1};
42     for (int i = 1; i <= n; i++) {
43         Z a;
44         cin >> a;
45         auto inv = comb.inv(i);
46         Z p = 1;
47         for (int j = 0; j < i; j++) {
48             f[j] *= p;
49             p *= (i - 1) * inv;
50         }
51         vector<Z> g(i + 1);
52         for (int j = 0; j < i; j++) {
53             g[j] += f[j] * (a + j * v);
54             g[j + 1] += f[j] * (j + 1) * v;
55         }
56         swap(f, g);
57     }
58     Z p = 1;
59     Z ans = 0;
60     for (int i = 0; i <= n; i++) {
61         ans += p * f[i];
62         p *= (m - i) * comb.inv(i + 1);
63     }
64     cout << ans << "\n";
65     return 0;
66 }

```

**869: Tenzing and Random Real Numbers**

- Time limit: 1 second

- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  uniform random real variables between 0 and 1, inclusive, which are denoted as  $x_1, x_2, \dots, x_n$ .

Tenzing has  $m$  conditions. Each condition has the form of  $x_i + x_j \leq 1$  or  $x_i + x_j \geq 1$ .

Tenzing wants to know the probability that all the conditions are satisfied, modulo 998 244 353.

Formally, let  $M = 998\ 244\ 353$ . It can be shown that the answer can be expressed as an irreducible fraction  $\frac{p}{q}$ , where  $p$  and  $q$  are integers and  $q \not\equiv 0 \pmod{M}$ . Output the integer equal to  $p \cdot q^{-1} \pmod{M}$ . In other words, output the integer  $x$  that  $0 \leq x < M$  and  $x \cdot q \equiv p \pmod{M}$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int n, m;
38     cin >> n >> m;
39     vector a(2, vector<int>(n));
40     for (int i = 0; i < m; i++) {

```

```

41     int t, x, y;
42     cin >> t >> x >> y;
43     x--, y--;
44     a[t][x] |= 1 << y;
45     a[t][y] |= 1 << x;
46 }
47 vector<Z> dp(1 << n);
48 dp[0] = 1;
49 for (int s = 1; s < (1 << n); s++) {
50     for (int i = 0; i < n; i++) {
51         if (s >> i & 1) {
52             dp[s] += dp[s ^ 1 << i] * (2 - ((a[0][i] & s) != 0) - ((a[1][i] & s) != 0));
53         }
54     }
55 }
56 auto ans = dp[(1 << n) - 1];
57 ans /= power(Z(2), n);
58 for (int i = 1; i <= n; i++) {
59     ans /= i;
60 }
61 cout << ans << "\n";
62 return 0;
63 }

```

## 870: Doctor's Brown Hypothesis

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The rebels have been crushed in the most recent battle with the imperial forces, but there is a ray of new hope.

Meanwhile, on one of the conquered planets, Luke was getting ready for an illegal street race (which should come as no surprise, given his family history). Luke arrived at the finish line with 88 miles per hour on his speedometer. After getting out of the car, he was greeted by a new reality. It turns out that the battle has not happened yet and will start in exactly  $k$  hours.

The rebels have placed a single battleship on each of the  $n$  planets.  $m$  unidirectional wormholes connect the planets. Traversing each wormhole takes exactly one hour. Generals of the Imperium have planned the battle precisely, but their troops cannot dynamically adapt to changing circumstances. Because of this, it is enough for the rebels to move some ships around before the battle to confuse the enemy, secure victory and change the galaxy's fate.

Owing to numerous strategical considerations, which we now omit, the rebels would like to choose two ships that will switch places so that both of them will be on the move for the whole time (exactly  $k$  hours). In other words, rebels look for two planets,  $x$  and  $y$ , such that paths of length  $k$  exist from  $x$  to  $y$  and from  $y$  to  $x$ .

Because of the limited fuel supply, choosing one ship would also be acceptable. This ship should fly through the wormholes for  $k$  hours and then return to its initial planet.

How many ways are there to choose the ships for completing the mission?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct SCC;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, m;
9     i64 k;
10    cin >> n >> m >> k;
11    vector<vector<int>> adj(n);
12    SCC scc(n);
13    for (int i = 0; i < m; i++) {
14        int u, v;
15        cin >> u >> v;
16        u--, v--;
17        adj[u].push_back(v);
18        scc.addEdge(u, v);
19    }
20    auto bel = scc.work();
21    vector<int> vis(n, -1);
22    for (int i = 0; i < n; i++) {
23        if (vis[i] == -1) {
24            queue<int> q;
25            q.push(i);
26            vis[i] = 0;
27            while (!q.empty()) {
28                int x = q.front();
29                q.pop();
30                for (auto y : adj[x]) {
31                    if (vis[y] == -1 && bel[y] == bel[x]) {
32                        q.push(y);
33                        vis[y] = vis[x] + 1;
34                    }
35                }
36            }
37        }
38    }
39    i64 ans = 0;
40    int cnt = scc.cnt;
41    vector<vector<int>> v(cnt);
42    for (int i = 0; i < n; i++) {
43        v[bel[i]].push_back(i);
44    }
45    for (auto a : v) {
46        int g = 0;
47        for (auto x : a) {
48            for (auto y : adj[x]) {
49                if (bel[x] == bel[y]) {
50                    g = gcd(g, vis[x] + 1 - vis[y]);
51                }
52            }
53        }
54        if (g == 0) {

```

```

55         continue;
56     }
57     vector<int> cnt(g);
58     for (auto x : a) {
59         cnt[vis[x] % g]++;
60     }
61     if (k % g == 0) {
62         ans += a.size();
63         for (int i = 0; i < g; i++) {
64             ans += 1LL * cnt[i] * (cnt[i] - 1) / 2;
65         }
66     } else if ((2 * k) % g == 0) {
67         for (int i = 0; i < g / 2; i++) {
68             ans += 1LL * cnt[i] * cnt[i + g / 2];
69         }
70     }
71 }
72 cout << ans << "\n";
73 return 0;
74 }
```

## misc

### 871: Pumping Lemma

- Time limit: 2 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

You are given two strings  $s, t$  of length  $n, m$ , respectively. Both strings consist of lowercase letters of the English alphabet.

Count the triples  $(x, y, z)$  of strings such that the following conditions are true:

$s = x + y + z$  (the symbol  $+$  represents the concatenation);

$t = x + \underbrace{y + \cdots + y}_{k \text{ times}} + z$  for some integer  $k$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11 }
```

```
11     }
12     return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr i64 P = i64(1E18) + 9;
33 using Z = MLong<P>;
34 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     const Z B = rng() % (P - 100) + 50;
39     int n, m;
40     cin >> n >> m;
41     string s, t;
42     cin >> s >> t;
43     vector<Z> h(m + 1), pw(m + 1);
44     pw[0] = 1;
45     for (int i = 0; i < m; i++) {
46         pw[i + 1] = pw[i] * B;
47         h[i + 1] = h[i] * B + t[i];
48     }
49     int more = m - n;
50     int lcp = 0, lcs = 0;
51     while (lcp < n && s[lcp] == t[lcp]) {
52         lcp++;
53     }
54     while (lcs < n && s[n - 1 - lcs] == t[m - 1 - lcs]) {
55         lcs++;
56     }
57     i64 ans = 0;
58     auto gethash = [&](int l, int r) {
59         return h[r] - h[l] * pw[r - l];
60     };
61     auto get = [&](int max, auto p) {
62         int x = 0;
63         while (2 * x + 1 <= max && p(2 * x + 1)) {
64             x = 2 * x + 1;
65         }
66         int lo = x, hi = min(2 * x, max);
67         while (lo < hi) {
68             x = (lo + hi + 1) / 2;
69             if (p(x)) {
70                 lo = x;
71             } else {
72                 hi = x - 1;
73             }
74     }
```

```

75         return lo;
76     };
77     for (int len = 1; len <= more; len++) {
78         if (more % len != 0) {
79             continue;
80         }
81         int l = max(0, n - lcs - len), r = min(lcp, n - len);
82         if (l > r) {
83             continue;
84         }
85         r += more;
86         for (int i = l; i + more <= r; i++) {
87             int j = i + more, k = j;
88             j -= get(more, [&](int x) {
89                 return gethash(j - x, j) == gethash(j - x + len, j + len);
90             });
91             k += get(r - k, [&](int x) {
92                 return gethash(k, k + x) == gethash(k + len, k + x + len);
93             });
94             ans += max(0, k - j - more + 1);
95             i = k;
96         }
97     }
98     cout << ans << "\n";
99 }
100 }
```

## 872: Landscaping

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are appointed to a very important task: you are in charge of flattening one specific road.

The road can be represented as a polygonal line starting at  $(0, 0)$ , ending at  $(n - 1, 0)$  and consisting of  $n$  vertices (including starting and ending points). The coordinates of the  $i$ -th vertex of the polyline are  $(i, a_i)$ .

“Flattening” road is equivalent to choosing some line segment from  $(0, y_0)$  to  $(n - 1, y_1)$  such that all points of the polyline are below the chosen segment (or on the same height). Values  $y_0$  and  $y_1$  may be real.

You can imagine that the road has some dips and pits, and you start pouring pavement onto it until you make the road flat. Points  $0$  and  $n - 1$  have infinitely high walls, so pavement doesn’t fall out of segment  $[0, n - 1]$ .

The cost of flattening the road is equal to the area between the chosen segment and the polyline. You want to minimize the cost, that’s why the flattened road is not necessary horizontal.

But there is a problem: your data may be too old, so you sent a person to measure new heights. The person goes from 0 to  $n - 1$  and sends you new heights  $b_i$  of each vertex  $i$  of the polyline.

Since measuring new heights may take a while, and you don't know when you'll be asked, calculate the minimum cost (and corresponding  $y_0$  and  $y_1$ ) to flatten the road after each new height  $b_i$  you get.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 # struct Point;
6 template<class T>
7 # struct Line;
8 template<class T>
9 T dot(const Point<T> &a, const Point<T> &b) {
10     return a.x * b.x + a.y * b.y;
11 }
12 template<class T>
13 T cross(const Point<T> &a, const Point<T> &b) {
14     return a.x * b.y - a.y * b.x;
15 }
16 template<class T>
17 T square(const Point<T> &p) {
18     return dot(p, p);
19 }
20 template<class T>
21 double length(const Point<T> &p) {
22     return sqrt(square(p));
23 }
24 template<class T>
25 double length(const Line<T> &l) {
26     return length(l.a - l.b);
27 }
28 template<class T>
29 Point<T> normalize(const Point<T> &p) {
30     return p / length(p);
31 }
32 template<class T>
33 bool parallel(const Line<T> &l1, const Line<T> &l2) {
34     return cross(l1.b - l1.a, l2.b - l2.a) == 0;
35 }
36 template<class T>
37 double distance(const Point<T> &a, const Point<T> &b) {
38     return length(a - b);
39 }
40 template<class T>
41 double distancePL(const Point<T> &p, const Line<T> &l) {
42     return abs(cross(l.a - l.b, l.a - p)) / length(l);
43 }
44 template<class T>
45 double distancePS(const Point<T> &p, const Line<T> &l) {
46     if (dot(p - l.a, l.b - l.a) < 0) {
47         return distance(p, l.a);
48     }
49     if (dot(p - l.b, l.a - l.b) < 0) {
50         return distance(p, l.b);
51 }
52 }
```

```

51     }
52     return distancePL(p, l);
53 }
54 template<class T>
55 Point<T> rotate(const Point<T> &a) {
56     return Point(-a.y, a.x);
57 }
58 template<class T>
59 int sgn(const Point<T> &a) {
60     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
61 }
62 template<class T>
63 bool pointOnLineLeft(const Point<T> &p, const Line<T> &l) {
64     return cross(l.b - l.a, p - l.a) > 0;
65 }
66 template<class T>
67 Point<T> lineIntersection(const Line<T> &l1, const Line<T> &l2) {
68     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1
       .a - l1.b));
69 }
70 template<class T>
71 bool pointOnSegment(const Point<T> &p, const Line<T> &l) {
72     return cross(p - l.a, l.b - l.a) == 0 && min(l.a.x, l.b.x) <= p.x && p.x <= max(l.a.x,
       l.b.x)
       && min(l.a.y, l.b.y) <= p.y && p.y <= max(l.a.y, l.b.y);
73 }
74 template<class T>
75 bool pointInPolygon(const Point<T> &a, const vector<Point<T>> &p) {
76     int n = p.size();
77     for (int i = 0; i < n; i++) {
78         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
79             return true;
80         }
81     }
82     int t = 0;
83     for (int i = 0; i < n; i++) {
84         auto u = p[i];
85         auto v = p[(i + 1) % n];
86         if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
87             t ^= 1;
88         }
89         if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
90             t ^= 1;
91         }
92     }
93     return t == 1;
94 }
95 // 0 : not intersect
96 // 1 : strictly intersect
97 // 2 : overlap
98 // 3 : intersect at endpoint
99 template<class T>
100 tuple<int, Point<T>, Point<T>> segmentIntersection(const Line<T> &l1, const Line<T> &l2) {
101     if (max(l1.a.x, l1.b.x) < min(l2.a.x, l2.b.x)) {
102         return {0, Point<T>(), Point<T>()};
103     }
104     if (min(l1.a.x, l1.b.x) > max(l2.a.x, l2.b.x)) {
105         return {0, Point<T>(), Point<T>()};
106     }
107     if (max(l1.a.y, l1.b.y) < min(l2.a.y, l2.b.y)) {
108         return {0, Point<T>(), Point<T>()};
109     }
110     if (min(l1.a.y, l1.b.y) > max(l2.a.y, l2.b.y)) {
111         return {0, Point<T>(), Point<T>()};
112     }

```

```

113     }
114     if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
115         if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
116             return {0, Point<T>(), Point<T>()};
117         } else {
118             auto maxx1 = max(l1.a.x, l1.b.x);
119             auto minx1 = min(l1.a.x, l1.b.x);
120             auto maxy1 = max(l1.a.y, l1.b.y);
121             auto miny1 = min(l1.a.y, l1.b.y);
122             auto maxx2 = max(l2.a.x, l2.b.x);
123             auto minx2 = min(l2.a.x, l2.b.x);
124             auto maxy2 = max(l2.a.y, l2.b.y);
125             auto miny2 = min(l2.a.y, l2.b.y);
126             Point<T> p1(max(minx1, minx2), max(miny1, miny2));
127             Point<T> p2(min(maxx1, maxx2), min(maxy1, maxy2));
128             if (!pointOnSegment(p1, l1)) {
129                 swap(p1.y, p2.y);
130             }
131             if (p1 == p2) {
132                 return {3, p1, p2};
133             } else {
134                 return {2, p1, p2};
135             }
136         }
137     }
138     auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
139     auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
140     auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
141     auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
142     if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) || (cp3 < 0
143         && cp4 < 0)) {
144         return {0, Point<T>(), Point<T>()};
145     }
146     Point p = lineIntersection(l1, l2);
147     if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
148         return {1, p, p};
149     } else {
150         return {3, p, p};
151     }
152     template<class T>
153     double distanceSS(const Line<T> &l1, const Line<T> &l2) {
154         if (get<0>(segmentIntersection(l1, l2)) != 0) {
155             return 0.0;
156         }
157         return min({distancePS(l1.a, l2), distancePS(l1.b, l2), distancePS(l2.a, l1),
158             distancePS(l2.b, l1)});
159     }
160     template<class T>
161     bool segmentInPolygon(const Line<T> &l, const vector<Point<T>> &p) {
162         int n = p.size();
163         if (!pointInPolygon(l.a, p)) {
164             return false;
165         }
166         if (!pointInPolygon(l.b, p)) {
167             return false;
168         }
169         for (int i = 0; i < n; i++) {
170             auto u = p[i];
171             auto v = p[(i + 1) % n];
172             auto w = p[(i + 2) % n];
173             auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
174             if (t == 1) {
175                 return false;
176             }
177         }
178     }

```

```

175         }
176     if (t == 0) {
177         continue;
178     }
179     if (t == 2) {
180         if (pointOnSegment(v, l) && v != l.a && v != l.b) {
181             if (cross(v - u, w - v) > 0) {
182                 return false;
183             }
184         }
185     } else {
186         if (p1 != u && p1 != v) {
187             if (pointOnLineLeft(l.a, Line(v, u))
188                 || pointOnLineLeft(l.b, Line(v, u))) {
189                 return false;
190             }
191         } else if (p1 == v) {
192             if (l.a == v) {
193                 if (pointOnLineLeft(u, l)) {
194                     if (pointOnLineLeft(w, l)
195                         && pointOnLineLeft(w, Line(u, v))) {
196                         return false;
197                     }
198                 } else {
199                     if (pointOnLineLeft(w, l)
200                         || pointOnLineLeft(w, Line(u, v))) {
201                         return false;
202                     }
203                 }
204             } else if (l.b == v) {
205                 if (pointOnLineLeft(u, Line(l.b, l.a))) {
206                     if (pointOnLineLeft(w, Line(l.b, l.a))
207                         && pointOnLineLeft(w, Line(u, v))) {
208                         return false;
209                     }
210                 } else {
211                     if (pointOnLineLeft(w, Line(l.b, l.a))
212                         || pointOnLineLeft(w, Line(u, v))) {
213                         return false;
214                     }
215                 }
216             } else {
217                 if (pointOnLineLeft(u, l)) {
218                     if (pointOnLineLeft(w, Line(l.b, l.a))
219                         || pointOnLineLeft(w, Line(u, v))) {
220                             return false;
221                         }
222                 } else {
223                     if (pointOnLineLeft(w, l)
224                         || pointOnLineLeft(w, Line(u, v))) {
225                             return false;
226                         }
227                 }
228             }
229         }
230     }
231 }
232 return true;
233 }
234 template<class T>
235 vector<Point<T>> hp(vector<Line<T>> lines) {
236     sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
237         auto d1 = l1.b - l1.a;
238         auto d2 = l2.b - l2.a;

```

```

239         if (sgn(d1) != sgn(d2)) {
240             return sgn(d1) == 1;
241         }
242         return cross(d1, d2) > 0;
243     });
244     deque<Line<T>> ls;
245     deque<Point<T>> ps;
246     for (auto l : lines) {
247         if (ls.empty()) {
248             ls.push_back(l);
249             continue;
250         }
251         while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
252             ps.pop_back();
253             ls.pop_back();
254         }
255         while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
256             ps.pop_front();
257             ls.pop_front();
258         }
259         if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
260             if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
261                 if (!pointOnLineLeft(ls.back().a, l)) {
262                     assert(ls.size() == 1);
263                     ls[0] = l;
264                 }
265                 continue;
266             }
267             return {};
268         }
269         ps.push_back(lineIntersection(ls.back(), l));
270         ls.push_back(l);
271     }
272     while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
273         ps.pop_back();
274         ls.pop_back();
275     }
276     if (ls.size() <= 2) {
277         return {};
278     }
279     ps.push_back(lineIntersection(ls[0], ls.back()));
280     return vector(ps.begin(), ps.end());
281 }
282 using P = Point<i64>;
283 using i128 = __int128;
284 int main() {
285     ios::sync_with_stdio(false);
286     cin.tie(nullptr);
287     int n;
288     cin >> n;
289     vector<int> a(n), b(n);
290     for (int i = 0; i < n; i++) {
291         cin >> a[i];
292     }
293     for (int i = 0; i < n; i++) {
294         cin >> b[i];
295     }
296     cout << fixed << setprecision(12);
297     auto get = [&](P a, P b) {
298         double y1 = a.y + 1.0 * (b.y - a.y) / (b.x - a.x) * (0.0 - a.x);
299         double y2 = a.y + 1.0 * (b.y - a.y) / (b.x - a.x) * (n - 1 - a.x);
300         return y1 + y2;
301     };
302     vector<P> left;

```

```

303     vector<double> prel;
304     vector<P> right(n);
305     int l = n;
306     vector<double> sufr(n);
307     vector<int> oldl(n);
308     vector<double> olds(n);
309     vector<P> oldr(n);
310     for (int i = n - 1; i >= 0; i--) {
311         oldl[i] = l;
312         P p(i, a[i]);
313         while (l <= n - 2 && cross(right[l] - p, right[l + 1] - right[l]) >= 0) {
314             l++;
315         }
316         l--;
317         olds[i] = exchange(sufr[l], l == n - 1 ? INFINITY : min(sufr[l + 1], get(p, right[
318             l + 1))));
318         oldr[i] = exchange(right[l], p);
319     }
320     for (int i = 0; i < n; i++) {
321         P p(i, b[i]);
322         while (left.size() >= 2 && cross(left.back() - left[left.size() - 2], p - left.
323             back()) >= 0) {
324             left.pop_back();
325             prel.pop_back();
326         }
327         prel.push_back(left.empty() ? INFINITY : min(prel.back(), get(left.back(), p)));
328         left.push_back(p);
329         right[l] = oldr[i];
330         sufr[l] = olds[i];
331         l = oldl[i];
332         double ans;
333         if (i == n - 1) {
334             ans = prel.back();
335         } else {
336             int lo1 = 0, hi1 = left.size() - 1;
337             int lo2 = l, hi2 = n - 1;
338             int k1, k2;
339             while (true) {
340                 int i1 = (lo1 + hi1 + 1) / 2;
341                 int i2 = (lo2 + hi2) / 2;
342                 bool int1 = false, int2 = false;
343                 if (i1 < hi1 && cross(left[i1 + 1] - left[i1], right[i2] - left[i1]) < 0)
344                 {
345                     int1 = true;
346                 }
347                 if (i2 > lo2 && cross(right[i2 - 1] - right[i2], left[i1] - right[i2]) >
348                     0) {
349                     int2 = true;
350                 }
351                 bool tan1 = true, tan2 = true;
352                 if (i1 > lo1 && cross(left[i1] - left[i1 - 1], right[i2] - left[i1]) >=
353                     0)
354                 {
355                     tan1 = false;
356                 }
357                 if (i2 < hi2 && cross(right[i2] - left[i1], right[i2 + 1] - right[i2]) >=
358                     0)
359                 {
360                     tan2 = false;
361                 }
362                 if (!tan1 || !tan2) {
363                     if (!tan1) {
364                         hi1 = i1 - 1;
365                     }
366                     if (!tan2) {
367                         lo2 = i2 + 1;
368                     }
369                 }
370             }
371         }
372     }
373 }

```

```

361         }
362     } else {
363         if (!int1 && !int2) {
364             k1 = i1, k2 = i2;
365             break;
366         }
367         if (int1 && int2) {
368             i128 y1 = i128(left[i1].y) * (right[i2 - 1].x - right[i2].x) * (
369                 left[i1 + 1].x - left[i1].x) + i128(left[i1 + 1].y - left[i1].
370                 y) * (i - left[i1].x) * (right[i2 - 1].x - right[i2].x);
371             i128 y2 = i128(right[i2].y) * (left[i1 + 1].x - left[i1].x) * (
372                 right[i2 - 1].x - right[i2].x) + i128(right[i2 - 1].y - right[i2].
373                 y) * (i - right[i2].x) * (left[i1 + 1].x - left[i1].x);
374             if (y1 <= y2) {
375                 lo1 = i1;
376             } else {
377                 hi2 = i2;
378             }
379         }
380     }
381     assert(k1 == 0 || cross(left[k1] - left[k1 - 1], right[k2] - left[k1]) < 0);
382     assert(k2 == n - 1 || cross(right[k2] - left[k1], right[k2 + 1] - right[k2]) <
383         0);
384     assert(k1 == left.size() - 1 || cross(left[k1 + 1] - left[k1], right[k2] -
385         left[k1]) >= 0);
386     assert(k2 == l || cross(right[k2 - 1] - right[k2], left[k1] - right[k2]) <= 0)
387         ;
388     ans = min(pre[pre[k1]], sufr[sufr[k2]]);
389     ans = min(ans, get(left[k1], right[k2]));
390 }
391 cout << ans << "\n"[i == n - 1];
392 }
393 return 0;
394 }
```

### 873: Symmetric Projections

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a set of  $n$  points on the plane. A line containing the origin is called good, if projection of the given set to this line forms a symmetric multiset of points. Find the total number of good lines.

Multiset is a set where equal elements are allowed.

Multiset is called symmetric, if there is a point  $P$  on the plane such that the multiset is centrally symmetric in respect of point  $P$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<i64> x(n), y(n);
10    i64 gx = 0, gy = 0;
11    for (int i = 0; i < n; i++) {
12        cin >> x[i] >> y[i];
13        gx += x[i];
14        gy += y[i];
15        x[i] *= n;
16        y[i] *= n;
17    }
18    vector<bool> good(n);
19    for (int i = 0; i < n; i++) {
20        for (int j = 0; j < n; j++) {
21            if (x[i] + x[j] == 2 * gx && y[i] + y[j] == 2 * gy) {
22                good[i] = good[j] = true;
23            }
24        }
25    }
26    vector<int> p;
27    for (int i = 0; i < n; i++) {
28        if (!good[i]) {
29            p.push_back(i);
30        }
31    }
32    if (p.empty()) {
33        cout << -1 << "\n";
34        return 0;
35    }
36    set<array<i64, 2>> s;
37    for (int i = 0; i < p.size(); i++) {
38        i64 vx = y[p[0]] + y[p[i]] - 2 * gy;
39        i64 vy = 2 * gx - x[p[0]] - x[p[i]];
40        vector<i64> a(n);
41        for (int i = 0; i < n; i++) {
42            a[i] = vx * (x[i] - gx) + vy * (y[i] - gy);
43        }
44        sort(a.begin(), a.end());
45        bool ok = true;
46        for (int i = 0; i < n; i++) {
47            if (a[i] + a[n - 1 - i] != 0) {
48                ok = false;
49            }
50        }
51        if (ok) {
52            i64 g = gcd(vx, vy);
53            vx /= g, vy /= g;
54            if (vx < 0 || (vx == 0 && vy < 0)) {
55                vx *= -1, vy *= -1;
56            }
57            s.insert({vx, vy});
58        }
59    }
60    int ans = s.size();
61    cout << ans << "\n";
62    return 0;

```

```
63 }
```

## 874: Monocarp and a Strategic Game

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp plays a strategic computer game in which he develops a city. The city is inhabited by creatures of four different races - humans, elves, orcs, and dwarves.

Each inhabitant of the city has a happiness value, which is an integer. It depends on how many creatures of different races inhabit the city. Specifically, the happiness of each inhabitant is 0 by default; it increases by 1 for each other creature of the same race and decreases by 1 for each creature of a hostile race. Humans are hostile to orcs (and vice versa), and elves are hostile to dwarves (and vice versa).

At the beginning of the game, Monocarp's city is not inhabited by anyone. During the game,  $n$  groups of creatures will come to his city, wishing to settle there. The  $i$ -th group consists of  $a_i$  humans,  $b_i$  orcs,  $c_i$  elves, and  $d_i$  dwarves. Each time, Monocarp can either accept the entire group of creatures into the city, or reject the entire group.

The game calculates Monocarp's score according to the following formula:  $m + k$ , where  $m$  is the number of inhabitants in the city, and  $k$  is the sum of the happiness values of all creatures in the city.

Help Monocarp earn the maximum possible number of points by the end of the game!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 # struct Point;
6 template<class T>
7 T dot(Point<T> a, Point<T> b) {
8     return a.x * b.x + a.y * b.y;
9 }
10 template<class T>
11 T cross(Point<T> a, Point<T> b) {
12     return a.x * b.y - a.y * b.x;
13 }
14 template<class T>
15 T square(Point<T> p) {
```

```

16     return dot(p, p);
17 }
18 template<class T>
19 double length(Point<T> p) {
20     return sqrt(double(square(p)));
21 }
22 long double length(Point<long double> p) {
23     return sqrt(square(p));
24 }
25 template<class T>
26 # struct Line;
27 template<class T>
28 Point<T> rotate(Point<T> a) {
29     return Point(-a.y, a.x);
30 }
31 template<class T>
32 int sgn(Point<T> a) {
33     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
34 }
35 template<class T>
36 bool pointOnLineLeft(Point<T> p, Line<T> l) {
37     return cross(l.b - l.a, p - l.a) > 0;
38 }
39 template<class T>
40 Point<T> lineIntersection(Line<T> l1, Line<T> l2) {
41     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1
        .a - l1.b));
42 }
43 template<class T>
44 bool pointOnSegment(Point<T> p, Line<T> l) {
45     return cross(p - l.a, l.b - l.a) == 0 && min(l.a.x, l.b.x) <= p.x && p.x <= max(l.a.x,
        l.b.x)
        && min(l.a.y, l.b.y) <= p.y && p.y <= max(l.a.y, l.b.y);
46 }
47 template<class T>
48 bool pointInPolygon(Point<T> a, vector<Point<T>> p) {
49     int n = p.size();
50     for (int i = 0; i < n; i++) {
51         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
52             return true;
53         }
54     }
55     int t = 0;
56     for (int i = 0; i < n; i++) {
57         auto u = p[i];
58         auto v = p[(i + 1) % n];
59         if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
60             t ^= 1;
61         }
62         if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
63             t ^= 1;
64         }
65     }
66     return t == 1;
67 }
68 // 0 : not intersect
69 // 1 : strictly intersect
70 // 2 : overlap
71 // 3 : intersect at endpoint
72 template<class T>
73 tuple<int, Point<T>, Point<T>> segmentIntersection(Line<T> l1, Line<T> l2) {
74     if (max(l1.a.x, l1.b.x) < min(l2.a.x, l2.b.x)) {
75         return {0, Point<T>(), Point<T>()};
76     }

```

```

78     if (min(l1.a.x, l1.b.x) > max(l2.a.x, l2.b.x)) {
79         return {0, Point<T>(), Point<T>()};
80     }
81     if (max(l1.a.y, l1.b.y) < min(l2.a.y, l2.b.y)) {
82         return {0, Point<T>(), Point<T>()};
83     }
84     if (min(l1.a.y, l1.b.y) > max(l2.a.y, l2.b.y)) {
85         return {0, Point<T>(), Point<T>()};
86     }
87     if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
88         if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
89             return {0, Point<T>(), Point<T>()};
90         } else {
91             auto maxx1 = max(l1.a.x, l1.b.x);
92             auto minx1 = min(l1.a.x, l1.b.x);
93             auto maxy1 = max(l1.a.y, l1.b.y);
94             auto miny1 = min(l1.a.y, l1.b.y);
95             auto maxx2 = max(l2.a.x, l2.b.x);
96             auto minx2 = min(l2.a.x, l2.b.x);
97             auto maxy2 = max(l2.a.y, l2.b.y);
98             auto miny2 = min(l2.a.y, l2.b.y);
99             Point<T> p1(max(minx1, minx2), max(miny1, miny2));
100            Point<T> p2(min(maxx1, maxx2), min(maxy1, maxy2));
101            if (!pointOnSegment(p1, l1)) {
102                swap(p1.y, p2.y);
103            }
104            if (p1 == p2) {
105                return {3, p1, p2};
106            } else {
107                return {2, p1, p2};
108            }
109        }
110    }
111    auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
112    auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
113    auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
114    auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
115    if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) || (cp3 < 0
116        && cp4 < 0)) {
117        return {0, Point<T>(), Point<T>()};
118    }
119    Point p = lineIntersection(l1, l2);
120    if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
121        return {1, p, p};
122    } else {
123        return {3, p, p};
124    }
125    template<class T>
126    bool segmentInPolygon(Line<T> l, vector<Point<T>> p) {
127        int n = p.size();
128        if (!pointInPolygon(l.a, p)) {
129            return false;
130        }
131        if (!pointInPolygon(l.b, p)) {
132            return false;
133        }
134        for (int i = 0; i < n; i++) {
135            auto u = p[i];
136            auto v = p[(i + 1) % n];
137            auto w = p[(i + 2) % n];
138            auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
139            if (t == 1) {
140                return false;

```

```

141         }
142         if (t == 0) {
143             continue;
144         }
145         if (t == 2) {
146             if (pointOnSegment(v, l) && v != l.a && v != l.b) {
147                 if (cross(v - u, w - v) > 0) {
148                     return false;
149                 }
150             }
151         } else {
152             if (p1 != u && p1 != v) {
153                 if (pointOnLineLeft(l.a, Line(v, u))
154                     || pointOnLineLeft(l.b, Line(v, u))) {
155                     return false;
156                 }
157             } else if (p1 == v) {
158                 if (l.a == v) {
159                     if (pointOnLineLeft(u, l)) {
160                         if (pointOnLineLeft(w, l)
161                             && pointOnLineLeft(w, Line(u, v))) {
162                             return false;
163                         }
164                     } else {
165                         if (pointOnLineLeft(w, l)
166                             || pointOnLineLeft(w, Line(u, v))) {
167                             return false;
168                         }
169                     }
170                 } else if (l.b == v) {
171                     if (pointOnLineLeft(u, Line(l.b, l.a))) {
172                         if (pointOnLineLeft(w, Line(l.b, l.a))
173                             && pointOnLineLeft(w, Line(u, v))) {
174                             return false;
175                         }
176                     } else {
177                         if (pointOnLineLeft(w, Line(l.b, l.a))
178                             || pointOnLineLeft(w, Line(u, v))) {
179                             return false;
180                         }
181                     }
182                 } else {
183                     if (pointOnLineLeft(u, l)) {
184                         if (pointOnLineLeft(w, Line(l.b, l.a))
185                             || pointOnLineLeft(w, Line(u, v))) {
186                             return false;
187                         }
188                     } else {
189                         if (pointOnLineLeft(w, l)
190                             || pointOnLineLeft(w, Line(u, v))) {
191                             return false;
192                         }
193                     }
194                 }
195             }
196         }
197     }
198     return true;
199 }
200 template<class T>
201 vector<Point<T>> hp(vector<Line<T>> lines) {
202     sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
203         auto d1 = l1.b - l1.a;
204         auto d2 = l2.b - l2.a;

```

```
205     if (sgn(d1) != sgn(d2)) {
206         return sgn(d1) == 1;
207     }
208     return cross(d1, d2) > 0;
209 });
210 deque<Line<T>> ls;
211 deque<Point<T>> ps;
212 for (auto l : lines) {
213     if (ls.empty()) {
214         ls.push_back(l);
215         continue;
216     }
217     while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
218         ps.pop_back();
219         ls.pop_back();
220     }
221     while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
222         ps.pop_front();
223         ls.pop_front();
224     }
225     if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
226         if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
227             if (!pointOnLineLeft(ls.back().a, l)) {
228                 assert(ls.size() == 1);
229                 ls[0] = l;
230             }
231             continue;
232         }
233         return {};
234     }
235     ps.push_back(lineIntersection(ls.back(), l));
236     ls.push_back(l);
237 }
238 while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
239     ps.pop_back();
240     ls.pop_back();
241 }
242 if (ls.size() <= 2) {
243     return {};
244 }
245 ps.push_back(lineIntersection(ls[0], ls.back()));
246 return vector(ps.begin(), ps.end());
247 }
248 using i128 = __int128;
249 using P = Point<i128>;
250 int main() {
251     ios::sync_with_stdio(false);
252     cin.tie(nullptr);
253     int n;
254     cin >> n;
255     vector<P> p;
256     for (int i = 0; i < n; i++) {
257         int a, b, c, d;
258         cin >> a >> b >> c >> d;
259         int x = a - b, y = c - d;
260         if (x != 0 || y != 0) {
261             p.emplace_back(x, y);
262         }
263     }
264     sort(p.begin(), p.end(),
265          [&](auto a, auto b) {
266              if (sgn(a) != sgn(b)) {
267                  return sgn(a) == 1;
```

```

269         return cross(a, b) > 0;
270     });
271     i128 ans = 0;
272     n = p.size();
273     P sum;
274     for (int i = 0, j = 0; i < n; i++) {
275         while (j < i + n && (cross(p[i], p[j % n]) > 0 || (cross(p[i], p[j % n]) == 0 &&
276             dot(p[i], p[j % n]) > 0))) {
277             sum += p[j % n];
278             ans = max(ans, dot(sum, sum));
279             j++;
280         }
281         sum -= p[i];
282         ans = max(ans, dot(sum, sum));
283     }
284     sum = 0;
285     reverse(p.begin(), p.end());
286     for (int i = 0, j = 0; i < n; i++) {
287         while (j < i + n && (cross(p[i], p[j % n]) < 0 || (cross(p[i], p[j % n]) == 0 &&
288             dot(p[i], p[j % n]) < 0))) {
289             sum += p[j % n];
290             ans = max(ans, dot(sum, sum));
291             j++;
292         }
293         sum -= p[i];
294         ans = max(ans, dot(sum, sum));
295     }
296     string s;
297     if (ans == 0) {
298         s = "0";
299     } else {
300         while (ans > 0) {
301             s += ans % 10 + '0';
302             ans /= 10;
303         }
304     }
305     cout << s << "\n";
306 }

```

### 875: Stuck Conveyor

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an interactive problem.

There is an  $n$  by  $n$  grid of conveyor belts, in positions  $(1, 1)$  through  $(n, n)$  of a coordinate plane. Every other square in the plane is empty. Each conveyor belt can be configured to move boxes up ('^'), down ('v'), left ('<'), or right ('>'). If a box moves onto an empty square, it stops moving.

However, one of the  $n^2$  belts is stuck, and will always move boxes in the same direction, no matter how

it is configured. Your goal is to perform a series of tests to determine which conveyor belt is stuck, and the direction in which it sends items.

To achieve this, you can perform up to 25 tests. In each test, you assign a direction to all  $n^2$  belts, place a box on top of one of them, and then turn all of the conveyors on.

The conveyors move the box around too quickly for you to see, so the only information you receive from a test is whether the box eventually stopped moving, and if so, the coordinates of its final position.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 string chdir = "^<>";
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n;
9     cin >> n;
10    vector s(n, vector(n, 0));
11    for (int i = 0; i < n; i++) {
12        if (i % 2 == 0) {
13            for (int j = 0; j < n - 1; j++) {
14                s[i][j] = 3;
15            }
16            s[i][n - 1] = 2;
17        } else {
18            s[i][0] = 2;
19            for (int j = 1; j < n; j++) {
20                s[i][j] = 1;
21            }
22        }
23    }
24    auto rotates = [&](auto &s) {
25        vector ns(n, vector(n, 0));
26        for (int i = 0; i < n; i++) {
27            for (int j = 0; j < n; j++) {
28                ns[n - 1 - j][i] = (s[i][j] + 1) % 4;
29            }
30        }
31        s = ns;
32    };
33    auto rotate = [&](auto &x, auto &y) {
34        if (x != -2) {
35            tie(x, y) = make_pair(n - 1 - y, x);
36        }
37    };
38    auto query = [&](auto s, auto x, auto y) {
39        cout << "? " << x + 1 << " " << y + 1 << endl;
40        for (int i = 0; i < n; i++) {
41            for (int j = 0; j < n; j++) {
42                cout << chdir[s[i][j]];
43            }
44            cout << endl;
45        }
46        cin >> x >> y;
47        return make_pair(x - 1, y - 1);
48    };

```

```
49     int dir = 0;
50     while (dir < 3) {
51         auto a = s;
52         int x = 0, y = 0;
53         for (int j = 0; j < dir; j++) {
54             rotates(a);
55             rotate(x, y);
56         }
57         auto [rx, ry] = query(a, x, y);
58         for (int i = 0; i < 4 - dir; i++) {
59             rotate(rx, ry);
60         }
61         if (rx == -1) {
62             break;
63         }
64         if (rx != -2) {
65             dir++;
66             continue;
67         }
68         a = s;
69         for (auto &v : a) {
70             reverse(v.begin(), v.end());
71             for (auto &x : v) {
72                 x = (4 - x) % 4;
73             }
74         }
75         x = 0, y = 0;
76         for (int j = 0; j < dir; j++) {
77             rotates(a);
78             rotate(x, y);
79         }
80         tie(rx, ry) = query(a, x, y);
81         for (int i = 0; i < 4 - dir; i++) {
82             rotate(rx, ry);
83         }
84         if (rx == -2) {
85             break;
86         }
87         dir++;
88     }
89     int lo = 0, hi = n * n - 1;
90     while (lo < hi) {
91         int m = (lo + hi + 1) / 2;
92         auto a = s;
93         int x = m / n, y = x % 2 == 0 ? m % n : n - m % n - 1;
94         for (int j = 0; j < dir; j++) {
95             rotates(a);
96             rotate(x, y);
97         }
98         auto [rx, ry] = query(a, x, y);
99         for (int i = 0; i < 4 - dir; i++) {
100             rotate(rx, ry);
101         }
102         if (rx == -1 || rx == -2) {
103             lo = m;
104         } else {
105             hi = m - 1;
106         }
107     }
108     int x = lo / n, y = x % 2 == 0 ? lo % n : n - lo % n - 1;
109     for (int j = 0; j < dir; j++) {
110         rotate(x, y);
111     }
112     cout << "! " << x + 1 << " " << y + 1 << " " << chdir[dir] << endl;
```

```
113     return 0;
114 }
```

## 876: Spider

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A plane contains a not necessarily convex polygon without self-intersections, consisting of  $n$  vertexes, numbered from 1 to  $n$ . There is a spider sitting on the border of the polygon, the spider can move like that:

Transfer. The spider moves from the point  $p_1$  with coordinates  $(x_1, y_1)$ , lying on the polygon border, to the point  $p_2$  with coordinates  $(x_2, y_2)$ , also lying on the border. The spider can't go beyond the polygon border as it transfers, that is, the spider's path from point  $p_1$  to point  $p_2$  goes along the polygon border. It's up to the spider to choose the direction of walking round the polygon border (clockwise or counterclockwise).

Descend. The spider moves from point  $p_1$  with coordinates  $(x_1, y_1)$  to point  $p_2$  with coordinates  $(x_2, y_2)$ , at that points  $p_1$  and  $p_2$  must lie on one vertical straight line ( $x_1 = x_2$ ), point  $p_1$  must be not lower than point  $p_2$  ( $y_1 \geq y_2$ ) and segment  $p_1p_2$  mustn't have points, located strictly outside the polygon (specifically, the segment can have common points with the border).

Initially the spider is located at the polygon vertex with number  $s$ . Find the length of the shortest path to the vertex number  $t$ , consisting of transfers and descends. The distance is determined by the usual Euclidean metric .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> x(n), y(n);
10    for (int i = 0; i < n; i++) {
11        cin >> x[i] >> y[i];
12    }
13    int s, t;
```

```

14     cin >> s >> t;
15     s--, t--;
16     vector<map<int, int>> segment(n);
17     int cur = n;
18     map<int, vector<int>> events;
19     for (int i = 0; i < n; i++) {
20         int a = i, b = (i + 1) % n;
21         if (x[a] == x[b]) {
22             continue;
23         }
24         events[x[a]].push_back(i);
25         events[x[b]].push_back(i);
26         segment[i][x[a]] = a;
27         segment[i][x[b]] = b;
28     }
29     vector<vector<pair<int, double>>> adj(n);
30     auto cmp = [&](int i, int j) {
31         if (i == j) {
32             return false;
33         }
34         int ai = i, bi = (i + 1) % n;
35         int aj = j, bj = (j + 1) % n;
36         if (x[ai] > x[bi]) {
37             swap(ai, bi);
38         }
39         if (x[aj] > x[bj]) {
40             swap(aj, bj);
41         }
42         int L = max(x[ai], x[aj]);
43         int R = min(x[bi], x[bj]);
44         double fiL = y[ai] + 1.0 * (y[bi] - y[ai]) / (x[bi] - x[ai]) * (L - x[ai]);
45         double fjL = y[aj] + 1.0 * (y[bj] - y[aj]) / (x[bj] - x[aj]) * (L - x[aj]);
46         if (fiL != fjL) {
47             // cerr << "L : " << L << "\n";
48             // cerr << "cmp " << i << " " << j << " " << fiL << " " << fjL << "\n";
49             return fiL < fjL;
50         }
51         if (L == R) {
52             return i < j;
53         }
54         double fiR = y[ai] + 1.0 * (y[bi] - y[ai]) / (x[bi] - x[ai]) * (R - x[ai]);
55         double fjR = y[aj] + 1.0 * (y[bj] - y[aj]) / (x[bj] - x[aj]) * (R - x[aj]);
56         return fiR < fjR;
57     };
58     set<int, decltype(cmp)> order(cmp);
59     auto getId = [&](int i, int x) {
60         int res;
61         if (!segment[i].count(x)) {
62             res = segment[i][x] = cur++;
63             adj.emplace_back();
64         } else {
65             res = segment[i][x];
66         }
67         return res;
68     };
69     auto addEdge = [&](int a, int b, double w) {
70         // cerr << "addEdge " << a << " " << b << " " << w << "\n";
71         adj[a].emplace_back(b, w);
72     };
73     for (auto [X, v] : events) {
74         // cerr << "X : " << X << "\n";
75         for (auto i : v) {
76             int a = i, b = (i + 1) % n;
77             if (X == min(x[a], x[b])) {

```

```

78         // cerr << "insert " << i << "\n";
79         order.insert(i);
80     }
81 }
82 // for (auto i : order) {
83 //     cerr << i << " ";
84 // }
85 // cerr << "\n";
86 for (auto i : v) {
87     int a = i, b = (i + 1) % n;
88     if (X == min(x[a], x[b])) {
89         auto it = order.find(i);
90         if (X == x[a]) {
91             it++;
92             assert(it != order.end());
93             int j = *it;
94             int aj = j, bj = (j + 1) % n;
95             double fJX = y[aj] + 1.0 * (y[bj] - y[aj]) / (x[bj] - x[aj]) * (X - x[aj]);
96             addEdge(getId(j, X), a, fJX - y[a]);
97         } else {
98             assert(it != order.begin());
99             it--;
100            int j = *it;
101            int aj = j, bj = (j + 1) % n;
102            double fJX = y[aj] + 1.0 * (y[bj] - y[aj]) / (x[bj] - x[aj]) * (X - x[aj]);
103            addEdge(b, getId(j, X), y[b] - fJX);
104        }
105    }
106 }
107 for (auto i : v) {
108     int a = i, b = (i + 1) % n;
109     if (X == max(x[a], x[b])) {
110         auto it = order.find(i);
111         if (X == x[b]) {
112             it++;
113             assert(it != order.end());
114             int j = *it;
115             int aj = j, bj = (j + 1) % n;
116             double fJX = y[aj] + 1.0 * (y[bj] - y[aj]) / (x[bj] - x[aj]) * (X - x[aj]);
117             addEdge(getId(j, X), b, fJX - y[b]);
118         } else {
119             assert(it != order.begin());
120             it--;
121             int j = *it;
122             int aj = j, bj = (j + 1) % n;
123             double fJX = y[aj] + 1.0 * (y[bj] - y[aj]) / (x[bj] - x[aj]) * (X - x[aj]);
124             addEdge(a, getId(j, X), y[a] - fJX);
125         }
126     }
127 }
128 for (auto i : v) {
129     int a = i, b = (i + 1) % n;
130     if (X == max(x[a], x[b])) {
131         order.erase(i);
132     }
133 }
134 }
135 for (int i = 0; i < n; i++) {
136     int a = i, b = (i + 1) % n;
137     double d = sqrt((x[a] - x[b]) * (x[a] - x[b]) + (y[a] - y[b]) * (y[a] - y[b]));

```

```

138     if (segment[i].empty()) {
139         addEdge(a, b, d);
140         addEdge(b, a, d);
141     } else {
142         int dx = abs(x[a] - x[b]);
143         for (auto it = segment[i].begin(); next(it) != segment[i].end(); it++) {
144             auto [x1, i1] = *it;
145             auto [x2, i2] = *next(it);
146             addEdge(i1, i2, d / dx * (x2 - x1));
147             addEdge(i2, i1, d / dx * (x2 - x1));
148         }
149     }
150 }
151 int N = adj.size();
152 vector dis(N, -1.0);
153 priority_queue<pair<double, int>, vector<pair<double, int>>, greater<>> q;
154 q.emplace(0.0, s);
155 while (!q.empty()) {
156     auto [d, x] = q.top();
157     q.pop();
158     if (dis[x] != -1) {
159         continue;
160     }
161     dis[x] = d;
162     for (auto [y, w] : adj[x]) {
163         q.emplace(d + w, y);
164     }
165 }
166 double ans = dis[t];
167 cout << fixed << setprecision(10);
168 cout << ans << "\n";
169 return 0;
170 }
```

## 877: Fading into Fog

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an interactive problem.

There are  $n$  distinct hidden points with real coordinates on a two-dimensional Euclidean plane. In one query, you can ask some line  $ax + by + c = 0$  and get the projections of all  $n$  points to this line in some order. The given projections are not exact, please read the interaction section for more clarity.

Using the minimum number of queries, guess all  $n$  points and output them in some order. Here minimality means the minimum number of queries required to solve any possible test case with  $n$  points.

The hidden points are fixed in advance and do not change throughout the interaction. In other words, the interactor is not adaptive.

A projection of point  $A$  to line  $ax + by + c = 0$  is the point on the line closest to  $A$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
5 constexpr double Pi = numbers::pi;
6 void solve() {
7     int n;
8     cin >> n;
9     vector<double> x(n), y(n);
10    cout << "? 1 0 0" << endl;
11    for (int i = 0; i < n; i++) {
12        double x;
13        cin >> x >> y[i];
14    }
15    cout << "? 0 1 0" << endl;
16    for (int i = 0; i < n; i++) {
17        double y;
18        cin >> x[i] >> y;
19    }
20    double ang;
21    double A, B;
22    vector<double> f;
23    while (true) {
24        ang = 1. * rng() / numeric_limits<unsigned>::max() * Pi * 2;
25        bool ok = true;
26        A = cos(ang);
27        B = sin(ang);
28        f.clear();
29        for (int i = 0; i < n; i++) {
30            for (int j = 0; j < n; j++) {
31                f.push_back(x[i] * A + y[j] * B);
32            }
33        }
34        sort(f.begin(), f.end());
35        for (int i = 1; i < f.size(); i++) {
36            if (f[i] - f[i - 1] < 1E-3) {
37                ok = false;
38            }
39        }
40        if (ok) {
41            break;
42        }
43    }
44    cout << "? " << B << " " << -A << " " << 0 << endl;
45    cout << "!";
46    for (int i = 0; i < n; i++) {
47        double X, Y;
48        cin >> X >> Y;
49        double f = X * A + Y * B;
50        for (int u = 0; u < n; u++) {
51            for (int v = 0; v < n; v++) {
52                if (abs(f - (x[u] * A + y[v] * B)) < 5E-4) {
53                    cout << " " << x[u] << " " << y[v];
54                }
55            }
56        }
57    }
58}
```

```

56         }
57     }
58     cout << endl;
59 }
60 int main() {
61     ios::sync_with_stdio(false);
62     cin.tie(nullptr);
63     cout << fixed << setprecision(10);
64     int t;
65     cin >> t;
66     while (t--) {
67         solve();
68     }
69     return 0;
70 }
```

## 878: Graph Game

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

In computer science, there is a method called “Divide And Conquer By Node” to solve some hard problems about paths on a tree. Let’s describe how this method works by function:

`solve(t)` (`t` is a tree):

Choose a node `x` (it’s common to choose weight-center) in tree `t`. Let’s call this step “Line A”.

Deal with all paths that pass `x`.

Then delete `x` from tree `t`.

After that `t` becomes some subtrees.

Apply `solve` on each subtree.

This ends when `t` has only one node because after deleting it, there’s nothing.

Now, WJMZBMR has mistakenly believed that it’s ok to choose any node in “Line A”. So he’ll choose a node at random. To make the situation worse, he thinks a “tree” should have the same number of edges and nodes! So this procedure becomes like that.

Let’s define the variable `totalCost`. Initially the value of `totalCost` equal to 0. So, `solve(t)` (now `t` is a graph):

`totalCost = totalCost + (size of t)`. The operation “=” means assignment. (Size of `t`) means the number of nodes in `t`.

Choose a node `x` in graph `t` at random (uniformly among all nodes of `t`).

Then delete x from graph t.

After that t becomes some connected components.

Apply solve on each component.

He'll apply solve on a connected graph with n nodes and n edges. He thinks it will work quickly, but it's very slow. So he wants to know the expectation of totalCost of this procedure. Can you help him?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     cout << fixed << setprecision(10);
10    vector<int> deg(n);
11    vector<vector<int>> adj(n);
12    for (int i = 0; i < n; i++) {
13        int u, v;
14        cin >> u >> v;
15        adj[u].push_back(v);
16        adj[v].push_back(u);
17        deg[u]++, deg[v]++;
18    }
19    double ans = 0;
20    queue<int> q;
21    for (int i = 0; i < n; i++) {
22        if (deg[i] == 1) {
23            q.push(i);
24        }
25    }
26    vector cyc(n, true);
27    while (!q.empty()) {
28        int x = q.front();
29        q.pop();
30        cyc[x] = false;
31        for (auto y : adj[x]) {
32            if (deg[y] && --deg[y] == 1) {
33                q.push(y);
34            }
35        }
36    }
37    vector<int> a;
38    vector<bool> vis(n);
39    function<void(int)> dfs = [&](int x) {
40        a.push_back(x);
41        vis[x] = true;
42        for (auto y : adj[x]) {
43            if (cyc[y] && !vis[y]) {
44                dfs(y);
45            }
46        }
47    };

```

```

48     dfs(find(cyc.begin(), cyc.end(), true) - cyc.begin()));
49     vector<int> pos(n, -1), dep(n);
50     int L = a.size();
51     for (int i = 0; i < L; i++) {
52         pos[a[i]] = i;
53     }
54     dfs = [&](int x) {
55         for (auto y : adj[x]) {
56             if (pos[y] == -1) {
57                 pos[y] = pos[x];
58                 dep[y] = dep[x] + 1;
59                 dfs(y);
60             }
61         }
62     };
63     for (auto x : a) {
64         dfs(x);
65     }
66     function<void(int, int, int)> dfs1 = [&](int x, int p, int d) {
67         ans += 1. / d;
68         for (auto y : adj[x]) {
69             if (y != p && pos[x] == pos[y]) {
70                 dfs1(y, x, d + 1);
71             }
72         }
73     };
74     for (int i = 0; i < n; i++) {
75         dfs1(i, -1, 1);
76     }
77     for (int i = 0; i < n; i++) {
78         for (int j = 0; j < n; j++) {
79             if (pos[i] != pos[j]) {
80                 int l = abs(pos[i] - pos[j]);
81                 ans += 1. / (dep[i] + dep[j] + l + 1);
82                 ans += 1. / (dep[i] + dep[j] + L - l + 1);
83                 ans -= 1. / (dep[i] + dep[j] + L);
84             }
85         }
86     }
87     cout << ans << "\n";
88     return 0;
89 }
```

## 879: Toy Machine

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

There is a toy machine with toys arranged in two rows of  $n$  cells each ( $n$  is odd).

Initially,  $n - 2$  toys are placed in the non-corner cells of the top row. The bottom row is initially empty, and its leftmost, rightmost, and central cells are blocked. There are 4 buttons to control the toy machine: left, right, up, and down marked by the letters L, R, U, and D correspondingly.

When pressing L, R, U, or D, all the toys will be moved simultaneously in the corresponding direction and will only stop if they push into another toy, the wall or a blocked cell. Your goal is to move the  $k$ -th toy into the leftmost cell of the top row. The toys are numbered from 1 to  $n - 2$  from left to right. Given  $n$  and  $k$ , find a solution that uses at most 1 000 000 button presses.

To test out the toy machine, a web page is available that lets you play the game in real time.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3 using namespace std::literals::string_literals;
4 int main() {
5     std::ios::sync_with_stdio(false);
6     std::cin.tie(nullptr);
7     int n, k;
8     std::cin >> n >> k;
9     n -= 2;
10    std::string s;
11    if (k == (n + 1) / 2) {
12        s = "DL"s;
13    } else if (k < (n + 1) / 2) {
14        for (int i = 0; i < k - 1; i++) {
15            s += "LDRU";
16        }
17        s += "L";
18    } else {
19        for (int i = 0; i < n - k; i++) {
20            s += "RDLU";
21        }
22        for (int i = 0; i < n; i++) {
23            s += "LDLU";
24        }
25        s += "RDL";
26    }
27    std::cout << s << "\n";
28    return 0;
29 }
```

## 880: Petya and Rectangle

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Little Petya loves playing with rectangles. Mom bought Petya a rectangle divided into cells  $n \times m$  in size (containing  $n$  rows,  $m$  columns). Petya marked two different cells of the rectangle and now he is solving the following task:

Let's define a simple path between those two cells as a sequence of distinct cells  $a_1, a_2, \dots, a_k$ , where

$a_1$  and  $a_k$  are the two marked cells. Besides,  $a_i$  and  $a_{i+1}$  are side-neighboring cells of the path ( $1 \leq i < k$ ). Let's denote the path length as number  $k$  (the sequence length).

Petya's task is to find the longest simple path's length and to print the path. Help him.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void g(vector<pair<int, int>> &ans, int x1, int y1, int x2, int y2, int tx, int ty);
5 void f(vector<pair<int, int>> &ans, int x1, int y1, int x2, int y2, int tx, int ty) {
6     if (x1 > x2 || y1 > y2) {
7         return;
8     }
9     if (y1 == y2) {
10         ans.emplace_back(tx ? x2 : x1, y1);
11         return;
12     }
13     if (tx) {
14         for (int i = x2; i >= x1; i--) {
15             ans.emplace_back(i, ty ? y2 : y1);
16         }
17     } else {
18         for (int i = x1; i <= x2; i++) {
19             ans.emplace_back(i, ty ? y2 : y1);
20         }
21     }
22     if (ty) {
23         y2--;
24     } else {
25         y1++;
26     }
27     g(ans, x1, y1, x2, y2, !tx, ty);
28 }
29 void g(vector<pair<int, int>> &ans, int x1, int y1, int x2, int y2, int tx, int ty) {
30     if (x1 > x2 || y1 > y2) {
31         return;
32     }
33     if (x1 == x2) {
34         if (ty == 0) {
35             for (int i = y1; i <= y2; i++) {
36                 ans.emplace_back(x1, i);
37             }
38         } else {
39             for (int i = y2; i >= y1; i--) {
40                 ans.emplace_back(x1, i);
41             }
42         }
43     }
44     if (y1 == y2) {
45         if (tx == 0) {
46             for (int i = x1; i <= x2; i++) {
47                 ans.emplace_back(i, y1);
48             }
49         } else {
50             for (int i = x2; i >= x1; i--) {
51                 ans.emplace_back(i, y1);
52             }
53         }
54     }

```

```

55         return;
56     }
57     if (x1 + 1 == x2 && y1 + 1 == y2) {
58         ans.emplace_back(tx ? x2 : x1, ty ? y2 : y1);
59         ans.emplace_back(tx ? x1 : x2, ty ? y2 : y1);
60         ans.emplace_back(tx ? x1 : x2, ty ? y1 : y2);
61         return;
62     }
63     if (x2 - x1 > y2 - y1) {
64         for (int v = 0; v < 2; v++) {
65             if (ty ^ v) {
66                 for (int i = y2; i >= y1; i--) {
67                     ans.emplace_back(tx ? x2 : x1, i);
68                 }
69             } else {
70                 for (int i = y1; i <= y2; i++) {
71                     ans.emplace_back(tx ? x2 : x1, i);
72                 }
73             }
74             if (tx) {
75                 x2--;
76             } else {
77                 x1++;
78             }
79         }
80         g(ans, x1, y1, x2, y2, tx, ty);
81     } else {
82         for (int v = 0; v < 2; v++) {
83             if (tx ^ v) {
84                 for (int i = x2; i >= x1; i--) {
85                     ans.emplace_back(i, ty ? y2 : y1);
86                 }
87             } else {
88                 for (int i = x1; i <= x2; i++) {
89                     ans.emplace_back(i, ty ? y2 : y1);
90                 }
91             }
92             if (ty) {
93                 y2--;
94             } else {
95                 y1++;
96             }
97         }
98         g(ans, x1, y1, x2, y2, tx, ty);
99     }
100 }
101 int main() {
102     ios::sync_with_stdio(false);
103     cin.tie(nullptr);
104     int n, m, x1, y1, x2, y2;
105     cin >> n >> m >> x1 >> y1 >> x2 >> y2;
106     vector<pair<int, int>> ans;
107     int req;
108     if (n * m % 2 == 0) {
109         if ((x1 + y1 + x2 + y2) % 2 == 0) {
110             req = n * m - 1;
111         } else {
112             req = n * m;
113         }
114     } else {
115         req = n * m - ((x1 + y1) % 2) - ((x2 + y2) % 2);
116     }
117     for (int fxy = 0; fxy < 2; fxy++) {
118         for (int fx = 0; fx < 2; fx++) {

```

```

119         for (int fy = 0; fy < 2; fy++) {
120             for (int rev = 0; rev < 2; rev++) {
121                 int N = n, M = m, X1 = x1, Y1 = y1, X2 = x2, Y2 = y2;
122                 if (fxy) {
123                     swap(N, M);
124                     swap(X1, Y1);
125                     swap(X2, Y2);
126                 }
127                 if (fy) {
128                     Y1 = M + 1 - Y1;
129                     Y2 = M + 1 - Y2;
130                 }
131                 if (fx) {
132                     X1 = N + 1 - X1;
133                     X2 = N + 1 - X2;
134                 }
135                 if (rev) {
136                     swap(X1, X2);
137                     swap(Y1, Y2);
138                 }
139                 if (ans.size() != req) {
140                     ans.clear();
141                     f(ans, 1, 1, X1, Y1, 1, 1);
142                     // cerr << ans.size() << " ";
143                     g(ans, X1 + 1, 1, N, Y1, 0, 0);
144                     // cerr << ans.size() << " ";
145                     f(ans, 1, Y1 + 1, N, Y2 - 1, 1, 0);
146                     // cerr << ans.size() << " ";
147                     g(ans, X2 + 1, Y2, N, M, 1, 0);
148                     // cerr << ans.size() << " ";
149                     f(ans, 1, Y2, X2, M, 1, 1);
150                     // cerr << ans.size() << "\n";
151             }
152             if (ans.size() != req && Y1 + 1 < Y2) {
153                 ans.clear();
154                 f(ans, 1, 1, X1, Y1, 1, 1);
155                 // cerr << ans.size() << " ";
156                 g(ans, X1 + 1, 1, N, Y1, 0, 0);
157                 // cerr << ans.size() << " ";
158                 g(ans, 1, Y1 + 1, N, Y2 - 1, 1, 0);
159                 // cerr << ans.size() << " ";
160                 g(ans, 1, Y2, X2 - 1, M, 0, 0);
161                 // cerr << ans.size() << " ";
162                 f(ans, X2, Y2, N, M, 0, 1);
163                 // cerr << ans.size() << "\n";
164             }
165             if (ans.size() != req) {
166                 continue;
167             }
168             if (rev) {
169                 reverse(ans.begin(), ans.end());
170             }
171             if (fx) {
172                 for (auto &[x, y] : ans) {
173                     x = N + 1 - x;
174                 }
175             }
176             if (fy) {
177                 for (auto &[x, y] : ans) {
178                     y = M + 1 - y;
179                 }
180             }
181             if (fxy) {
182                 for (auto &[x, y] : ans) {

```

```

183                     swap(x, y);
184                 }
185             }
186             cout << ans.size() << "\n";
187             for (auto [x, y] : ans) {
188                 cout << x << " " << y << "\n";
189             }
190             return 0;
191         }
192     }
193 }
194 return 0;
195
196 }
```

## 881: Edge coloring of bipartite graph

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an undirected bipartite graph without multiple edges. You should paint the edges of graph to minimal number of colours, so that no two adjacent edges have the same colour.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int a, b, m;
8     cin >> a >> b >> m;
9     vector<vector<pair<int, int>>> adj(a+b);
10    vector<int> x(m), y(m);
11    for (int i = 0; i < m; i++) {
12        cin >> x[i] >> y[i];
13        x[i]--, y[i]--;
14        y[i] += a;
15        adj[x[i]].emplace_back(y[i], i);
16        adj[y[i]].emplace_back(x[i], i);
17    }
18    int d = 0;
19    for (int i = 0; i < a+b; i++) {
20        d = max(d, int(adj[i].size()));
21    }
22    vector f(a+b, vector(d, -1));
23    vector c(m, -1);
24    for (int i = 0; i < m; i++) {
25        vector<int> cx, cy;
26        for (int k = 0; k < d; k++) {
27            if (f[x[i]][k] != -1 && f[y[i]][k] == -1) {
28                cx.push_back(k);
29            }
30        }
31        for (int k = 0; k < d; k++) {
32            if (f[x[i]][k] == -1 && f[y[i]][k] != -1) {
33                cy.push_back(k);
34            }
35        }
36        for (int k = 0; k < d; k++) {
37            if (f[x[i]][k] != -1 && f[y[i]][k] != -1) {
38                if (cx[k] < cy[k]) {
39                    swap(cx[k], cy[k]);
40                }
41                if (cx[k] < cy[k]) {
42                    swap(cx[k], cy[k]);
43                }
44            }
45        }
46        for (int k = 0; k < d; k++) {
47            if (f[x[i]][k] == -1 && f[y[i]][k] == -1) {
48                if (cx[k] < cy[k]) {
49                    swap(cx[k], cy[k]);
50                }
51            }
52        }
53    }
54    for (int i = 0; i < a+b; i++) {
55        for (int j = 0; j < d; j++) {
56            if (f[i][j] == -1) {
57                cout << j << " ";
58            }
59        }
60    }
61}
```

```

29         }
30         if (f[x[i]][k] == -1 && f[y[i]][k] != -1) {
31             cy.push_back(k);
32         }
33     }
34     for (int j = 0; j < min(cx.size(), cy.size()); j++) {
35         int u = cx[j], v = cy[j];
36         int t = x[i];
37         vector<int> e;
38         while (f[t][u] != -1) {
39             assert(0 <= t && t < a+b);
40             assert(0 <= u && u < d);
41             int k = f[t][u];
42             e.push_back(k);
43             f[x[k]][u] = f[y[k]][u] = -1;
44             swap(u, v);
45             t = x[k] ^ y[k] ^ t;
46         }
47         for (auto k : e) {
48             c[k] ^= u ^ v;
49             f[x[k]][c[k]] = f[y[k]][c[k]] = k;
50         }
51     }
52     for (int k = 0; k < d; k++) {
53         if (f[x[i]][k] == -1 && f[y[i]][k] == -1) {
54             c[i] = k;
55             f[x[i]][k] = f[y[i]][k] = i;
56             break;
57         }
58     }
59 }
60 cout << d << "\n";
61 for (int i = 0; i < m; i++) {
62     cout << c[i]+1 << " \n"[i == m-1];
63 }
64 return 0;
65 }

```

## 882: Cut Length

- Time limit: 0.5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Given simple (without self-intersections) n-gon. It is not necessary convex. Also you are given m lines. For each line find the length of common part of the line and the n-gon.

The boundary of n-gon belongs to polygon. It is possible that n-gon contains 180-degree angles.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;

```

```

3  using i64 = long long;
4  template<class T>
5  # struct Point;
6  template<class T>
7  T dot(Point<T> a, Point<T> b) {
8      return a.x * b.x + a.y * b.y;
9  }
10 template<class T>
11 T cross(Point<T> a, Point<T> b) {
12     return a.x * b.y - a.y * b.x;
13 }
14 template<class T>
15 T square(Point<T> p) {
16     return dot(p, p);
17 }
18 template<class T>
19 double length(Point<T> p) {
20     return sqrt(double(square(p)));
21 }
22 long double length(Point<long double> p) {
23     return sqrt(square(p));
24 }
25 template<class T>
26 # struct Line;
27 template<class T>
28 Point<T> rotate(Point<T> a) {
29     return Point(-a.y, a.x);
30 }
31 template<class T>
32 int sgn(Point<T> a) {
33     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
34 }
35 template<class T>
36 bool pointOnLineLeft(Point<T> p, Line<T> l) {
37     return cross(l.b - l.a, p - l.a) > 0;
38 }
39 template<class T>
40 Point<T> lineIntersection(Line<T> l1, Line<T> l2) {
41     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1
42 .a - l1.b));
43 }
44 constexpr long double eps = 1E-8;
45 template<class T>
46 bool pointOnSegment(Point<T> p, Line<T> l) {
47     return abs(cross(p - l.a, l.b - l.a)) < eps && min(l.a.x, l.b.x) < p.x + eps && p.x -
48         eps < max(l.a.x, l.b.x)
49         && min(l.a.y, l.b.y) < p.y + eps && p.y - eps < max(l.a.y, l.b.y);
50 }
51 template<class T>
52 bool pointInPolygon(Point<T> a, vector<Point<T>> p) {
53     int n = p.size();
54     for (int i = 0; i < n; i++) {
55         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
56             return true;
57         }
58         int t = 0;
59         for (int i = 0; i < n; i++) {
60             auto u = p[i];
61             auto v = p[(i + 1) % n];
62             if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
63                 t ^= 1;
64             }
65             if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
66                 t ^= 1;
67             }
68         }
69         if (t > 0) {
70             return true;
71         }
72     }
73 }

```

```

65             t ^= 1;
66         }
67     }
68     return t == 1;
69 }
70 // 0 : not intersect
71 // 1 : strictly intersect
72 // 2 : overlap
73 // 3 : intersect at endpoint
74 template<class T>
75 tuple<int, Point<T>, Point<T>> segmentIntersection(Line<T> l1, Line<T> l2) {
76     if (max(l1.a.x, l1.b.x) < min(l2.a.x, l2.b.x)) {
77         return {0, Point<T>(), Point<T>()};
78     }
79     if (min(l1.a.x, l1.b.x) > max(l2.a.x, l2.b.x)) {
80         return {0, Point<T>(), Point<T>()};
81     }
82     if (max(l1.a.y, l1.b.y) < min(l2.a.y, l2.b.y)) {
83         return {0, Point<T>(), Point<T>()};
84     }
85     if (min(l1.a.y, l1.b.y) > max(l2.a.y, l2.b.y)) {
86         return {0, Point<T>(), Point<T>()};
87     }
88     if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
89         if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
90             return {0, Point<T>(), Point<T>()};
91         } else {
92             auto maxx1 = max(l1.a.x, l1.b.x);
93             auto minx1 = min(l1.a.x, l1.b.x);
94             auto maxy1 = max(l1.a.y, l1.b.y);
95             auto miny1 = min(l1.a.y, l1.b.y);
96             auto maxx2 = max(l2.a.x, l2.b.x);
97             auto minx2 = min(l2.a.x, l2.b.x);
98             auto maxy2 = max(l2.a.y, l2.b.y);
99             auto miny2 = min(l2.a.y, l2.b.y);
100            Point<T> p1(max(minx1, minx2), max(miny1, miny2));
101            Point<T> p2(min(maxx1, maxx2), min(maxy1, maxy2));
102            if (!pointOnSegment(p1, l1)) {
103                swap(p1.y, p2.y);
104            }
105            if (p1 == p2) {
106                return {3, p1, p2};
107            } else {
108                return {2, p1, p2};
109            }
110        }
111    }
112    auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
113    auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
114    auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
115    auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
116    if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) || (cp3 < 0
117        && cp4 < 0)) {
118        return {0, Point<T>(), Point<T>()};
119    }
120    Point p = lineIntersection(l1, l2);
121    if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
122        return {1, p, p};
123    } else {
124        return {3, p, p};
125    }
126 template<class T>
127 bool segmentInPolygon(Line<T> l, vector<Point<T>> p) {

```

```
128     int n = p.size();
129     if (!pointInPolygon(l.a, p)) {
130         return false;
131     }
132     if (!pointInPolygon(l.b, p)) {
133         return false;
134     }
135     for (int i = 0; i < n; i++) {
136         auto u = p[i];
137         auto v = p[(i + 1) % n];
138         auto w = p[(i + 2) % n];
139         auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
140         if (t == 1) {
141             return false;
142         }
143         if (t == 0) {
144             continue;
145         }
146         if (t == 2) {
147             if (pointOnSegment(v, l) && v != l.a && v != l.b) {
148                 if (cross(v - u, w - v) > 0) {
149                     return false;
150                 }
151             }
152         } else {
153             if (p1 != u && p1 != v) {
154                 if (pointOnLineLeft(l.a, Line(v, u))
155                     || pointOnLineLeft(l.b, Line(v, u))) {
156                     return false;
157                 }
158             } else if (p1 == v) {
159                 if (l.a == v) {
160                     if (pointOnLineLeft(u, l)) {
161                         if (pointOnLineLeft(w, l)
162                             && pointOnLineLeft(w, Line(u, v))) {
163                             return false;
164                         }
165                     } else {
166                         if (pointOnLineLeft(w, l)
167                             || pointOnLineLeft(w, Line(u, v))) {
168                             return false;
169                         }
170                     }
171                 } else if (l.b == v) {
172                     if (pointOnLineLeft(u, Line(l.b, l.a))) {
173                         if (pointOnLineLeft(w, Line(l.b, l.a))
174                             && pointOnLineLeft(w, Line(u, v))) {
175                             return false;
176                         }
177                     } else {
178                         if (pointOnLineLeft(w, Line(l.b, l.a))
179                             || pointOnLineLeft(w, Line(u, v))) {
180                             return false;
181                         }
182                     }
183                 } else {
184                     if (pointOnLineLeft(u, l)) {
185                         if (pointOnLineLeft(w, Line(l.b, l.a))
186                             || pointOnLineLeft(w, Line(u, v))) {
187                             return false;
188                         }
189                     } else {
190                         if (pointOnLineLeft(w, l)
191                             || pointOnLineLeft(w, Line(u, v))) {
```

```

192                     return false;
193                 }
194             }
195         }
196     }
197 }
198 return true;
199 }
200 }
201 template<class T>
202 vector<Point<T>> hp(vector<Line<T>> lines) {
203     sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
204         auto d1 = l1.b - l1.a;
205         auto d2 = l2.b - l2.a;
206         if (sgn(d1) != sgn(d2)) {
207             return sgn(d1) == 1;
208         }
209         return cross(d1, d2) > 0;
210     });
211     deque<Line<T>> ls;
212     deque<Point<T>> ps;
213     for (auto l : lines) {
214         if (ls.empty()) {
215             ls.push_back(l);
216             continue;
217         }
218         while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
219             ps.pop_back();
220             ls.pop_back();
221         }
222         while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
223             ps.pop_front();
224             ls.pop_front();
225         }
226         if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
227             if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
228                 if (!pointOnLineLeft(ls.back().a, l)) {
229                     assert(ls.size() == 1);
230                     ls[0] = l;
231                 }
232                 continue;
233             }
234             return {};
235         }
236         ps.push_back(lineIntersection(ls.back(), l));
237         ls.push_back(l);
238     }
239     while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
240         ps.pop_back();
241         ls.pop_back();
242     }
243     if (ls.size() <= 2) {
244         return {};
245     }
246     ps.push_back(lineIntersection(ls[0], ls.back()));
247     return vector(ps.begin(), ps.end());
248 }
249 template<class T>
250 # struct Frac;
251 using P = Point<Frac<__int128>>;
252 int main() {
253     ios::sync_with_stdio(false);
254     cin.tie(nullptr);
255     int n, m;

```

```

256     cin >> n >> m;
257     vector<P> p(n);
258     for (int i = 0; i < n; i++) {
259         double x, y;
260         cin >> x >> y;
261         p[i] = {round(x * 100), round(y * 100)};
262     }
263     vector<P> q;
264     for (auto p : p) {
265         if (q.size() >= 2 && cross(p - q.back(), q.back() - q.end()[-2]) == 0) {
266             q.pop_back();
267         }
268         q.push_back(p);
269     }
270     if (cross(q[0] - q[1], q.back() - q[0]) == 0) {
271         q.erase(q.begin());
272     }
273     if (cross(q[0] - q.back(), q.back() - q.end()[-2]) == 0) {
274         q.pop_back();
275     }
276     p = q;
277     n = p.size();
278     cout << fixed << setprecision(10);
279     while (m--) {
280         P a, b;
281         double ax, ay, bx, by;
282         cin >> ax >> ay >> bx >> by;
283         a = {round(ax * 100), round(ay * 100)};
284         b = {round(bx * 100), round(by * 100)};
285         double ans = 0;
286         Line l(a, b);
287         vector<Point<double>> q, r;
288         vector<pair<Point<double>, int>> e;
289         for (int i = 0; i < n; i++) {
290             Point c = p[i], d = p[(i+1) % n];
291             if (cross(a - b, c - d) == 0) {
292                 if (cross(a - b, a - c) == 0) {
293                     r.push_back(c);
294                     r.push_back(d);
295                 }
296                 continue;
297             }
298             if (pointOnLineLeft(c, l) != pointOnLineLeft(d, l)) {
299                 auto p = lineIntersection(l, Line(c, d));
300                 q.push_back(p);
301             }
302         }
303         sort(q.begin(), q.end(), [&](auto c, auto d) {
304             return dot(c, Point<double>(b) - Point<double>(a)) < dot(d, Point<double>(b) -
305             Point<double>(a));
306         });
307         sort(r.begin(), r.end(), [&](auto c, auto d) {
308             return dot(c, Point<double>(b) - Point<double>(a)) < dot(d, Point<double>(b) -
309             Point<double>(a));
310         });
311         int t = 0;
312         for (int i = 0; i+1 < q.size(); i += 2) {
313             e.emplace_back(q[i], 1);
314             e.emplace_back(q[i+1], -1);
315         }
316         for (int i = 0; i+1 < r.size(); i += 2) {
317             e.emplace_back(r[i], 1);
318             e.emplace_back(r[i+1], -1);
319         }

```

```

318     sort(e.begin(), e.end(), [&](auto c, auto d) {
319         return dot(c.first, Point<double>(b) - Point<double>(a)) < dot(d.first, Point<
320             double>(b) - Point<double>(a));
321     });
322     for (int i = 0; i < e.size(); i++) {
323         if (t > 0) {
324             ans += length(e[i].first - e[i-1].first);
325         }
326         t += e[i].second;
327     }
328     ans /= 100;
329     cout << ans << "\n";
330 }
331 }
```

### 883: Parquet Re-laying

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Peter decided to lay a parquet in the room of size  $n \times m$ , the parquet consists of tiles of size  $1 \times 2$ . When the workers laid the parquet, it became clear that the tiles pattern looks not like Peter likes, and workers will have to re-lay it.

The workers decided that removing entire parquet and then laying it again is very difficult task, so they decided to make such an operation every hour: remove two tiles, which form a  $2 \times 2$  square, rotate them 90 degrees and put them back on the same place.

They have no idea how to obtain the desired configuration using these operations, and whether it is possible at all.

Help Peter to make a plan for the workers or tell that it is impossible. The plan should contain at most 100 000 commands.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int dx[] = {-1, 1, 0, 0};
5 constexpr int dy[] = {0, 0, -1, 1};
6 int main() {
7     ios::sync_with_stdio(false);
8     cin.tie(nullptr);
9     int n, m;
```

```

10     cin >> n >> m;
11     vector<string> a(n), b(n);
12     for (int i = 0; i < n; i++) {
13         cin >> a[i];
14     }
15     for (int i = 0; i < n; i++) {
16         cin >> b[i];
17     }
18     vector f(n+1, vector(m+1, 0));
19     for (int i = 1; i <= n; i++) {
20         for (int j = 1; j <= m; j++) {
21             f[i][j] = f[i-1][j] ^ (a[i-1][j-1] == 'L') ^ (b[i-1][j-1] == 'L');
22         }
23     }
24     vector dis(n+1, vector(m+1, -1));
25     deque<tuple<int, int, int>> q;
26     for (int i = 0; i <= n; i++) {
27         for (int j = 0; j <= m; j++) {
28             if (i == 0 || j == 0 || i == n || j == m) {
29                 q.emplace_back(i, j, 0);
30             }
31         }
32     }
33     auto valid = [&](int x, int y) {
34         return 0 <= x && x <= n && 0 <= y && y <= m;
35     };
36     while (!q.empty()) {
37         auto [x, y, d] = q.front();
38         q.pop_front();
39         if (dis[x][y] != -1) {
40             continue;
41         }
42         dis[x][y] = d;
43         for (int k = 0; k < 4; k++) {
44             int nx = x + dx[k];
45             int ny = y + dy[k];
46             if (valid(nx, ny)) {
47                 if (f[nx][ny] == f[x][y]) {
48                     q.emplace_front(nx, ny, d);
49                 } else {
50                     q.emplace_back(nx, ny, d+1);
51                 }
52             }
53         }
54     }
55     vector<pair<int, int>> ansL, ansR;
56     while (true) {
57         int max = 0;
58         for (int i = 0; i <= n; i++) {
59             for (int j = 0; j <= m; j++) {
60                 max = max(max, dis[i][j]);
61             }
62         }
63         if (!max) {
64             break;
65         }
66         for (int i = 0; i <= n; i++) {
67             for (int j = 0; j <= m; j++) {
68                 if (dis[i][j] == max) {
69                     bool adj = false;
70                     for (int k = 0; k < 4; k++) {
71                         int nx = i + dx[k];
72                         int ny = j + dy[k];
73                         if (valid(nx, ny) && dis[nx][ny] == max-1) {

```

```

74             adj = true;
75         }
76     }
77     if (!adj) {
78         continue;
79     }
80     if (a[i-1][j-1] == 'L' && a[i-1][j] == 'R' && a[i][j-1] == 'L' && a[i]
81         ][j] == 'R') {
82         a[i-1][j-1] = 'U';
83         a[i-1][j] = 'U';
84         a[i][j-1] = 'D';
85         a[i][j] = 'D';
86         dis[i][j] -= 1;
87         ansL.emplace_back(i, j);
88     } else if (b[i-1][j-1] == 'L' && b[i-1][j] == 'R' && b[i][j-1] == 'L'
89         && b[i][j] == 'R') {
90         b[i-1][j-1] = 'U';
91         b[i-1][j] = 'U';
92         b[i][j-1] = 'D';
93         b[i][j] = 'D';
94         dis[i][j] -= 1;
95         ansR.emplace_back(i, j);
96     } else if (a[i-1][j-1] == 'U' && a[i-1][j] == 'U' && a[i][j-1] == 'D'
97         && a[i][j] == 'D') {
98         a[i-1][j-1] = 'L';
99         a[i-1][j] = 'R';
100        a[i][j-1] = 'L';
101        a[i][j] = 'R';
102        dis[i][j] -= 1;
103        ansL.emplace_back(i, j);
104    } else if (b[i-1][j-1] == 'U' && b[i-1][j] == 'U' && b[i][j-1] == 'D'
105         && b[i][j] == 'D') {
106         b[i-1][j-1] = 'L';
107         b[i-1][j] = 'R';
108         b[i][j-1] = 'L';
109         b[i][j] = 'R';
110         dis[i][j] -= 1;
111     }
112     }
113     // for (int i = 0; i < n; i++) {
114     //     cerr << a[i] << "\n";
115     // }
116     // cerr << "\n";
117     // for (int i = 0; i < n; i++) {
118     //     cerr << b[i] << "\n";
119     // }
120     // cerr << "\n";
121     // for (int i = 0; i <= n; i++) {
122     //     for (int j = 0; j <= m; j++) {
123     //         cerr << dis[i][j];
124     //     }
125     //     cerr << "\n";
126     // }
127     // cerr << "\n";
128 }
129 ansL.insert(ansL.end(), ansR.rbegin(), ansR.rend());
130 cout << ansL.size() << "\n";
131 for (auto [x, y] : ansL) {
132     cout << x << " " << y << "\n";
133 }
134 return 0;

```

```
134 }
```

## 884: DravDe saves the world

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

How horrible! The empire of galactic chickens tries to conquer a beautiful city “Z”, they have built a huge incubator that produces millions of chicken soldiers a day, and fenced it around. The huge incubator looks like a polygon on the plane Oxy with  $n$  vertices. Naturally, DravDe can't keep still, he wants to destroy the chicken empire. For sure, he will start with the incubator.

DravDe is strictly outside the incubator's territory in point A( $x_a, y_a$ ), and wants to get inside and kill all the chickens working there. But it takes a lot of doing! The problem is that recently DravDe went roller skating and has broken both his legs. He will get to the incubator's territory in his jet airplane LEVAP-41.

LEVAP-41 flies at speed  $V(x_v, y_v, z_v)$ . DravDe can get on the plane in point A, fly for some time, and then air drop himself. DravDe is very heavy, that's why he falls vertically at speed  $F_{down}$ , but in each point of his free fall DravDe can open his parachute, and from that moment he starts to fall at the wind speed  $U(x_u, y_u, z_u)$  until he lands. Unfortunately, DravDe isn't good at mathematics. Would you help poor world's saviour find such an air dropping plan, that allows him to land on the incubator's territory? If the answer is not unique, DravDe wants to find the plan with the minimum time of his flight on the plane. If the answers are still multiple, he wants to find the one with the minimum time of his free fall before opening his parachute

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 # struct Frac;
6 template<class T>
7 # struct Point;
8 template<class T>
9 T dot(Point<T> a, Point<T> b) {
10     return a.x * b.x + a.y * b.y;
11 }
12 template<class T>
13 T cross(Point<T> a, Point<T> b) {
```

```

14     return a.x * b.y - a.y * b.x;
15 }
16 template<class T>
17 T square(Point<T> p) {
18     return dot(p, p);
19 }
20 template<class T>
21 double length(Point<T> p) {
22     return sqrt(double(square(p)));
23 }
24 long double length(Point<long double> p) {
25     return sqrt(square(p));
26 }
27 template<class T>
28 # struct Line;
29 template<class T>
30 Point<T> rotate(Point<T> a) {
31     return Point(-a.y, a.x);
32 }
33 template<class T>
34 int sgn(Point<T> a) {
35     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
36 }
37 template<class T>
38 bool pointOnLineLeft(Point<T> p, Line<T> l) {
39     return cross(l.b - l.a, p - l.a) > 0;
40 }
41 template<class T>
42 Point<T> lineIntersection(Line<T> l1, Line<T> l2) {
43     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1
        .a - l1.b));
44 }
45 template<class T>
46 bool pointOnSegment(Point<T> p, Line<T> l) {
47     const long double eps = 1E-6;
48     return abs(cross(p - l.a, l.b - l.a)) < eps && min(l.a.x, l.b.x) < p.x + eps && p.x -
        eps < max(l.a.x, l.b.x)
49         && min(l.a.y, l.b.y) < p.y + eps && p.y - eps < max(l.a.y, l.b.y);
50 }
51 template<class T>
52 bool pointInPolygon(Point<T> a, vector<Point<T>> p) {
53     int n = p.size();
54     for (int i = 0; i < n; i++) {
55         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
56             return true;
57         }
58     }
59     int t = 0;
60     for (int i = 0; i < n; i++) {
61         auto u = p[i];
62         auto v = p[(i + 1) % n];
63         if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
64             t ^= 1;
65         }
66         if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
67             t ^= 1;
68         }
69     }
70     return t == 1;
71 }
72 // 0 : not intersect
73 // 1 : strictly intersect
74 // 2 : overlap
75 // 3 : intersect at endpoint

```

```

76  template<class T>
77  tuple<int, Point<T>, Point<T>> segmentIntersection(Line<T> l1, Line<T> l2) {
78      if (max(l1.a.x, l1.b.x) < min(l2.a.x, l2.b.x)) {
79          return {0, Point<T>(), Point<T>()};
80      }
81      if (min(l1.a.x, l1.b.x) > max(l2.a.x, l2.b.x)) {
82          return {0, Point<T>(), Point<T>()};
83      }
84      if (max(l1.a.y, l1.b.y) < min(l2.a.y, l2.b.y)) {
85          return {0, Point<T>(), Point<T>()};
86      }
87      if (min(l1.a.y, l1.b.y) > max(l2.a.y, l2.b.y)) {
88          return {0, Point<T>(), Point<T>()};
89      }
90      if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
91          if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
92              return {0, Point<T>(), Point<T>()};
93          } else {
94              int maxx1 = max(l1.a.x, l1.b.x);
95              int minx1 = min(l1.a.x, l1.b.x);
96              int maxy1 = max(l1.a.y, l1.b.y);
97              int miny1 = min(l1.a.y, l1.b.y);
98              int maxx2 = max(l2.a.x, l2.b.x);
99              int minx2 = min(l2.a.x, l2.b.x);
100             int maxy2 = max(l2.a.y, l2.b.y);
101             int miny2 = min(l2.a.y, l2.b.y);
102             Point<T> p1(max(minx1, minx2), max(miny1, miny2));
103             Point<T> p2(min(maxx1, maxx2), min(maxy1, maxy2));
104             if (!pointOnSegment(p1, l1)) {
105                 swap(p1.y, p2.y);
106             }
107             if (p1 == p2) {
108                 return {3, p1, p2};
109             } else {
110                 return {2, p1, p2};
111             }
112         }
113     }
114     auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
115     auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
116     auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
117     auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
118     if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) || (cp3 < 0
119         && cp4 < 0)) {
120         return {0, Point<T>(), Point<T>()};
121     }
122     Point p = lineIntersection(l1, l2);
123     if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
124         return {1, p, p};
125     } else {
126         return {3, p, p};
127     }
128 template<class T>
129 bool segmentInPolygon(Line<T> l, vector<Point<T>> p) {
130     int n = p.size();
131     if (!pointInPolygon(l.a, p)) {
132         return false;
133     }
134     if (!pointInPolygon(l.b, p)) {
135         return false;
136     }
137     for (int i = 0; i < n; i++) {
138         auto u = p[i];

```

```
139         auto v = p[(i + 1) % n];
140         auto w = p[(i + 2) % n];
141         auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
142         if (t == 1) {
143             return false;
144         }
145         if (t == 0) {
146             continue;
147         }
148         if (t == 2) {
149             if (pointOnSegment(v, l) && v != l.a && v != l.b) {
150                 if (cross(v - u, w - v) > 0) {
151                     return false;
152                 }
153             }
154         } else {
155             if (p1 != u && p1 != v) {
156                 if (pointOnLineLeft(l.a, Line(v, u))
157                     || pointOnLineLeft(l.b, Line(v, u))) {
158                     return false;
159                 }
160             } else if (p1 == v) {
161                 if (l.a == v) {
162                     if (pointOnLineLeft(u, l)) {
163                         if (pointOnLineLeft(w, l)
164                             && pointOnLineLeft(w, Line(u, v))) {
165                             return false;
166                         }
167                     } else {
168                         if (pointOnLineLeft(w, l)
169                             || pointOnLineLeft(w, Line(u, v))) {
170                             return false;
171                         }
172                     }
173                 } else if (l.b == v) {
174                     if (pointOnLineLeft(u, Line(l.b, l.a))) {
175                         if (pointOnLineLeft(w, Line(l.b, l.a))
176                             && pointOnLineLeft(w, Line(u, v))) {
177                             return false;
178                         }
179                     } else {
180                         if (pointOnLineLeft(w, Line(l.b, l.a))
181                             || pointOnLineLeft(w, Line(u, v))) {
182                             return false;
183                         }
184                     }
185                 } else {
186                     if (pointOnLineLeft(u, l)) {
187                         if (pointOnLineLeft(w, Line(l.b, l.a))
188                             || pointOnLineLeft(w, Line(u, v))) {
189                             return false;
190                         }
191                     } else {
192                         if (pointOnLineLeft(w, l)
193                             || pointOnLineLeft(w, Line(u, v))) {
194                             return false;
195                         }
196                     }
197                 }
198             }
199         }
200     }
201     return true;
202 }
```

```

203 template<class T>
204 vector<Point<T>> hp(vector<Line<T>> lines) {
205     sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
206         auto d1 = l1.b - l1.a;
207         auto d2 = l2.b - l2.a;
208         if (sgn(d1) != sgn(d2)) {
209             return sgn(d1) == 1;
210         }
211         return cross(d1, d2) > 0;
212     });
213     deque<Line<T>> ls;
214     deque<Point<T>> ps;
215     for (auto l : lines) {
216         if (ls.empty()) {
217             ls.push_back(l);
218             continue;
219         }
220         while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
221             ps.pop_back();
222             ls.pop_back();
223         }
224         while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
225             ps.pop_front();
226             ls.pop_front();
227         }
228         if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
229             if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
230                 if (!pointOnLineLeft(ls.back().a, l)) {
231                     assert(ls.size() == 1);
232                     ls[0] = l;
233                 }
234                 continue;
235             }
236             return {};
237         }
238         ps.push_back(lineIntersection(ls.back(), l));
239         ls.push_back(l);
240     }
241     while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
242         ps.pop_back();
243         ls.pop_back();
244     }
245     if (ls.size() <= 2) {
246         return {};
247     }
248     ps.push_back(lineIntersection(ls[0], ls.back()));
249     return vector(ps.begin(), ps.end());
250 }
251 using P = Point<long double>;
252 const long double eps = 1E-6L;
253 int main() {
254     ios::sync_with_stdio(false);
255     cin.tie(nullptr);
256     int n;
257     cin >> n;
258     vector<P> p(n);
259     for (int i = 0; i < n; i++) {
260         int x, y;
261         cin >> x >> y;
262         p[i].x = x;
263         p[i].y = y;
264     }
265     int xa, ya;
266     cin >> xa >> ya;

```

```

267     int xv, yv, zv;
268     cin >> xv >> yv >> zv;
269     int fdown;
270     cin >> fdown;
271     int xu, yu, zu;
272     cin >> xu >> yu >> zu;
273     P A(xa, ya);
274     P V(1.L * xv / zv, 1.L * yv / zv);
275     P U(1.L * xu / -zu, 1.L * yu / -zu);
276     cout << fixed << setprecision(10);
277     long double t1 = -1, t2 = -1;
278     auto update = [&](long double x, long double y) {
279         if (t1 == -1 || x < t1 - eps) {
280             t1 = x;
281             t2 = y;
282         } else if (x < t1 + eps && y < t2) {
283             t2 = y;
284         }
285     };
286     for (int i = 0; i < n; i++) {
287         P B = p[i];
288         P C = p[(i + 1) % n];
289         if (cross(V, U) == 0) {
290             for (auto d : {V, V + U}) {
291                 if (d == P(0, 0)) {
292                     continue;
293                 }
294                 vector<P> cand{B, C};
295                 if (cross(d, B - C) != 0) {
296                     cand.push_back(lineIntersection(Line(A, A + d), Line(B, C)));
297                 }
298                 for (auto p : cand) {
299                     if (!pointOnSegment(p, Line(B, C))) {
300                         continue;
301                     }
302                     if (abs(cross(d, p - A)) > eps) {
303                         continue;
304                     }
305                     if (dot(d, p - A) < -eps) {
306                         continue;
307                     }
308                     auto t1 = dot(p - A, d) / square(d) / zv;
309                     long double t2;
310                     if (d != V + U) {
311                         t2 = 1.0L * t1 * zv / -fdown;
312                     } else {
313                         t2 = 0;
314                     }
315                     update(t1, t2);
316                 }
317             }
318         } else {
319             P l = V, r = V + U;
320             if (cross(l, r) < 0) {
321                 swap(l, r);
322             }
323             vector<P> cand{B, C};
324             if (abs(cross(l, B - C)) > eps) {
325                 cand.push_back(lineIntersection(Line(A, A + l), Line(B, C)));
326             }
327             if (abs(cross(r, B - C)) > eps) {
328                 cand.push_back(lineIntersection(Line(A, A + r), Line(B, C)));
329             }
330             for (auto p : cand) {

```

```

331         if (!pointOnSegment(p, Line(B, C))) {
332             continue;
333         }
334         if (cross(l, p - A) < -eps || cross(r, p - A) > eps) {
335             continue;
336         }
337         auto t1 = dot(p - A, rotate(l - r)) / dot(l, rotate(l - r)) / zv;
338         long double t2;
339         if (l == V) {
340             t2 = length(A + t1 * r * zv - p) / length(U) / -fdown;
341         } else {
342             t2 = length(A + t1 * l * zv - p) / length(U) / -fdown;
343         }
344         update(t1, t2);
345     }
346 }
347 cout << t1 << " " << t2 << "\n";
349 return 0;
350 }
```

## 885: Palisection

- Time limit: 2 seconds
- Memory limit: 128 megabytes
- Input file: standard input
- Output file: standard output

In an English class Nick had nothing to do at all, and remembered about wonderful strings called palindromes. We should remind you that a string is called a palindrome if it can be read the same way both from left to right and from right to left. Here are examples of such strings: eye, pop, level, aba, deed, racecar, rotor, madam.

Nick started to look carefully for all palindromes in the text that they were reading in the class. For each occurrence of each palindrome in the text he wrote a pair - the position of the beginning and the position of the ending of this occurrence in the text. Nick called each occurrence of each palindrome he found in the text subpalindrome. When he found all the subpalindromes, he decided to find out how many different pairs among these subpalindromes cross. Two subpalindromes cross if they cover common positions in the text. No palindrome can cross itself.

Let's look at the actions, performed by Nick, by the example of text babb. At first he wrote out all subpalindromes:

Then Nick counted the amount of different pairs among these subpalindromes that cross. These pairs were six:

Since it's very exhausting to perform all the described actions manually, Nick asked you to help him and write a program that can find out the amount of different subpalindrome pairs that cross. Two

subpalindrome pairs are regarded as different if one of the pairs contains a subpalindrome that the other does not.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 51123987;
33 using Z = MInt<P>;
34 vector<int> manacher(string s) {
35     string t = "#";
36     for (auto c : s) {
37         t += c;
38         t += '#';
39     }
40     int n = t.size();
41     vector<int> r(n);
42     for (int i = 0, j = 0; i < n; i++) {
43         if (2 * j - i >= 0 && j + r[j] > i) {
44             r[i] = min(r[2 * j - i], j + r[j] - i);
45         }
46         while (i - r[i] >= 0 && i + r[i] < n && t[i - r[i]] == t[i + r[i]]) {
47             r[i] += 1;
48         }
49         if (i + r[i] > j + r[j]) {
50             j = i;
51         }
52     }
53     return r;
54 }
```

```

55 int main() {
56     ios::sync_with_stdio(false);
57     cin.tie(nullptr);
58     int n;
59     cin >> n;
60     string s;
61     cin >> s;
62     auto r = manacher(s);
63     vector<i64> L(n + 1), R(n + 1);
64     Z ans = 0;
65     for (int i = 0; i <= 2 * n; i++) {
66         L[(i - r[i] + 1) / 2] += 1;
67         L[(i + 1) / 2] -= 1;
68         R[i / 2] += 1;
69         R[(i + r[i] - 1) / 2] -= 1;
70     }
71     for (int i = 1; i <= n; i++) {
72         L[i] += L[i - 1];
73         R[i] += R[i - 1];
74     }
75     i64 sum = 0;
76     for (int i = 0; i < n; i++) {
77         sum += L[i];
78         ans += Z(sum) * R[i];
79     }
80     if (sum % 2 == 0) {
81         ans -= Z(sum / 2) * (sum + 1);
82     } else {
83         ans -= Z((sum + 1) / 2) * sum;
84     }
85     cout << ans << "\n";
86     return 0;
87 }

```

## 886: Communication Towers

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

There are  $n$  communication towers, numbered from 1 to  $n$ , and  $m$  bidirectional wires between them. Each tower has a certain set of frequencies that it accepts, the  $i$ -th of them accepts frequencies from  $l_i$  to  $r_i$ .

Let's say that a tower  $b$  is accessible from a tower  $a$ , if there exists a frequency  $x$  and a sequence of towers  $a = v_1, v_2, \dots, v_k = b$ , where consecutive towers in the sequence are directly connected by a wire, and each of them accepts frequency  $x$ . Note that accessibility is not transitive, i. e if  $b$  is accessible from  $a$  and  $c$  is accessible from  $b$ , then  $c$  may not be accessible from  $a$ .

Your task is to determine the towers that are accessible from the 1-st tower.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int N = 1 << 18;
5 vector<pair<int, int>> e[2 * N];
6 void add(int p, int l, int r, int x, int y, int u, int v) {
7     if (l >= y || r <= x) {
8         return;
9     }
10    if (l >= x && r <= y) {
11        e[p].emplace_back(u, v);
12        return;
13    }
14    int m = (l + r) / 2;
15    add(2 * p, l, m, x, y, u, v);
16    add(2 * p + 1, m, r, x, y, u, v);
17}
18 constexpr int L = 2E7;
19 int f[N], lc[N], rc[N], ok[N];
20 int find(int x) {
21     while (f[x] >= 0) {
22         x = f[x];
23     }
24     return x;
25}
26 vector<array<int, 5>> h;
27 void solve(int p, int l, int r) {
28     int s = h.size();
29     for (auto [u, v] : e[p]) {
30         u = find(u);
31         v = find(v);
32         if (u != v) {
33             if (f[u] > f[v]) {
34                 swap(u, v);
35             }
36             h.push_back({u, v, f[u], f[v], ok[u]} );
37             f[u] += f[v];
38             f[v] = u;
39             ok[u] = find(0) == u;
40         }
41     }
42     if (r - l > 1) {
43         int m = (l + r) / 2;
44         solve(2 * p, l, m);
45         solve(2 * p + 1, m, r);
46     }
47     while (h.size() > s) {
48         auto [u, v, fu, fv, oku] = h.back();
49         h.pop_back();
50         f[u] = fu;
51         f[v] = fv;
52         if (ok[u]) {
53             ok[u] = ok[v] = 1;
54         } else {
55             ok[u] = oku;
56         }
57     }
58 }
59 int main() {
60     ios::sync_with_stdio(false);

```

```

61     cin.tie(nullptr);
62     int n, m;
63     cin >> n >> m;
64     vector<int> l(n), r(n);
65     for (int i = 0; i < n; i++) {
66         cin >> l[i] >> r[i];
67     }
68     for (int i = 0; i < m; i++) {
69         int u, v;
70         cin >> u >> v;
71         u--, v--;
72         int L = max(l[u], l[v]);
73         int R = min(r[u], r[v]);
74         if (L <= R) {
75             add(1, 0, N, L, R + 1, u, v);
76         }
77     }
78     fill(f, f + n, -1);
79     ok[0] = 1;
80     solve(1, 0, N);
81     for (int i = 0; i < n; i++) {
82         if (ok[i]) {
83             cout << i + 1 << " ";
84         }
85     }
86     cout << "\n";
87     return 0;
88 }
```

## 887: Approximate Diameter

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Jack has a graph of  $n$  vertices and  $m$  edges. All edges are bidirectional and of unit length. The graph is connected, i. e. there exists a path between any two of its vertices. There can be more than one edge connecting the same pair of vertices. The graph can contain self-loops, i. e. edges connecting a node to itself.

The distance between vertices  $u$  and  $v$  is denoted as  $\rho(u, v)$  and equals the minimum possible number of edges on a path between  $u$  and  $v$ . The diameter of graph  $G$  is defined as the maximum possible distance between some pair of its vertices. We denote it as  $d(G)$ . In other words,

$$d(G) = \max_{1 \leq u, v \leq n} \rho(u, v).$$

Jack plans to consecutively apply  $q$  updates to his graph. Each update adds exactly one edge to the graph. Denote as  $G_i$  the graph after exactly  $i$  updates are made. Jack wants to calculate  $q + 1$  values  $d(G_0), d(G_1), d(G_2), \dots, d(G_q)$ .

However, Jack suspects that finding the exact diameters of  $q + 1$  graphs might be a difficult task, so he is fine with approximate answers that differ from the correct answers no more than twice. Formally, Jack wants to find a sequence of positive integers  $a_0, a_1, a_2, \dots, a_q$  such that

$$\left\lceil \frac{d(G_i)}{2} \right\rceil \leq a_i \leq 2 \cdot d(G_i)$$

for each  $i$ .

### Hacks

You cannot make hacks in this problem.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, m, q;
8     cin >> n >> m >> q;
9     vector<vector<pair<int, int>>> adj(n);
10    for (int i = 0; i < m; i++) {
11        int u, v;
12        cin >> u >> v;
13        u--, v--;
14        adj[u].emplace_back(v, 0);
15        adj[v].emplace_back(u, 0);
16    }
17    for (int i = 0; i < q; i++) {
18        int u, v;
19        cin >> u >> v;
20        u--, v--;
21        adj[u].emplace_back(v, i + 1);
22        adj[v].emplace_back(u, i + 1);
23    }
24    auto get = [&](int t) {
25        queue<int> q;
26        vector<int> dis(n, -1);
27        q.push(0);
28        dis[0] = 0;
29        while (!q.empty()) {
30            int x = q.front();
31            q.pop();
32            for (auto [y, w] : adj[x]) {
33                if (w <= t && dis[y] == -1) {
34                    dis[y] = dis[x] + 1;
35                    q.push(y);
36                }
37            }
38        }
39        return *max_element(dis.begin(), dis.end());
40    };
41    vector<int> ans(q + 1);
42    for (int i = 0; i < q + 1; ) {
43        int val = get(i);

```

```

44     int lo = i + 1, hi = q + 1;
45     while (lo < hi) {
46         int x = (lo + hi) / 2;
47         if (2 * get(x) < val) {
48             hi = x;
49         } else {
50             lo = x + 1;
51         }
52     }
53     while (i < lo) {
54         ans[i] = val;
55         i += 1;
56     }
57 }
58 for (int i = 0; i <= q; i++) {
59     cout << ans[i] << " \n"[i == q];
60 }
61 return 0;
62 }
```

### 888: Strange Triples

- Time limit: 10 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Let's call a triple of positive integers  $(a, b, n)$  strange if the equality  $\frac{an}{nb} = \frac{a}{b}$  holds, where  $an$  is the concatenation of  $a$  and  $n$  and  $nb$  is the concatenation of  $n$  and  $b$ . For the purpose of concatenation, the integers are considered without leading zeroes.

For example, if  $a = 1, b = 5$  and  $n = 9$ , then the triple is strange, because  $\frac{19}{95} = \frac{1}{5}$ . But  $a = 7, b = 3$  and  $n = 11$  is not strange, because  $\frac{711}{113} \neq \frac{7}{3}$ .

You are given three integers  $A, B$  and  $N$ . Calculate the number of strange triples  $(a, b, n)$ , such that  $1 \leq a < A, 1 \leq b < B$  and  $1 \leq n < N$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int bits(int n) {
5     int k = 1;
6     while (k <= n) {
7         k *= 10;
8     }
9     return k;
10 }
```

```

11 int main() {
12     ios::sync_with_stdio(false);
13     cin.tie(nullptr);
14     int A, B, N;
15     cin >> A >> B >> N;
16     int ans = 0;
17     for (int ln = 1, t = 9; ln <= 9; ln++, t = 10 * t + 9) {
18         map<int, int> et;
19         int x = t;
20         for (int i = 2; i * i <= x; i++) {
21             while (x % i == 0) {
22                 et[i] += 1;
23                 x /= i;
24             }
25         }
26         if (x > 1) {
27             et[x] += 1;
28         }
29         for (int b = 1; b < B; b++) {
30             auto e = et;
31             int x = b;
32             for (int i = 2; i * i <= x; i++) {
33                 while (x % i == 0) {
34                     e[i] += 2;
35                     x /= i;
36                 }
37             }
38             if (x > 1) {
39                 e[x] += 2;
40             }
41             x = gcd(b, bits(b));
42             int e2 = e[2] / 2;
43             int e5 = e[5] / 2;
44             for (int i = 2; i * i <= x; i++) {
45                 while (x % i == 0) {
46                     e[i] += 1;
47                     x /= i;
48                 }
49             }
50             if (x > 1) {
51                 e[x] += 1;
52             }
53             i64 init = 1;
54             e2 = e[2] - 2 * e2;
55             e5 = e[5] - 2 * e5;
56             init *= 1LL << e2;
57             e[2] -= e2;
58             e[5] -= e5;
59             for (int j = 0; j < e5; j++) {
60                 init *= 5;
61             }
62             int bitsb = bits(b);
63             int ctz = __builtin_ctz(bitsb);
64             if (e2 < ctz) {
65                 e.erase(2);
66             }
67             if (e5 < ctz) {
68                 e.erase(5);
69             }
70             vector<pair<int, int>> p(e.begin(), e.end());
71             reverse(p.begin(), p.end());
72             i64 lim = 1LL * A * bitsb;
73             auto dfs = [&](auto self, int i, i64 v) {
74                 if (i == p.size()) {

```

```

75             i64 a = v + b;
76             if (a % bitsb != 0) {
77                 return;
78             }
79             a /= bitsb;
80             if (a >= A) {
81                 return;
82             }
83             i64 u = a * b;
84             if (v < u) {
85                 return;
86             }
87             i64 g = gcd(u, v);
88             u /= g;
89             v /= g;
90             if (t % v == 0) {
91                 i64 n = t / v * u;
92                 // cerr << a << " " << b << " " << u << " " << v << " " << n << "\n";
93                 if (n < N) {
94                     // cerr << a << " " << b << " " << n << "\n";
95                     ans += 1;
96                 }
97             }
98             return;
99         }
100        for (int j = 0; j <= p[i].second; j++) {
101            if (j > 0) {
102                v *= p[i].first;
103            }
104            if (v > lim) {
105                break;
106            }
107            self(self, i + 1, v);
108        }
109    };
110    dfs(dfs, 0, init);
111}
112}
113// for (int a = 1; a < A; a++) {
114//     for (int b = 1; b < B; b++) {
115//         i64 x = 1LL * a * bits(b) - b;
116//         i64 y = 1LL * a * b;
117//         if (x < y) {
118//             continue;
119//         }
120//         for (int l = 1, t = 9; l <= 9; l++, t = 10 * t + 9) {
121//             i64 v = y * t;
122//             if (v % x == 0) {
123//                 i64 n = v / x;
124//                 if (n < N) {
125//                     ans += 1;
126//                 }
127//             }
128//         }
129//     }
130// }
131cout << ans << "\n";
132return 0;
133}

```

## 889: Removal Sequences

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a simple undirected graph, consisting of  $n$  vertices and  $m$  edges. The vertices are numbered from 1 to  $n$ . The  $i$ -th vertex has a value  $a_i$  written on it.

You will be removing vertices from that graph. You are allowed to remove vertex  $i$  only if its degree is equal to  $a_i$ . When a vertex is removed, all edges incident to it are also removed, thus, decreasing the degree of adjacent non-removed vertices.

A valid sequence of removals is a permutation  $p_1, p_2, \dots, p_n$  ( $1 \leq p_i \leq n$ ) such that the  $i$ -th vertex to be removed is  $p_i$ , and every removal is allowed.

A pair  $(x, y)$  of vertices is nice if there exist two valid sequences of removals such that  $x$  is removed before  $y$  in one of them and  $y$  is removed before  $x$  in the other one.

Count the number of nice pairs  $(x, y)$  such that  $x < y$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n, m;
6     cin >> n >> m;
7     vector<int> a(n);
8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10    }
11    vector<vector<int>> adj(n);
12    for (int i = 0; i < m; i++) {
13        int u, v;
14        cin >> u >> v;
15        u--;
16        v--;
17        adj[u].push_back(v);
18        adj[v].push_back(u);
19    }
20    for (int i = 0; i < n; i++) {
21        a[i] = adj[i].size() - a[i];
22    }
23    vector<int> q;
24    for (int i = 0; i < n; i++) {
25        if (a[i] == 0) {
26            q.push_back(i);
27        }
28    }

```

```

28     for (int i = 0; i < n; i++) {
29         int x = q[i];
30         for (auto y : adj[x]) {
31             if (a[y] > 0 && --a[y] == 0) {
32                 q.push_back(y);
33             }
34         }
35     }
36     vector<int> pos(n);
37     for (int i = 0; i < n; i++) {
38         pos[q[i]] = i;
39     }
40     for (int i = 0; i < n; i++) {
41         vector<int> nadj;
42         for (auto y : adj[i]) {
43             if (pos[y] > pos[i]) {
44                 nadj.push_back(y);
45             }
46         }
47         swap(nadj, adj[i]);
48     }
49     i64 ans = 1LL * n * (n + 1) / 2;
50     vector<bitset<8192>> f(n);
51     for (int i = 0; i < n; i += 8192) {
52         f.assign(n, {});
53         for (auto x : q) {
54             if (i <= x && x < i + 8192) {
55                 f[x][x - i] = 1;
56             }
57             for (auto y : adj[x]) {
58                 f[y] |= f[x];
59             }
60         }
61         for (int i = 0; i < n; i++) {
62             ans -= f[i].count();
63         }
64     }
65     cout << ans << "\n";
66 }
67 int main() {
68     ios::sync_with_stdio(false);
69     cin.tie(nullptr);
70     int t;
71     cin >> t;
72     while (t--) {
73         solve();
74     }
75     return 0;
76 }
```

**890: Hero to Zero**

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

There are no heroes in this problem. I guess we should have named it “To Zero”.

You are given two arrays  $a$  and  $b$ , each of these arrays contains  $n$  non-negative integers.

Let  $c$  be a matrix of size  $n \times n$  such that  $c_{i,j} = |a_i - b_j|$  for every  $i \in [1, n]$  and every  $j \in [1, n]$ .

Your goal is to transform the matrix  $c$  so that it becomes the zero matrix, i. e. a matrix where every element is exactly 0. In order to do so, you may perform the following operations any number of times, in any order:

choose an integer  $i$ , then decrease  $c_{i,j}$  by 1 for every  $j \in [1, n]$  (i. e. decrease all elements in the  $i$ -th row by 1). In order to perform this operation, you pay 1 coin;

choose an integer  $j$ , then decrease  $c_{i,j}$  by 1 for every  $i \in [1, n]$  (i. e. decrease all elements in the  $j$ -th column by 1). In order to perform this operation, you pay 1 coin;

choose two integers  $i$  and  $j$ , then decrease  $c_{i,j}$  by 1. In order to perform this operation, you pay 1 coin;

choose an integer  $i$ , then increase  $c_{i,j}$  by 1 for every  $j \in [1, n]$  (i. e. increase all elements in the  $i$ -th row by 1). When you perform this operation, you receive 1 coin;

choose an integer  $j$ , then increase  $c_{i,j}$  by 1 for every  $i \in [1, n]$  (i. e. increase all elements in the  $j$ -th column by 1). When you perform this operation, you receive 1 coin.

You have to calculate the minimum number of coins required to transform the matrix  $c$  into the zero matrix. Note that all elements of  $c$  should be equal to 0 simultaneously after the operations.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n), b(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    for (int i = 0; i < n; i++) {
14        cin >> b[i];
15    }
16    sort(a.begin(), a.end());
17    sort(b.begin(), b.end());
18    i64 match = 0, sum = 0;
19    for (int i = 0; i < n; i++) {
20        match += abs(a[i] - b[i]);
21    }
22    vector<i64> sb(n + 1);
23    for (int i = 0; i < n; i++) {
24        sb[i + 1] = sb[i] + b[i];
25    }

```

```

26     for (int i = 0, j = 0; i < n; i++) {
27         while (j < n && b[j] < a[i]) {
28             j++;
29         }
30         sum += 1LL * a[i] * (j - n + j) + sb[n] - sb[j] - sb[j];
31     }
32     sum -= match * (n - 1);
33     cout << sum << "\n";
34     return 0;
35 }
```

## 891: Infinite Chess

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

The black king lives on a chess board with an infinite number of columns (files) and 8 rows (ranks). The columns are numbered with all integer numbers (including negative). The rows are numbered from 1 to 8.

Initially, the black king is located on the starting square  $(x_s, y_s)$ , and he needs to reach some target square  $(x_t, y_t)$ . Unfortunately, there are also white pieces on the board, and they threaten the black king. After negotiations, the white pieces agreed to let the black king pass to the target square on the following conditions:

each turn, the black king makes a move according to the movement rules;

the black king cannot move to a square occupied by a white piece;

the black king cannot move to a square which is under attack by any white piece. A square is under attack if a white piece can reach it in one move according to the movement rules;

the white pieces never move.

Help the black king find the minimum number of moves needed to reach the target square while not violating the conditions. The black king cannot leave the board at any time.

The black king moves according to the movement rules below. Even though the white pieces never move, squares which they can reach in one move are considered to be under attack, so the black king cannot move into those squares.

Below are the movement rules. Note that the pieces (except for the knight) cannot jump over other pieces.

a king moves exactly one square horizontally, vertically, or diagonally.

a rook moves any number of vacant squares horizontally or vertically.

a bishop moves any number of vacant squares diagonally.

a queen moves any number of vacant squares horizontally, vertically, or diagonally.

a knight moves to one of the nearest squares not on the same rank, file, or diagonal (this can be thought of as moving two squares horizontally then one square vertically, or moving one square horizontally then two squares vertically - i. e. in an “L” pattern). Knights are not blocked by other pieces, they can simply jump over them.

There are no pawns on the board.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int xs, ys;
8     cin >> xs >> ys;
9     ys--;
10    int xt, yt;
11    cin >> xt >> yt;
12    yt--;
13    int n;
14    cin >> n;
15    vector<char> type(n);
16    vector<int> x(n), y(n);
17    set<pair<int, int>> piece;
18    vector<int> vx{xs, xt};
19    for (int i = 0; i < n; i++) {
20        cin >> type[i] >> x[i] >> y[i];
21        y[i]--;
22        piece.emplace(x[i], y[i]);
23        vx.push_back(x[i]);
24    }
25    auto valid = [&](int x, int y) {
26        return 0 <= y && y < 8;
27    };
28    set<pair<int, int>> ban;
29    for (int i = 0; i < n; i++) {
30        ban.emplace(x[i], y[i]);
31        if (type[i] == 'K') {
32            for (auto dx = -1; dx <= 1; dx++) {
33                for (auto dy = -1; dy <= 1; dy++) {
34                    if (valid(x[i] + dx, y[i] + dy)) {
35                        ban.emplace(x[i] + dx, y[i] + dy);
36                    }
37                }
38            }
39        } else if (type[i] == 'R') {
40            for (auto [dx, dy] : {pair(-1, 0), {1, 0}, {0, -1}, {0, 1}}) {
41                for (int d = 1; d <= 20; d++) {
42                    int nx = x[i] + dx * d;
43                    int ny = y[i] + dy * d;

```

```

44             if (piece.contains({nx, ny}) || !valid(nx, ny)) break;
45             ban.emplace(nx, ny);
46         }
47     }
48 } else if (type[i] == 'B') {
49     for (auto dx : {-1, 1}) {
50         for (auto dy : {-1, 1}) {
51             for (int d = 1; d <= 20; d++) {
52                 int nx = x[i] + dx * d;
53                 int ny = y[i] + dy * d;
54                 if (piece.contains({nx, ny}) || !valid(nx, ny)) break;
55                 ban.emplace(nx, ny);
56             }
57         }
58     }
59 } else if (type[i] == 'Q') {
60     for (auto dx : {-1, 0, 1}) {
61         for (auto dy : {-1, 0, 1}) {
62             for (int d = 1; d <= 20; d++) {
63                 int nx = x[i] + dx * d;
64                 int ny = y[i] + dy * d;
65                 if (piece.contains({nx, ny}) || !valid(nx, ny)) break;
66                 ban.emplace(nx, ny);
67             }
68         }
69     }
70 } else {
71     for (auto [dx, dy] : {pair(-2, -1), {-2, 1}, {-1, -2}, {-1, 2}, {1, -2}, {1,
72         2}, {2, -1}, {2, 1}}) {
73         if (valid(x[i] + dx, y[i] + dy)) {
74             ban.emplace(x[i] + dx, y[i] + dy);
75         }
76     }
77 }
78 map<pair<int, int>, int> dis;
79 priority_queue<tuple<int, int, int>> h;
80 h.emplace(0, xs, ys);
81 vx.push_back(-20);
82 vx.push_back(1E8 + 20);
83 sort(vx.begin(), vx.end());
84 vector<pair<int, int>> skip;
85 map<pair<int, int>, vector<tuple<int, int, int>>> adj;
86 for (int i = 0; i < vx.size() - 1; i++) {
87     array<int, 8> xl, xr;
88     array<char, 8> typel, typer;
89     xl.fill(-1);
90     xr.fill(1E9);
91     typel.fill('K');
92     typer.fill('K');
93     for (int j = 0; j < n; j++) {
94         if (x[j] <= vx[i]) {
95             if (x[j] > xl[y[j]]) {
96                 xl[y[j]] = x[j];
97                 typel[y[j]] = type[j];
98             }
99         } else {
100             if (x[j] < xr[y[j]]) {
101                 xr[y[j]] = x[j];
102                 typer[y[j]] = type[j];
103             }
104         }
105     }
106     if (vx[i + 1] - vx[i] > 30) {

```

```

107         skip.emplace_back(vx[i] + 10, vx[i + 1] - 10);
108         int last = -1;
109         for (int y = 0; y <= 8; y++) {
110             if (y == 8 || typel[y] == 'Q' || typel[y] == 'R' || typer[y] == 'Q' ||
111                 typer[y] == 'R') {
112                 for (int y1 = last + 1; y1 < y; y1++) {
113                     for (int y2 = last + 1; y2 < y; y2++) {
114                         int x1 = vx[i] + 9, x2 = vx[i + 1] - 9;
115                         adj[{x1, y1}].emplace_back(x2, y2, x2 - x1);
116                         adj[{x2, y2}].emplace_back(x1, y1, x2 - x1);
117                     }
118                 }
119             }
120         }
121         for (int y = 0; y < 8; y++) {
122             if (typel[y] == 'R' || typel[y] == 'Q' || typer[y] == 'Q' || typer[y] == 'R') {
123                 for (int x = vx[i]; x <= vx[i + 1]; x++) {
124                     if (x >= vx[i] + 10) {
125                         x = max(x, vx[i + 1] - 9);
126                     }
127                     ban.emplace(x, y);
128                 }
129             }
130         }
131     } else {
132         for (int y = 0; y < 8; y++) {
133             if (typel[y] == 'Q' || typel[y] == 'R' || typer[y] == 'Q' || typer[y] == 'R') {
134                 for (int x = vx[i]; x <= vx[i + 1]; x++) {
135                     ban.emplace(x, y);
136                 }
137             }
138         }
139     }
140 }
141 while (!h.empty()) {
142     auto [d, x, y] = h.top();
143     h.pop();
144     d = -d;
145     if (dis.contains({x, y})) continue;
146     dis[{x, y}] = d;
147     for (auto dx = -1; dx <= 1; dx++) {
148         for (auto dy = -1; dy <= 1; dy++) {
149             int nx = x + dx;
150             int ny = y + dy;
151             if (!valid(nx, ny) || ban.contains({nx, ny})) continue;
152             if (nx < vx[0] || nx > vx.back()) continue;
153             int j = lower_bound(skip.begin(), skip.end(), pair(nx + 1, -1)) - skip.
154                 begin() - 1;
155             if (j >= 0 && skip[j].second >= nx) {
156                 continue;
157             }
158             h.emplace(-d - 1, nx, ny);
159         }
160     }
161     if (adj.contains({x, y})) {
162         for (auto [nx, ny, w] : adj[{x, y}]) {
163             h.emplace(-d - w, nx, ny);
164         }
165     }
166     if (!dis.contains({xt, yt})) {

```

```

167         cout << -1 << "\n";
168     } else {
169         cout << dis[{xt, yt}] << "\n";
170     }
171     return 0;
172 }
```

## 892: The Game of the Century

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The time has finally come, MKnez and Baltic are to host The Game of the Century. For that purpose, they built a village to lodge its participants.

The village has the shape of an equilateral triangle delimited by three roads of length  $n$ . It is cut into  $n^2$  smaller equilateral triangles, of side length 1, by  $3n - 3$  additional roads which run parallel to the sides. See the figure for  $n = 3$ . Each of the  $3n$  roads is made of multiple (possibly 1) road segments of length 1 which connect adjacent intersections.

The direction has already been chosen for each of the  $3n$  roads (so, for each road, the same direction is assigned to all its road segments). Traffic can only go in the specified directions (i. e. the roads are monodirectional).

You are tasked with making adjustments to the traffic plan so that from each intersection it is possible to reach every other intersection. Specifically, you can invert the traffic direction of any number of road segments of length 1. What is the minimal number of road segments for which you need to invert the traffic direction?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<vector<int>> a(3, vector<int>(n));
8     for (int i = 0; i < 3; i++) {
9         for (int j = 0; j < n; j++) {
10            char c;
11            cin >> c;
12            a[i][j] = c - '0';
13        }
14    }
15    if (a[0][n - 1] == a[1][n - 1] && a[1][n - 1] == a[2][n - 1]) {
```

```

16         cout << 0 << "\n";
17     }
18 }
19 if (a[0][n - 1] + a[1][n - 1] + a[2][n - 1] == 1) {
20     for (auto &b : a) {
21         for (auto &c : b) c ^= 1;
22     }
23 }
24 while (a[2][n - 1] != 0) {
25     rotate(a.begin(), a.begin() + 1, a.end());
26 }
27 int mx[3] {};
28 for (int j = 0; j < 3; j++) {
29     for (int i = 1; i <= n; i++) {
30         if (a[j][i - 1] != a[j][n - 1]) mx[j] = i;
31     }
32 }
33 int ans = min(n, min(n - mx[0], n - mx[2]) + min(n - mx[1], n - mx[2]));
34 cout << ans << "\n";
35 }
36 int main() {
37     ios::sync_with_stdio(false);
38     cin.tie(nullptr);
39     int t;
40     cin >> t;
41     while (t--) {
42         solve();
43     }
44     return 0;
45 }

```

**893: MCF**

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given a graph consisting of  $n$  vertices and  $m$  directed arcs. The  $i$ -th arc goes from the vertex  $x_i$  to the vertex  $y_i$ , has capacity  $c_i$  and weight  $w_i$ . No arc goes into the vertex 1, and no arc goes from the vertex  $n$ . There are no cycles of negative weight in the graph (it is impossible to travel from any vertex to itself in such a way that the total weight of all arcs you go through is negative).

You have to assign each arc a flow (an integer between 0 and its capacity, inclusive). For every vertex except 1 and  $n$ , the total flow on the arcs going to this vertex must be equal to the total flow on the arcs going from that vertex. Let the flow on the  $i$ -th arc be  $f_i$ , then the cost of the flow is equal to  $\sum_{i=1}^m f_i w_i$ .

You have to find a flow which minimizes the cost.

Sounds classical, right? Well, we have some additional constraints on the flow on every edge:

if  $c_i$  is even,  $f_i$  must be even;

if  $c_i$  is odd,  $f_i$  must be odd.

Can you solve this problem?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct MCFGraph;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, m;
9     cin >> n >> m;
10    MCFGraph g(n + 2);
11    vector<int> out(n), par(m);
12    for (int i = 0; i < m; i++) {
13        int x, y, c, w;
14        cin >> x >> y >> c >> w;
15        par[i] = c % 2;
16        x--, y--;
17        out[x] += c % 2;
18        out[y] -= c % 2;
19        g.addEdge(x, y, c / 2, w);
20        if (w < 0) {
21            out[x] += c / 2 * 2;
22            out[y] -= c / 2 * 2;
23        }
24    }
25    for (int i = 1; i < n - 1; i++) {
26        if (out[i] % 2) {
27            cout << "Impossible\n";
28            return 0;
29        }
30    }
31    g.addEdge(n - 1, 0, 1E9, 0);
32    if (out[0] % 2) {
33        out[0]++;
34        out[n - 1]--;
35    }
36    int tot = 0;
37    for (int i = 0; i < n; i++) {
38        if (out[i] > 0) {
39            tot += out[i] / 2;
40            g.addEdge(i, n + 1, out[i] / 2, 0);
41        } else g.addEdge(n, i, -out[i] / 2, 0);
42    }
43    if (g.flow(n, n + 1).first < tot) {
44        cout << "Impossible\n";
45        return 0;
46    }
47    cout << "Possible\n";
48    for (int i = 0; i < m; i++) {
49        cout << g.e[2 * i + 1].c * 2 + par[i] << " \n"[i == m - 1];
50    }
51    return 0;
52 }
```

### 894: Doremy's Perfect DS Class (Medium Version)

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The only difference between this problem and the other two versions is the maximum number of queries. In this version, you are allowed to ask at most 25 queries. You can make hacks only if all versions of the problem are solved.

This is an interactive problem.

“Everybody! Doremy’s Perfect Data Structure Class is about to start! Come and do your best if you want to have as much IQ as me!” In today’s Data Structure class, Doremy is teaching everyone a powerful data structure - Doremy tree! Now she gives you a quiz to prove that you are paying attention in class.

Given an array  $a$  of length  $m$ , Doremy tree supports the query  $Q(l, r, k)$ , where  $1 \leq l \leq r \leq m$  and  $1 \leq k \leq m$ , which returns the number of distinct integers in the array  $\lfloor \frac{a_l}{k} \rfloor, \lfloor \frac{a_{l+1}}{k} \rfloor, \dots, \lfloor \frac{a_r}{k} \rfloor$ .

Doremy has a secret permutation  $p$  of integers from 1 to  $n$ . You can make queries, in one query, you give 3 integers  $l, r, k$  ( $1 \leq l \leq r \leq n, 1 \leq k \leq n$ ) and receive the value of  $Q(l, r, k)$  for the array  $p$ . Can you find the index  $y$  ( $1 \leq y \leq n$ ) such that  $p_y = 1$  in at most 25 queries?

Note that the permutation  $p$  is fixed before any queries are made.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int Q(int l, int r, int k) {
5     cout << "? " << l + 1 << " " << r << " " << k << endl;
6     int res;
7     cin >> res;
8     return res;
9 }
10 int main() {
11     ios::sync_with_stdio(false);
12     cin.tie(nullptr);
13     int n;
14     cin >> n;
15     int pn = -1;
16     int s = n / 2 + 1;
17     int l = 0, r = n;
18     while (r - l > 1) {
19         int m = (l + r) / 2;
20         int ql = Q(0, m, 2);
21         int qr = Q(m, n, 2);
22         if (n % 2 == 1 || pn != -1) {
23             int lenl = m;

```

```
24         int lenr = n - m;
25         if (0 <= pn && pn < m) {
26             ql--;
27             lenl--;
28         }
29         if (m <= pn && pn < n) {
30             qr--;
31             lenr--;
32         }
33         int both = ql + qr - s;
34         assert((lenl - both) % 2 != (lenr - both) % 2);
35         // cerr << lenl - both - (ql - both) * 2 << " " << lenr - both - (qr - both) *
36         //      2 << "\n";
37         if ((lenl - both) % 2 != 0) {
38             r = m;
39         } else {
40             l = m;
41         }
42     } else {
43         int lenl = m;
44         int lenr = n - m;
45         int both = ql + qr - s;
46         int vl = lenl - both - (ql - both) * 2;
47         int vr = lenr - both - (qr - both) * 2;
48         assert(vl <= 0);
49         assert(vr <= 0);
50         assert(vl + vr == -2);
51         if (vl == -2) {
52             r = m;
53             continue;
54         }
55         if (vr == -2) {
56             l = m;
57             continue;
58         }
59         int t = 0;
60         if (m >= 2) {
61             if (Q(0, m, n) == 1) {
62                 t = 1;
63             }
64         } else {
65             if (Q(m, n, n) == 2) {
66                 t = 1;
67             }
68         }
69         if (t == 0) {
70             pn = l;
71             l = m;
72         } else {
73             pn = m;
74             r = m;
75         }
76         s--;
77     }
78     cout << "! " << l + 1 << endl;
79     return 0;
80 }
```

### 895: Doremy's Perfect DS Class (Easy Version)

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The only difference between this problem and the other two versions is the maximum number of queries. In this version, you are allowed to ask at most **30** queries. You can make hacks only if all versions of the problem are solved.

This is an interactive problem.

“Everybody! Doremy’s Perfect Data Structure Class is about to start! Come and do your best if you want to have as much IQ as me!” In today’s Data Structure class, Doremy is teaching everyone a powerful data structure - Doremy tree! Now she gives you a quiz to prove that you are paying attention in class.

Given an array  $a$  of length  $m$ , Doremy tree supports the query  $Q(l, r, k)$ , where  $1 \leq l \leq r \leq m$  and  $1 \leq k \leq m$ , which returns the number of distinct integers in the array  $\lfloor \frac{a_l}{k} \rfloor, \lfloor \frac{a_{l+1}}{k} \rfloor, \dots, \lfloor \frac{a_r}{k} \rfloor$ .

Doremy has a secret permutation  $p$  of integers from 1 to  $n$ . You can make queries, in one query, you give 3 integers  $l, r, k$  ( $1 \leq l \leq r \leq n, 1 \leq k \leq n$ ) and receive the value of  $Q(l, r, k)$  for the array  $p$ . Can you find the index  $y$  ( $1 \leq y \leq n$ ) such that  $p_y = 1$  in at most **30** queries?

Note that the permutation  $p$  is fixed before any queries are made.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int Q(int l, int r, int k) {
5     cout << "? " << l + 1 << " " << r << " " << k << endl;
6     int res;
7     cin >> res;
8     return res;
9 }
10 int main() {
11     ios::sync_with_stdio(false);
12     cin.tie(nullptr);
13     int n;
14     cin >> n;
15     int pn = -1;
16     if (n % 2 == 0) {
17         int l = 0, r = n;
18         while (r - l > 2) {
19             int m = (l + r) / 2;
20             assert(r - m >= 2);
21             if (Q(m, r, n) == 2) {
22                 l = m;
23             } else {

```

```

24             r = m;
25         }
26     }
27     pn = l;
28     if (r - l == 2) {
29         if (l > 0) {
30             if (Q(l - 1, l + 1, n) == 1) {
31                 pn = l + 1;
32             }
33         } else {
34             if (Q(l + 1, r + 1, n) == 2) {
35                 pn = l + 1;
36             }
37         }
38     }
39     int s = (n + 1) / 2;
40     int l = 0, r = n;
41     while (r - l > 1) {
42         int m = (l + r) / 2;
43         int ql = Q(0, m, 2);
44         int qr = Q(m, n, 2);
45         int lenl = m;
46         int lenr = n - m;
47         if (0 <= pn && pn < m) {
48             ql--;
49             lenl--;
50         }
51         if (m <= pn && pn < n) {
52             qr--;
53             lenr--;
54         }
55         int both = ql + qr - s;
56         assert((lenl - both) % 2 != (lenr - both) % 2);
57         if ((lenl - both) % 2 != 0) {
58             r = m;
59         } else {
60             l = m;
61         }
62     }
63     cout << "!" << l + 1 << endl;
64     return 0;
65 }
66 }
```

## 896: Decent Division

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

A binary string is a string where every character is 0 or 1. Call a binary string decent if it has an equal number of 0s and 1s.

Initially, you have an infinite binary string  $t$  whose characters are all 0s. You are given a sequence  $a$  of  $n$  updates, where  $a_i$  indicates that the character at index  $a_i$  will be flipped ( $0 \leftrightarrow 1$ ). You need to keep

and modify after each update a set  $S$  of disjoint ranges such that:

for each range  $[l, r]$ , the substring  $t_l \dots t_r$  is a decent binary string, and

for all indices  $i$  such that  $t_i = 1$ , there exists  $[l, r]$  in  $S$  such that  $l \leq i \leq r$ .

You only need to output the ranges that are added to or removed from  $S$  after each update. You can only add or remove ranges from  $S$  at most  $10^6$  times.

More formally, let  $S_i$  be the set of ranges after the  $i$ -th update, where  $S_0 = \emptyset$  (the empty set). Define  $X_i$  to be the set of ranges removed after update  $i$ , and  $Y_i$  to be the set of ranges added after update  $i$ . Then for  $1 \leq i \leq n$ ,  $S_i = (S_{i-1} \setminus X_i) \cup Y_i$ . The following should hold for all  $1 \leq i \leq n$ :

$\forall a, b \in S_i, (a \neq b) \rightarrow (a \cap b = \emptyset)$ ;

$X_i \subseteq S_{i-1}$ ;

$(S_{i-1} \setminus X_i) \cap Y_i = \emptyset$ ;

$$\sum_{i=1}^n (|X_i| + |Y_i|) \leq 10^6.$$

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int N = 4E5;
5 # struct SegmentTree;
6 int main() {
7     ios::sync_with_stdio(false);
8     cin.tie(nullptr);
9     int n;
10    cin >> n;
11    vector<int> t(N);
12    map<int, int> ranges;
13    SegmentTree seg(vector(N, -1));
14    for (int i = 0; i < n; i++) {
15        int a;
16        cin >> a;
17        a--;
18        vector<pair<int, int>> x, y;
19        if (t[a]) {
20            t[a] = 0;
21            seg.modify(a, -1);
22            auto it = prev(ranges.upper_bound(a));
23            auto [l, r] = *it;
24            X.emplace_back(l, r);
25            ranges.erase(it);
26            int j = seg.find(l, r, -2);
27            if (l < j - 2) {
28                ranges[l] = j - 2;
29                Y.emplace_back(l, j - 2);
30            }
31            if (j < r) {
32                ranges[j] = r;
33                Y.emplace_back(j, r);
34            }
35        }
36    }
37 }
```

```

34         }
35     } else {
36         t[a] = 1;
37         seg.modify(a, 1);
38         auto it = ranges.upper_bound(a);
39         if (it != ranges.begin() && a < prev(it)->second) {
40             it--;
41             auto [l, r] = *it;
42             X.emplace_back(l, r);
43             it = ranges.erase(it);
44             r += 2;
45             if (it != ranges.end() && it->first == r) {
46                 X.push_back(*it);
47                 r = it->second;
48                 ranges.erase(it);
49             }
50             Y.emplace_back(l, r);
51             ranges[l] = r;
52         } else {
53             if (a % 2 == 1) {
54                 a--;
55             }
56             int l = a, r = a + 2;
57             if (it != ranges.begin() && prev(it)->second == l) {
58                 it--;
59                 l = it->first;
60                 X.push_back(*it);
61                 it = ranges.erase(it);
62             }
63             if (it != ranges.end() && it->first == r) {
64                 X.push_back(*it);
65                 r = it->second;
66                 ranges.erase(it);
67             }
68             Y.emplace_back(l, r);
69             ranges[l] = r;
70         }
71     }
72     cout << X.size() << "\n";
73     for (auto [l, r] : X) {
74         cout << l + 1 << " " << r << "\n";
75     }
76     cout << Y.size() << "\n";
77     for (auto [l, r] : Y) {
78         cout << l + 1 << " " << r << "\n";
79     }
80 }
81 return 0;
82 }
```

**898: Location**

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given two arrays of integers  $a_1, a_2, \dots, a_n$  and  $b_1, b_2, \dots, b_n$ . You need to handle  $q$  queries of

the following two types:

1  $l \ r \ x$ : assign  $a_i := x$  for all  $l \leq i \leq r$ ;

2  $l \ r$ : find the minimum value of the following expression among all  $l \leq i \leq r$ :

$$\frac{\text{lcm}(a_i, b_i)}{\text{gcd}(a_i, b_i)}.$$

In this problem  $\text{gcd}(x, y)$  denotes the greatest common divisor of  $x$  and  $y$ , and  $\text{lcm}(x, y)$  denotes the least common multiple of  $x$  and  $y$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int M = 50000;
5 constexpr int B = 200;
6 constexpr i64 inf = 1E12;
7 int main() {
8     ios::sync_with_stdio(false);
9     cin.tie(nullptr);
10    vector<int> primes;
11    vector<bool> isprime(M + 1, true);
12    for (int i = 2; i <= M; i++) {
13        if (isprime[i]) {
14            primes.push_back(i);
15        }
16        for (auto p : primes) {
17            if (i * p > M) {
18                break;
19            }
20            isprime[i * p] = false;
21            if (i % p == 0) {
22                break;
23            }
24        }
25    }
26    int n, q;
27    cin >> n >> q;
28    vector<int> a(n), b(n);
29    for (int i = 0; i < n; i++) {
30        cin >> a[i];
31    }
32    for (int i = 0; i < n; i++) {
33        cin >> b[i];
34    }
35    const int blocks = (n - 1) / B + 1;
36    vector same(blocks, vector<i64>(M + 1, inf));
37    auto init = [&](int j) {
38        auto &a = same[j];
39        for (int i = j * B; i < min((j + 1) * B, n); i++) {
40            a[b[i]] = 1;
41        }
42        for (auto p : primes) {
43            for (int i = M / p; i; i--) {
44                a[i] = min(a[i], a[i * p] * p);
45            }
46        }
47    };
48    for (int i = 0; i < n; i++) {
49        if (i % B == 0) {
50            init(i / B);
51        }
52        a[i] = min(a[i], a[i * B]);
53    }
54}
```

```

45         }
46     }
47     for (auto p : primes) {
48         for (int i = 1; i * p <= M; i++) {
49             a[i * p] = min(a[i * p], a[i] * p);
50         }
51     }
52 };
53 for (int i = 0; i < blocks; i++) {
54     init(i);
55 }
56 auto getAnsBlock = [&](int j) {
57     i64 ans = inf;
58     for (int i = j * B; i < min((j + 1) * B, n); i++) {
59         int g = gcd(a[i], b[i]);
60         ans = min(ans, 1LL * a[i] * b[i] / g / g);
61     }
62     return ans;
63 };
64 vector<i64> ansBlock(blocks);
65 for (int i = 0; i < blocks; i++) {
66     ansBlock[i] = getAnsBlock(i);
67 }
68 vector<int> tag(blocks, -1);
69 for (int i = 0; i < q; i++) {
70     int o;
71     cin >> o;
72     if (o == 1) {
73         int l, r, x;
74         cin >> l >> r >> x;
75         l--;
76         for (int j = 0; j < blocks; j++) {
77             int L = B * j, R = min(L + B, n);
78             if (l >= R || r <= L) {
79                 continue;
80             }
81             if (l <= L && R <= r) {
82                 tag[j] = x;
83                 ansBlock[j] = same[j][x];
84             } else {
85                 for (int k = L; k < R; k++) {
86                     if (l <= k && k < r) {
87                         a[k] = x;
88                     } else if (tag[j] != -1) {
89                         a[k] = tag[j];
90                     }
91                 }
92                 tag[j] = -1;
93                 ansBlock[j] = getAnsBlock(j);
94             }
95         }
96     } else {
97         i64 ans = inf;
98         int l, r;
99         cin >> l >> r;
100        l--;
101        for (int j = 0; j < blocks; j++) {
102            int L = B * j, R = min(L + B, n);
103            if (l >= R || r <= L) {
104                continue;
105            }
106            if (l <= L && R <= r) {
107                ans = min(ans, ansBlock[j]);
108            } else {

```

```

109             for (int k = L; k < R; k++) {
110                 if (l <= k && k < r) {
111                     int ak = tag[j] == -1 ? a[k] : tag[j];
112                     int g = gcd(ak, b[k]);
113                     ans = min(ans, 1LL * ak * b[k] / g / g);
114                 }
115             }
116         }
117     }
118     cout << ans << "\n";
119 }
120
121 return 0;
122 }
```

**899: Kazaee**

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have an array  $a$  consisting of  $n$  positive integers and you have to handle  $q$  queries of the following types:

1  $i x$ : change  $a_i$  to  $x$ ,

2  $l r k$ : check if the number of occurrences of every positive integer in the subarray  $a_l, a_{l+1}, \dots, a_r$  is a multiple of  $k$  (check the example for better understanding).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 using u32 = unsigned;
5 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
6 constexpr int N = 40;
7 template <typename T>
8 # struct Fenwick;
9 int main() {
10     ios::sync_with_stdio(false);
11     cin.tie(nullptr);
12     int n, q;
13     cin >> n >> q;
14     vector<int> a(n);
15     for (int i = 0; i < n; i++) {
16         cin >> a[i];
17     }
18     auto v = a;
19     vector<int> o(q), l(q), r(q), k(q);
20     for (int i = 0; i < q; i++) {
21         cin >> o[i] >> l[i] >> r[i];
```

```

22     l[i]--;
23     if (o[i] == 2) {
24         cin >> k[i];
25     }
26     v.push_back(r[i]);
27 }
28 sort(v.begin(), v.end());
29 const int M = v.size();
30 for (int i = 0; i < n; i++) {
31     a[i] = lower_bound(v.begin(), v.end(), a[i]) - v.begin();
32 }
33 for (int i = 0; i < q; i++) {
34     if (o[i] == 1) {
35         r[i] = lower_bound(v.begin(), v.end(), r[i]) - v.begin();
36     }
37 }
38 vector<int> ans(q, 1);
39 for (int t = 0; t < N; t++) {
40     vector<i64> f(M);
41     for (int i = 0; i < M; i++) {
42         f[i] = rng();
43     }
44     Fenwick<i64> fen(n);
45     for (int i = 0; i < n; i++) {
46         fen.add(i, f[a[i]]);
47     }
48     auto b = a;
49     for (int i = 0; i < q; i++) {
50         if (o[i] == 1) {
51             fen.add(l[i], f[r[i]] - f[b[l[i]]]);
52             b[l[i]] = r[i];
53         } else {
54             if (fen.rangeSum(l[i], r[i]) % k[i] != 0) {
55                 ans[i] = 0;
56             }
57         }
58     }
59 }
60 for (int i = 0; i < q; i++) {
61     if (o[i] == 2) {
62         cout << (ans[i] ? "YES" : "NO") << "\n";
63     }
64 }
65 return 0;
66 }

```

## trees

### 900: Michael and Hotel

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Michael and Brian are stuck in a hotel with  $n$  rooms, numbered from 1 to  $n$ , and need to find each other.

But this hotel's doors are all locked and the only way of getting around is by using the teleporters in each room. Room  $i$  has a teleporter that will take you to room  $a_i$  (it might be that  $a_i = i$ ). But they don't know the values of  $a_1, a_2, \dots, a_n$ .

Instead, they can call up the front desk to ask queries. In one query, they give a room  $u$ , a positive integer  $k$ , and a set of rooms  $S$ . The hotel concierge answers whether a person starting in room  $u$ , and using the teleporters  $k$  times, ends up in a room in  $S$ .

Brian is in room 1. Michael wants to know the set  $A$  of rooms so that if he starts in one of those rooms they can use the teleporters to meet up. He can ask at most 2000 queries.

The values  $a_1, a_2, \dots, a_n$  are fixed before the start of the interaction and do not depend on your queries. In other words, the interactor is not adaptive.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 bool query(int u, int k, set<int> s) {
5     cout << "? " << u + 1 << " " << k << " " << s.size();
6     for (auto x : s) {
7         cout << " " << x + 1;
8     }
9     cout << endl;
10    int res;
11    cin >> res;
12    return res;
13 }
14 int main() {
15     ios::sync_with_stdio(false);
16     cin.tie(nullptr);
17     int n;
18     cin >> n;
19     set<int> s;
20     for (int i = 0; i < 125; i++) {
21         int lo = 0, hi = n - 1;
22         int k = n + i;
23         while (lo < hi) {
24             int m = (lo + hi) / 2;
25             set<int> s;
26             for (int i = lo; i <= m; i++) {
27                 s.insert(i);
28             }
29             if (query(0, k, s)) {
30                 hi = m;
31             } else {
32                 lo = m + 1;
33             }
34         }
35         s.insert(lo);
36     }
37     auto fin = [&]() {
38         for (int i = 0; i < n; i++) {
39             if (!s.count(i) && query(i, n, s)) {
40                 s.insert(i);
41             }
42         }
43     };
44 }
```

```

41         }
42     }
43     cout << "! " << s.size();
44     for (auto x : s) {
45         cout << " " << x + 1;
46     }
47     cout << endl;
48     exit(0);
49 }
50 if (s.size() < 125) {
51     fin();
52 }
53 for (int i = 0; i < n; i++) {
54     if (!s.count(i) && query(i, 125, s)) {
55         s.insert(i);
56     }
57 }
58 if (s.size() < 250) {
59     fin();
60 }
61 for (int i = 0; i < n; i++) {
62     if (!s.count(i) && query(i, 250, s)) {
63         s.insert(i);
64     }
65 }
66 fin();
67 return 0;
68 }
```

## 901: Baldman and the military

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Baldman is a warp master. He possesses a unique ability - creating wormholes! Given two positions in space, Baldman can make a wormhole which makes it possible to move between them in both directions. Unfortunately, such operation isn't free for Baldman: each created wormhole makes him lose plenty of hair from his head.

Because of such extraordinary abilities, Baldman has caught the military's attention. He has been charged with a special task. But first things first.

The military base consists of several underground objects, some of which are connected with bidirectional tunnels. There necessarily exists a path through the tunnel system between each pair of objects. Additionally, exactly two objects are connected with surface. For the purposes of security, a patrol inspects the tunnel system every day: he enters one of the objects which are connected with surface, walks the base passing each tunnel at least once and leaves through one of the objects connected with surface. He can enter and leave either through the same object, or through different objects. The

military management noticed that the patrol visits some of the tunnels multiple times and decided to optimize the process. Now they are faced with a problem: a system of wormholes needs to be made to allow of a patrolling which passes each tunnel exactly once. At the same time a patrol is allowed to pass each wormhole any number of times.

This is where Baldman comes to operation: he is the one to plan and build the system of the wormholes. Unfortunately for him, because of strict confidentiality the military can't tell him the arrangement of tunnels. Instead, they insist that his system of portals solves the problem for any arrangement of tunnels which satisfies the given condition. Nevertheless, Baldman has some information: he knows which pairs of objects he can potentially connect and how much it would cost him (in hair). Moreover, tomorrow he will be told which objects (exactly two) are connected with surface. Of course, our hero decided not to waste any time and calculate the minimal cost of getting the job done for some pairs of objects (which he finds likely to be the ones connected with surface). Help Baldman!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct HLD;
5 # struct DSU;
6 int main() {
7     ios::sync_with_stdio(false);
8     cin.tie(nullptr);
9     int n, m;
10    cin >> n >> m;
11    vector<array<int, 3>> e(m);
12    for (int i = 0; i < m; i++) {
13        int u, v, c;
14        cin >> u >> v >> c;
15        u--, v--;
16        e[i] = {c, u, v};
17    }
18    sort(e.begin(), e.end());
19    int cnt = n;
20    DSU dsu(2*n-1);
21    HLD t(2*n-1);
22    i64 ans = 0;
23    vector<int> w(2*n-1);
24    for (auto [c, u, v] : e) {
25        if (!dsu.same(u, v)) {
26            t.addEdge(cnt, dsu.find(u));
27            t.addEdge(cnt, dsu.find(v));
28            dsu.merge(cnt, u);
29            dsu.merge(cnt, v);
30            w[cnt] = c;
31            ans += c;
32            cnt++;
33        }
34    }
35    if (cnt == 2*n-1) {
36        t.work(cnt-1);

```

```

37     }
38     int q;
39     cin >> q;
40     while (q--) {
41         int u, v;
42         cin >> u >> v;
43         u--, v--;
44         if (cnt < 2*n-2) {
45             cout << -1 << "\n";
46         } else if (cnt == 2*n-2) {
47             if (dsu.same(u, v)) {
48                 cout << -1 << "\n";
49             } else {
50                 cout << ans << "\n";
51             }
52         } else {
53             cout << ans - w[t.lca(u, v)] << "\n";
54         }
55     }
56     return 0;
57 }
```

## 902: Bracket Insertion

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Vika likes playing with bracket sequences. Today she wants to create a new bracket sequence using the following algorithm. Initially, Vika's sequence is an empty string, and then she will repeat the following actions  $n$  times:

Choose a place in the current bracket sequence to insert new brackets uniformly at random. If the length of the current sequence is  $k$ , then there are  $k + 1$  such places: before the first bracket, between the first and the second brackets, ..., after the  $k$ -th bracket. In particular, there is one such place in an empty bracket sequence.

Choose string “(” with probability  $p$  or string “)” with probability  $1 - p$  and insert it into the chosen place. The length of the bracket sequence will increase by 2.

A bracket sequence is called regular if it is possible to obtain a correct arithmetic expression by inserting characters ‘+’ and ‘1’ into it. For example, sequences “((())()”, “()”, and “((()())())” are regular, while “)()”, “((())”, and “((())())” are not.

Vika wants to know the probability that her bracket sequence will be a regular one at the end. Help her and find this probability modulo 998 244 353 (see Output section).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 998244353;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
8     if (x < 0) {
9         x += P;
10    }
11    if (x >= P) {
12        x -= P;
13    }
14    return x;
15 }
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25 }
26 # struct Z;
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(nullptr);
30     int n;
31     Z p;
32     cin >> n >> p;
33     p /= 10000;
34     vector<Z> inv(n + 1);
35     for (int i = 1; i <= n; i++) {
36         inv[i] = Z(i).inv();
37     }
38     vector<vector<Z>> tree(n + 1, vector<Z>(n + 1)), fore(tree);
39     fore[0][0] = 1;
40     for (int i = 1; i <= n; i++) {
41         for (int j = 0; j < i; j++) {
42             tree[i][max(0, j - 1)] += fore[i - 1][j] * p * inv[i];
43             tree[i][j + 1] += fore[i - 1][j] * (1 - p) * inv[i];
44         }
45         for (int j = 0; j < i; j++) {
46             for (int k = 1; k <= i; k++) fore[j][k] += fore[j][k - 1];
47             for (int k = 1; k <= i; k++) tree[i - j][k] += tree[i - j][k - 1];
48             for (int k = 0; k <= i; k++) fore[i][k] += fore[j][k] * tree[i - j][k];
49             for (int k = i; k-->0) fore[j][k] -= fore[j][k - 1];
50             for (int k = i; k-->0) tree[i - j][k] -= tree[i - j][k - 1];
51         }
52         for (int k = i; k-->0) fore[i][k] -= fore[i][k - 1];
53     }
54     Z ans = fore[n][0];
55     for (int i = 1; i <= n; i++) {
56         ans /= 2 * i - 1;
57         ans *= i;
58     }
59     cout << ans << "\n";
60     return 0;

```

```
61 }
```

### 903: Distance to the Path

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given a tree consisting of  $n$  vertices. Initially, each vertex has a value 0.

You need to perform  $m$  queries of two types:

You are given a vertex index  $v$ . Print the value of the vertex  $v$ .

You are given two vertex indices  $u$  and  $v$  and values  $k$  and  $d$  ( $d \leq 20$ ). You need to add  $k$  to the value of each vertex such that the distance from that vertex to the path from  $u$  to  $v$  is less than or equal to  $d$ .

The distance between two vertices  $x$  and  $y$  is equal to the number of edges on the path from  $x$  to  $y$ . For example, the distance from  $x$  to  $x$  itself is equal to 0.

The distance from the vertex  $v$  to some path from  $x$  to  $y$  is equal to the minimum among distances from  $v$  to any vertex on the path from  $x$  to  $y$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template <typename T>
5 # struct Fenwick;
6 # struct HLD;
7 int main() {
8     ios::sync_with_stdio(false);
9     cin.tie(nullptr);
10    int n;
11    cin >> n;
12    HLD t(n);
13    for (int i = 1; i < n; i++) {
14        int u, v;
15        cin >> u >> v;
16        u--, v--;
17        t.addEdge(u, v);
18    }
19    t.init();
20    int m;
21    cin >> m;
22    Fenwick<int> fen[21];
23    for (int i = 0; i <= 20; i++) fen[i].init(n);

```

```

24     for (int i = 0; i < m; i++) {
25         int o;
26         cin >> o;
27         if (o == 1) {
28             int v;
29             cin >> v;
30             v--;
31             int ans = 0;
32             for (int i = 0; i <= 20; i++) {
33                 if (v == -1) break;
34                 ans += fen[i].sum(t.in[v] + 1);
35                 v = t.parent[v];
36             }
37             cout << ans << "\n";
38         } else {
39             int u, v, k, d;
40             cin >> u >> v >> k >> d;
41             u--, v--;
42             while (t.top[u] != t.top[v]) {
43                 if (t.dep[t.top[u]] < t.dep[t.top[v]]) swap(u, v);
44                 fen[d].add(t.in[t.top[u]], k);
45                 fen[d].add(t.in[u] + 1, -k);
46                 u = t.parent[t.top[u]];
47             }
48             if (t.dep[u] > t.dep[v]) swap(u, v);
49             fen[d].add(t.in[u] + 1, k);
50             fen[d].add(t.in[v] + 1, -k);
51             while (1) {
52                 fen[d].add(t.in[u], k);
53                 fen[d].add(t.in[u] + 1, -k);
54                 if (!d--) break;
55                 fen[d].add(t.in[u], k);
56                 fen[d].add(t.in[u] + 1, -k);
57                 if (t.parent[u] == -1) {
58                     while (d--) {
59                         fen[d].add(t.in[u], k);
60                         fen[d].add(t.in[u] + 1, -k);
61                     }
62                     break;
63                 }
64                 u = t.parent[u];
65             }
66         }
67     }
68     return 0;
69 }
```

## black

### constructive algorithms

#### 904: Minimum Segments

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input

- Output file: standard output

You had a sequence  $a_1, a_2, \dots, a_n$  consisting of integers from 1 to  $n$ , not necessarily distinct. For some unknown reason, you decided to calculate the following characteristic of the sequence:

Let  $r_i$  ( $1 \leq i \leq n$ ) be the smallest  $j \geq i$  such that on the subsegment  $a_i, a_{i+1}, \dots, a_j$  all distinct numbers from the sequence  $a$  appear. More formally, for any  $k \in [1, n]$ , there exists  $l \in [i, j]$  such that  $a_k = a_l$ . If such  $j$  does not exist,  $r_i$  is considered to be equal to  $n + 1$ .

The characteristic of the sequence  $a$  is defined as the sequence  $r_1, r_2, \dots, r_n$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 // mt19937 rng;
5 // int ok = 0;
6 void solve() {
7     // int n = rng() % 10 + 1;
8     int n;
9     cin >> n;
10    // vector<int> A(n + 1);
11    // set<int> S;
12    // for (int i = 1; i <= n; i++) {
13    //     A[i] = rng() % i + 1;
14    //     S.insert(A[i]);
15    // }
16    vector<int> r(n + 2);
17    for (int i = 1; i <= n; i++) {
18        cin >> r[i];
19    }
20    // for (int i = 1; i <= n; i++) {
21    //     int j = i - 1;
22    //     set<int> T;
23    //     while (j <= n && T.size() < S.size()) {
24    //         j++;
25    //         if (j <= n) {
26    //             T.insert(A[j]);
27    //         }
28    //     }
29    //     r[i] = j;
30    //     // cin >> r[i];
31    // }
32    r[n + 1] = n + 1;
33    if (r[1] > n) {
34        cout << "No\n";
35        return;
36    }
37    if (!is_sorted(r.begin(), r.end())) {
38        cout << "No\n";
39        return;
40    }
41    vector<int> L(n + 2, -1), R(n + 2, -1);
42    for (int i = 0; i <= n; i++) {
43        if (r[i] < r[i + 1]) {

```

```
44         R[i] = r[i + 1];
45         L[r[i + 1]] = i;
46     }
47 }
48 vector<int> pl, pr;
49 for (int i = 1; i <= n; i++) {
50     if (R[i] == -1) {
51         pl.push_back(i);
52     }
53     if (L[i] == -1) {
54         pr.push_back(i);
55     }
56 }
57 int k = pl.size();
58 assert(pr.size() == k);
59 int res = 0;
60 for (int i = 0, j = 0; i < k; i++) {
61     while (j < k && pl[i] >= pr[j]) {
62         j++;
63     }
64     res = max(res, j - i);
65 }
66 for (int i = 0; i < k; i++) {
67     R[pl[i]] = i + res < k ? pr[i + res] : n + 1;
68     L[pr[i]] = i - res >= 0 ? pl[i - res] : 0;
69 }
70 vector<int> a(n + 1);
71 int tot = 0;
72 for (int i = 1; i <= n; i++) {
73     a[i] = L[i] == 0 ? ++tot : a[L[i]];
74 }
75 vector<int> f(n + 2);
76 for (int i = 1; i <= n; i++) {
77     f[i] = max(f[i], R[i]);
78     f[L[i]] = max(f[L[i]], i);
79 }
80 for (int i = 1; i <= n + 1; i++) {
81     f[i] = max(f[i], f[i - 1]);
82 }
83 for (int i = 1; i <= n; i++) {
84     if (r[i] < f[i - 1]) {
85         cout << "No\n";
86         return;
87     }
88 }
89 cout << "Yes\n";
90 for (int i = 1; i <= n; i++) {
91     cout << a[i] << " \n"[i == n];
92 }
93 }
94 int main() {
95     ios::sync_with_stdio(false);
96     cin.tie(nullptr);
97     int t;
98     cin >> t;
99     while (t--) {
100         solve();
101     }
102     return 0;
103 }
```

## 905: Good Colorings

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Alice suggested Bob to play a game. Bob didn't like this idea, but he couldn't refuse Alice, so he asked you to write a program that would play instead of him.

The game starts with Alice taking out a grid sheet of size  $n \times n$ , the cells of which are initially not colored. After that she colors some  $2n$  cells with colors 1, 2, ...,  $2n$ , respectively, and informs Bob about these cells.

In one move, Bob can point to a cell that has not been colored yet and ask Alice to color that cell. Alice colors that cell with one of the  $2n$  colors of her choice, informing Bob of the chosen color. Bob can make no more than 10 moves, after which he needs to find a good set of four cells.

A set of four cells is considered good if the following conditions are met:

All the cells in the set are colored;

No two cells in the set are colored with the same color;

The centers of the cells form a rectangle with sides parallel to the grid lines.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<vector<pair<int, int>>> adj(2 * n);
8     for (int i = 0; i < 2 * n; i++) {
9         int x, y;
10        cin >> x >> y;
11        x--, y--;
12        y += n;
13        adj[x].emplace_back(y, i);
14        adj[y].emplace_back(x, i);
15    }
16    vector<bool> vis(2 * n);
17    vector<int> par(2 * n), dep(2 * n), parc(2 * n);
18    vector<int> cyc;
19    vector<int> col;
20    auto dfs = [&](auto self, int x) -> void {
21        vis[x] = true;
22        for (auto [y, c] : adj[x]) {

```

```

23         if (!vis[y]) {
24             par[y] = x;
25             parc[y] = c;
26             dep[y] = dep[x] + 1;
27             self(self, y);
28         } else if (dep[y] > dep[x] && cyc.empty()) {
29             for (int i = y; i != x; i = par[i]) {
30                 cyc.push_back(i);
31                 col.push_back(parc[i]);
32             }
33             cyc.push_back(x);
34             col.push_back(c);
35         }
36     }
37 };
38 for (int i = 0; i < 2 * n; i++) {
39     if (!vis[i] && cyc.empty()) {
40         dfs(dfs, i);
41     }
42 }
43 assert(!cyc.empty());
44 if (cyc[0] >= n) {
45     rotate(cyc.begin(), cyc.begin() + 1, cyc.end());
46     rotate(col.begin(), col.begin() + 1, col.end());
47 }
48 while (cyc.size() > 4) {
49     int m = (cyc.size() + 3) / 4 * 2;
50     cout << "? " << cyc[0] + 1 << " " << cyc[m - 1] - n + 1 << endl;
51     int res;
52     cin >> res;
53     res--;
54     if (find(col.begin(), col.begin() + m - 1, res) == col.begin() + m - 1) {
55         cyc.erase(cyc.begin() + m, cyc.end());
56         col.erase(col.begin() + m, col.end());
57         col.back() = res;
58     } else {
59         cyc.erase(cyc.begin() + 1, cyc.begin() + m - 1);
60         col.erase(col.begin() + 1, col.begin() + m - 1);
61         col[0] = res;
62     }
63 }
64 cout << "! " << cyc[0] + 1 << " " << cyc[2] + 1 << " " << cyc[1] - n + 1 << " " << cyc
65 [3] - n + 1 << endl;
66 string s;
67 cin >> s;
68 }
69 int main() {
70     ios::sync_with_stdio(false);
71     cin.tie(nullptr);
72     int t;
73     cin >> t;
74     while (t--) {
75         solve();
76     }
77     return 0;
78 }
```

**906: Two Permutations (Hard Version)**

- Time limit: 2 seconds

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. The difference between the two versions is that you have to minimize the number of operations in this version. You can make hacks only if both versions of the problem are solved.

You have two permutations<sup>†</sup>  $p_1, p_2, \dots, p_n$  (of integers 1 to  $n$ ) and  $q_1, q_2, \dots, q_m$  (of integers 1 to  $m$ ). Initially  $p_i = a_i$  for  $i = 1, 2, \dots, n$ , and  $q_j = b_j$  for  $j = 1, 2, \dots, m$ . You can apply the following operation on the permutations several (possibly, zero) times.

In one operation,  $p$  and  $q$  will change according to the following three steps:

You choose integers  $i, j$  which satisfy  $1 \leq i \leq n$  and  $1 \leq j \leq m$ .

Permutation  $p$  is partitioned into three parts using  $p_i$  as a pivot: the left part is formed by elements  $p_1, p_2, \dots, p_{i-1}$  (this part may be empty), the middle part is the single element  $p_i$ , and the right part is  $p_{i+1}, p_{i+2}, \dots, p_n$  (this part may be empty). To proceed, swap the left and the right parts of this partition. Formally, after this step,  $p$  will become  $p_{i+1}, p_{i+2}, \dots, p_n, p_i, p_1, p_2, \dots, p_{i-1}$ . The elements of the newly formed  $p$  will be reindexed starting from 1.

Perform the same transformation on  $q$  with index  $j$ . Formally, after this step,  $q$  will become  $q_{j+1}, q_{j+2}, \dots, q_m, q_j, q_1, q_2, \dots, q_{j-1}$ . The elements of the newly formed  $q$  will be reindexed starting from 1.

Your goal is to simultaneously make  $p_i = i$  for  $i = 1, 2, \dots, n$ , and  $q_j = j$  for  $j = 1, 2, \dots, m$ .

Find any way to achieve the goal using the minimum number of operations possible, or say that none exists. Please note that you have to minimize the number of operations.

<sup>†</sup> A permutation of length  $k$  is an array consisting of  $k$  distinct integers from 1 to  $k$  in arbitrary order. For example,  $[2, 3, 1, 5, 4]$  is a permutation, but  $[1, 2, 2]$  is not a permutation (2 appears twice in the array), and  $[1, 3, 4]$  is also not a permutation ( $k = 3$  but there is 4 in the array).

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 array<int, 2> solve(vector<int> a) {
5     int n = a.size();
6     array<int, 2> ans{-1, -1};
7     for (int i = 0; i < n; i++) {
8         int res = 0;
9         auto b = a;
10        while (b[0] != 0) {
11            swap(b[0], b[b[0]]);
```

```

12         res++;
13     }
14     for (int j = 0; j < n; j++) {
15         if (b[j] != j) {
16             swap(b[0], b[j]);
17             res++;
18             while (b[0] != 0) {
19                 swap(b[0], b[b[0]]);
20                 res++;
21             }
22         }
23     }
24     if (ans[res % 2] == -1 || ans[res % 2] > res) {
25         ans[res % 2] = res;
26     }
27     for (int j = 0; j < n; j++) {
28         a[j] = (a[j] + 1) % n;
29     }
30 }
31 return ans;
32 }
33 vector<int> get(vector<int> a, int k) {
34     int n = a.size();
35     for (int i = 0; i < n; i++) {
36         int res = 0;
37         auto b = a;
38         vector<int> ans;
39         while (b[0] != 0) {
40             ans.push_back((b[b[0]] - b[0] + n) % n);
41             swap(b[0], b[b[0]]);
42             res++;
43         }
44         for (int j = 0; j < n; j++) {
45             if (b[j] != j) {
46                 ans.push_back((b[j] - b[0] + n) % n);
47                 swap(b[0], b[j]);
48                 res++;
49                 while (b[0] != 0) {
50                     ans.push_back((b[b[0]] - b[0] + n) % n);
51                     swap(b[0], b[b[0]]);
52                     res++;
53                 }
54             }
55         }
56         if (res <= k && (k - res) % 2 == 0) {
57             while (res < k) {
58                 ans.push_back((b[1] - b[0] + n) % n);
59                 swap(b[0], b[1]);
60                 res++;
61             }
62         }
63         return ans;
64     }
65     for (int j = 0; j < n; j++) {
66         a[j] = (a[j] + 1) % n;
67     }
68     assert(false);
69 }
70 int main() {
71     ios::sync_with_stdio(false);
72     cin.tie(nullptr);
73     int n, m;
74     cin >> n >> m;
75     vector<int> a(n + 1), b(m + 1);

```

```

76     for (int i = 1; i <= n; i++) {
77         int x;
78         cin >> x;
79         a[x] = i;
80     }
81     for (int i = 1; i <= m; i++) {
82         int x;
83         cin >> x;
84         b[x] = i;
85     }
86     auto [ea, oa] = solve(a);
87     auto [eb, ob] = solve(b);
88     int ans = -1;
89     if (ea != -1 && eb != -1) {
90         ans = max(ea, eb);
91     }
92     if (oa != -1 && ob != -1 && (ans == -1 || ans > max(oa, ob))) {
93         ans = max(oa, ob);
94     }
95     if (ans == -1) {
96         cout << -1 << "\n";
97         return 0;
98     }
99     auto sa = get(a, ans);
100    auto sb = get(b, ans);
101    cout << ans << "\n";
102    for (int i = 0; i < ans; i++) {
103        cout << sa[i] << " " << sb[i] << "\n";
104    }
105    return 0;
106 }
```

## 907: Magic Square

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Aquamoon has a Rubik's Square which can be seen as an  $n \times n$  matrix, the elements of the matrix constitute a permutation of numbers  $1, \dots, n^2$ .

Aquamoon can perform two operations on the matrix:

Row shift, i.e. shift an entire row of the matrix several positions (at least 1 and at most  $n - 1$ ) to the right. The elements that come out of the right border of the matrix are moved to the beginning of the row. For example, shifting a row  $(a \ b \ c)$  by 2 positions would result in  $(b \ c \ a)$ ;

Column shift, i.e. shift an entire column of the matrix several positions (at least 1 and at most  $n - 1$ ) downwards. The elements that come out of the lower border of the matrix are moved to the beginning of the column. For example, shifting a column  $\begin{pmatrix} a \\ b \\ c \end{pmatrix}$  by 2 positions would result in  $\begin{pmatrix} b \\ c \\ a \end{pmatrix}$ .

The rows are numbered from 1 to  $n$  from top to bottom, the columns are numbered from 1 to  $n$  from left to right. The cell at the intersection of the  $x$ -th row and the  $y$ -th column is denoted as  $(x, y)$ .

Aquamoon can perform several (possibly, zero) operations, but she has to obey the following restrictions:

each row and each column can be shifted at most once;

each integer of the matrix can be moved at most twice;

the offsets of any two integers moved twice cannot be the same. Formally, if integers  $a$  and  $b$  have been moved twice, assuming  $a$  has changed its position from  $(x_1, y_1)$  to  $(x_2, y_2)$ , and  $b$  has changed its position from  $(x_3, y_3)$  to  $(x_4, y_4)$ , then  $x_2 - x_1 \not\equiv x_4 - x_3 \pmod{n}$  or  $y_2 - y_1 \not\equiv y_4 - y_3 \pmod{n}$ .

Aquamoon wonders in how many ways she can transform the Rubik's Square from the given initial state to a given target state. Two ways are considered different if the sequences of applied operations are different. Since the answer can be very large, print the result modulo 998 244 353.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 void solve() {
36     int n;

```

```

37     cin >> n;
38     vector a(n, vector<int>(n)), b(n, vector<int>(n));
39     vector<array<int, 2>> pa(n * n), pb(n * n);
40     for (int i = 0; i < n; i++) {
41         for (int j = 0; j < n; j++) {
42             cin >> a[i][j];
43             a[i][j]--;
44             pa[a[i][j]] = {i, j};
45         }
46     }
47     for (int i = 0; i < n; i++) {
48         for (int j = 0; j < n; j++) {
49             cin >> b[i][j];
50             b[i][j]--;
51             pb[b[i][j]] = {i, j};
52         }
53     }
54     vector<int> sr(n), sc(n);
55     for (int i = 0; i < n * n; i++) {
56         auto [x1, y1] = pa[i];
57         auto [x2, y2] = pb[i];
58         if (x1 == x2) {
59             sr[x1] = (y2 - y1 + n) % n;
60         }
61         if (y1 == y2) {
62             sc[y1] = (x2 - x1 + n) % n;
63         }
64     }
65     // for (int i = 0; i < n; i++) {
66     //     cerr << sr[i] << " \n"[i == n - 1];
67     // }
68     // for (int i = 0; i < n; i++) {
69     //     cerr << sc[i] << " \n"[i == n - 1];
70     // }
71     bool ok = true;
72     for (int i = 0; i < n * n; i++) {
73         auto [x1, y1] = pa[i];
74         auto [x2, y2] = pb[i];
75         if (x1 == x2) {
76             if (sr[x1] != (y2 - y1 + n) % n) {
77                 ok = false;
78             }
79         }
80         if (y1 == y2) {
81             if (sc[y1] != (x2 - x1 + n) % n) {
82                 ok = false;
83             }
84         }
85     }
86     vector<int> cntr(n), cntc(n);
87     for (int i = 0; i < n; i++) {
88         cntr[sr[i]]++;
89         cntc[sc[i]]++;
90     }
91     if (cntr[0] == n) {
92         for (int i = 0; i < n; i++) {
93             vector<int> v(n);
94             for (int j = 0; j < n; j++) {
95                 v[j] = a[j][i];
96             }
97             rotate(v.begin(), v.end() - sc[i], v.end());
98             for (int j = 0; j < n; j++) {
99                 a[j][i] = v[j];
100            }
101        }
102    }

```

```

101         }
102     if (a == b) {
103         cout << comb.fac(n - cntc[0]) << "\n";
104     } else {
105         cout << 0 << "\n";
106     }
107     return;
108 }
109 if (cntc[0] == n) {
110     for (int i = 0; i < n; i++) {
111         rotate(a[i].begin(), a[i].end() - sr[i], a[i].end());
112     }
113     if (a == b) {
114         cout << comb.fac(n - cntr[0]) << "\n";
115     } else {
116         cout << 0 << "\n";
117     }
118     return;
119 }
120 for (int i = 1; i < n; i++) {
121     if (cntr[i] > 1) {
122         ok = false;
123     }
124     if (cntc[i] > 1) {
125         ok = false;
126     }
127 }
128 if (!ok) {
129     cout << 0 << "\n";
130     return;
131 }
132 vector<vector<int>> adj(2 * n);
133 vector<int> deg(2 * n);
134 auto addEdge = [&](int x, int y) {
135     deg[y]++;
136     adj[x].push_back(y);
137 };
138 for (int i = 0; i < n * n; i++) {
139     auto [x1, y1] = pa[i];
140     auto [x2, y2] = pb[i];
141     if (x1 == x2 && y1 == y2) {
142         continue;
143     }
144     if (x1 == x2) {
145         if (sc[y2]) {
146             addEdge(n + y2, x1);
147             // cerr << "col " << y2 << " -> " << "row " << x1 << "\n";
148         }
149         if (sc[y1]) {
150             addEdge(x1, n + y1);
151             // cerr << "row " << x1 << " -> " << "col " << y1 << "\n";
152         }
153     } else if (y1 == y2) {
154         if (sr[x2]) {
155             addEdge(x2, n + y1);
156             // cerr << "row " << x2 << " -> " << "col " << y1 << "\n";
157         }
158         if (sr[x1]) {
159             addEdge(n + y1, x1);
160             // cerr << "col " << y1 << " -> " << "row " << x1 << "\n";
161         }
162     } else {
163         if (y2 == (y1 + sr[x1]) % n && x2 == (x1 + sc[y2]) % n) {
164             if (sc[y1]) {

```

```

165             addEdge(x1, n + y1);
166             // cerr << "row " << x1 << " -> " << "col " << y1 << "\n";
167         }
168         addEdge(x1, n + y2);
169         // cerr << "row " << x1 << " -> " << "col " << y2 << "\n";
170         if (sr[x2]) {
171             addEdge(x2, n + y2);
172             // cerr << "row " << x2 << " -> " << "col " << y2 << "\n";
173         }
174     } else if (x2 == (x1 + sc[y1]) % n && y2 == (y1 + sr[x2]) % n) {
175         if (sr[x1]) {
176             addEdge(n + y1, x1);
177             // cerr << "col " << y1 << " -> " << "row " << x1 << "\n";
178         }
179         addEdge(n + y1, x2);
180         // cerr << "col " << y1 << " -> " << "row " << x2 << "\n";
181         if (sc[y2]) {
182             addEdge(n + y2, x2);
183             // cerr << "col " << y2 << " -> " << "row " << x2 << "\n";
184         }
185     } else {
186         cout << 0 << "\n";
187         return;
188     }
189 }
190 Z ans = 1;
191 vector<int> q;
192 for (int i = 0; i < n; i++) {
193     if (sr[i] && deg[i] == 0) {
194         q.push_back(i);
195     }
196     if (sc[i] && deg[n + i] == 0) {
197         q.push_back(n + i);
198     }
199 }
200 for (int i = 0; i < q.size(); i++) {
201     int x = q[i];
202     for (auto y : adj[x]) {
203         if (--deg[y] == 0) {
204             q.push_back(y);
205         }
206     }
207 }
208 if (q.size() != 2 * n - cntr[0] - cntc[0]) {
209     cout << 0 << "\n";
210     return;
211 }
212 for (int l = 0, r = 0; l < q.size(); l = r) {
213     while (r < q.size() && q[l] / n == q[r] / n) {
214         r++;
215     }
216     ans *= comb.fac(r - l);
217 }
218 cout << ans << "\n";
219 }
220 int main() {
221     ios::sync_with_stdio(false);
222     cin.tie(nullptr);
223     int t;
224     cin >> t;
225     while (t--) {
226         solve();
227     }
228 }
```

```
229     return 0;
230 }
```

## 908: Bus Routes

- Time limit: 2.5 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

There is a country consisting of  $n$  cities and  $n - 1$  bidirectional roads connecting them such that we can travel between any two cities using these roads. In other words, these cities and roads form a tree.

There are  $m$  bus routes connecting the cities together. A bus route between city  $x$  and city  $y$  allows you to travel between any two cities in the simple path between  $x$  and  $y$  with this route.

Determine if for every pair of cities  $u$  and  $v$ , you can travel from  $u$  to  $v$  using at most two bus routes.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct HLD;
5 // mt19937 rng;
6 void solve() {
7     int n, m;
8     cin >> n >> m;
9     // n = rng() % 10 + 1;
10    // m = rng() % 10 + 1;
11    // cerr << n << " " << m << "\n";
12    HLD t(n);
13    vector<int> deg(n);
14    for (int i = 0; i < n - 1; i++) {
15        int u, v;
16        cin >> u >> v;
17        // v = i + 2;
18        // u = rng() % (v - 1) + 1;
19        // cerr << u << " " << v << "\n";
20        u--, v--;
21        deg[u]++;
22        deg[v]++;
23        t.addEdge(u, v);
24    }
25    t.work();
26    vector<vector<int>> route(n);
27    for (int i = 0; i < m; i++) {
28        int x, y;
29        cin >> x >> y;
30        // x = rng() % n + 1;
31        // y = rng() % n + 1;
32        // cerr << x << " " << y << "\n";
33        x--, y--;
```

```

34         route[x].push_back(y);
35         route[y].push_back(x);
36     }
37     vector<int> d(n);
38     auto cmp = [&](int i, int j) {
39         return t.in[i] < t.in[j];
40     };
41     vector<vector<int>> l(n);
42     for (int i = 0; i < n; i++) {
43         if (deg[i] == 1) {
44             auto a = route[i];
45             a.push_back(i);
46             sort(a.begin(), a.end(), cmp);
47             int m = unique(a.begin(), a.end()) - a.begin();
48             a.resize(m);
49             l[i].resize(m);
50             route[i] = a;
51             for (int j = 1; j < m; j++) {
52                 l[i][j] = t.lca(a[j], a[j - 1]);
53             }
54             // cerr << "x : " << i + 1 << "\n";
55             // for (auto y : a) {
56             //     cerr << y + 1 << " \n"[y == a.back()];
57             // }
58             l[i][0] = t.parent[t.lca(a[0], a.back())];
59         }
60     }
61     auto check = [&](int k) {
62         int cnt = 0;
63         d.assign(n, 0);
64         for (int i = 0; i < k; i++) {
65             if (deg[i] == 1) {
66                 cnt++;
67                 for (auto x : route[i]) {
68                     d[x]++;
69                 }
70                 for (auto x : l[i]) {
71                     if (x != -1) {
72                         d[x]--;
73                     }
74                 }
75             }
76         }
77         for (int i = n - 1; i; i--) {
78             int x = t.seq[i];
79             d[t.parent[x]] += d[x];
80         }
81         return find(d.begin(), d.end(), cnt) != d.end();
82     };
83     if (check(n)) {
84         cout << "YES\n";
85         return;
86     }
87     cout << "NO\n";
88     int lo = 0, hi = n - 1;
89     while (lo < hi) {
90         int x = (lo + hi + 1) / 2;
91         if (check(x)) {
92             lo = x;
93         } else {
94             hi = x - 1;
95         }
96     }
97     int y = lo;

```

```

98     d.assign(n, 0);
99     for (auto x : route[y]) {
100        d[x]++;
101    }
102    for (auto x : l[y]) {
103        if (x != -1) {
104            d[x]--;
105        }
106    }
107    for (int i = n - 1; i; i--) {
108        int x = t.seq[i];
109        d[t.parent[x]] += d[x];
110    }
111    for (int i = 1; i < n; i++) {
112        int x = t.seq[i];
113        d[x] += d[t.parent[x]];
114    }
115    for (int i = 0; i < y; i++) {
116        if (deg[i] == 1) {
117            int res = 0;
118            for (auto x : route[i]) {
119                res += d[x];
120            }
121            for (auto x : l[i]) {
122                if (x != -1) {
123                    res -= d[x];
124                }
125            }
126            if (!res) {
127                cout << i + 1 << " " << y + 1 << "\n";
128                return;
129            }
130        }
131    }
132    assert(false);
133 }
134 int main() {
135     ios::sync_with_stdio(false);
136     cin.tie(nullptr);
137     int t;
138     cin >> t;
139     while (t--) {
140         solve();
141     }
142     return 0;
143 }
```

**909: OH NO1 (-2-3-4)**

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given an undirected graph with  $n$  vertices and  $3m$  edges. The graph may contain multi-edges, but does not contain self-loops.

The graph satisfies the following property: the given edges can be divided into  $m$  groups of 3, such that each group is a triangle.

A triangle is defined as three edges  $(a, b)$ ,  $(b, c)$  and  $(c, a)$  for some three distinct vertices  $a, b, c$  ( $1 \leq a, b, c \leq n$ ).

Initially, each vertex  $v$  has a non-negative integer weight  $a_v$ . For every edge  $(u, v)$  in the graph, you should perform the following operation exactly once:

Choose an integer  $x$  between 1 and 4. Then increase both  $a_u$  and  $a_v$  by  $x$ .

After performing all operations, the following requirement should be satisfied: if  $u$  and  $v$  are connected by an edge, then  $a_u \neq a_v$ .

It can be proven this is always possible under the constraints of the task. Output a way to do so, by outputting the choice of  $x$  for each edge. It is easy to see that the order of operations does not matter. If there are multiple valid answers, output any.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 mt19937 rng;
5 void solve() {
6     int n, m;
7     cin >> n >> m;
8     vector<int> a(n);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11     }
12     vector<vector<pair<int, int>>> adj(n);
13     vector<array<int, 3>> ver(m);
14     vector f(m, array<int, 3>{});
15     for (int i = 0; i < m; i++) {
16         int u, v, w;
17         cin >> u >> v >> w;
18         u--, v--, w--;
19         // do {
20         //     u = rng() % n;
21         //     v = rng() % n;
22         //     w = rng() % n;
23         // } while (u == v || v == w || u == w);
24         ver[i] = {u, v, w};
25         adj[u].push_back({i, 0});
26         adj[v].push_back({i, 1});
27         adj[w].push_back({i, 2});
28         array<int, 3> b{0, 1, 2};
29         sort(b.begin(), b.end(), [&](int x, int y) {
30             return ver[i][x] < ver[i][y];
31         });
32         for (int j = 0; j < 3; j++) {
33             a[ver[i][b[j]]] += j + 3;
34             f[i][b[j]] += j + 3;
35         }
36     }
}

```

```

37     priority_queue<pair<int, int>> h;
38     for (int i = 0; i < n; i++) {
39         h.emplace(-a[i], i);
40     }
41     while (!h.empty()) {
42         auto [v, x] = h.top();
43         h.pop();
44         if (a[x] != -v) {
45             continue;
46         }
47         for (auto [j, t] : adj[x]) {
48             int y = ver[j][(t + 1) % 3];
49             int z = ver[j][(t + 2) % 3];
50             if (a[x] == a[y] && a[x] == a[z]) {
51                 a[y] += 1;
52                 f[j][(t + 1) % 3] += 1;
53                 h.emplace(-a[y], y);
54                 a[z] += 1;
55                 f[j][(t + 2) % 3] += 1;
56                 h.emplace(-a[z], z);
57             } else if (a[x] == a[y]) {
58                 a[y] += 2;
59                 f[j][(t + 1) % 3] += 2;
60                 h.emplace(-a[y], y);
61             } else if (a[x] == a[z]) {
62                 a[z] += 2;
63                 f[j][(t + 2) % 3] += 2;
64                 h.emplace(-a[z], z);
65             }
66         }
67     }
68     for (int i = 0; i < m; i++) {
69         auto [u, v, w] = ver[i];
70         assert(a[u] != a[v]);
71         assert(a[u] != a[w]);
72         assert(a[v] != a[w]);
73         int a = (f[i][0] + f[i][1] - f[i][2]) / 2;
74         int b = (f[i][1] + f[i][2] - f[i][0]) / 2;
75         int c = (f[i][2] + f[i][0] - f[i][1]) / 2;
76         // cerr << f[i][0] << " " << f[i][1] << " " << f[i][2] << "\n";
77         assert(1 <= a && a <= 4);
78         assert(1 <= b && b <= 4);
79         assert(1 <= c && c <= 4);
80         cout << a << " " << b << " " << c << "\n";
81     }
82 }
83 int main() {
84     ios::sync_with_stdio(false);
85     cin.tie(nullptr);
86     int t;
87     cin >> t;
88     while (t--) {
89         solve();
90     }
91     return 0;
92 }
```

**910: Maximum Permutation**

- Time limit: 2 seconds

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Ecrade bought a deck of cards numbered from 1 to  $n$ . Let the value of a permutation  $a$  of length  $n$  be  $\min_{i=1}^{n-k+1} \sum_{j=i}^{i+k-1} a_j$ .

Ecrade wants to find the most valuable one among all permutations of the cards. However, it seems a little difficult, so please help him!

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 vector<int> con# structMultiple(int n,
5 vector<int> con# struct(int n,
6 void solve() {
7     int n, k;
8     cin >> n >> k;
9     auto ans = con# struct(n, k);
10    i64 res = 1E18;
11    for (int i = 0; i + k <= n; i++) {
12        res = min(res, pre[i + k] - pre[i]);
13    }
14    cout << res << "\n";
15    for (int i = 0; i < n; i++) {
16        cout << ans[i] + 1 << " \n"[i == n - 1];
17    }
18 }
19 int main() {
20     ios::sync_with_stdio(false);
21     cin.tie(nullptr);
22     int t;
23     cin >> t;
24     while (t--) {
25         solve();
26     }
27     return 0;
28 }
```

## 911: Koxia, Mahiru and Winter Festival

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Koxia and Mahiru are enjoying the Winter Festival. The streets of the Winter Festival can be represented as a  $n \times n$  undirected grid graph. Formally, the set of vertices is  $\{(i, j) \mid 1 \leq i, j \leq n\}$  and two vertices

$(i_1, j_1)$  and  $(i_2, j_2)$  are connected by an edge if and only if  $|i_1 - i_2| + |j_1 - j_2| = 1$ .

Koxia and Mahiru are planning to visit The Winter Festival by traversing  $2n$  routes. Although routes are not planned yet, the endpoints of the routes are already planned as follows:

In the  $i$ -th route, they want to start from vertex  $(1, i)$  and end at vertex  $(n, p_i)$ , where  $p$  is a permutation of length  $n$ .

In the  $(i + n)$ -th route, they want to start from vertex  $(i, 1)$  and end at vertex  $(q_i, n)$ , where  $q$  is a permutation of length  $n$ .

Your task is to find a routing scheme -  $2n$  paths where each path connects the specified endpoints. Let's define the congestion of an edge as the number of times it is used (both directions combined) in the routing scheme. In order to ensure that Koxia and Mahiru won't get too bored because of traversing repeated edges, please find a routing scheme that minimizes the maximum congestion among all edges.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 vector<vector<pair<int, int>>> solve(vector<int> p, vector<int> q) {
5     int n = p.size();
6     if (n == 0) {
7         return {};
8     }
9     if (n == 1) {
10        return {{0, 0}}, {{0, 0}};;
11    }
12    vector<vector<pair<int, int>>> ans(2 * n);
13    vector<bool> visu(n), visl(n), visr(n), visd(n);
14    visu[0] = true;
15    visd[p[0]] = true;
16    for (int i = 0; i < n; i++) {
17        ans[0].emplace_back(i, 0);
18    }
19    for (int j = 1; j <= p[0]; j++) {
20        ans[0].emplace_back(n - 1, j);
21    }
22    visu[n - 1] = true;
23    visd[p[n - 1]] = true;
24    for (int i = 0; i < n; i++) {
25        ans[n - 1].emplace_back(i, n - 1);
26    }
27    for (int j = n - 2; j >= p[n - 1]; j--) {
28        ans[n - 1].emplace_back(n - 1, j);
29    }
30    visl[0] = true;
31    visr[q[0]] = true;
32    for (int i = 0; i < n; i++) {
33        ans[n].emplace_back(0, i);
34    }
35    for (int i = 1; i <= q[0]; i++) {
36        ans[n].emplace_back(i, n - 1);

```

```

37     }
38     if (q[0] != 0) {
39         int x = find(q.begin(), q.end(), 0) - q.begin();
40         visl[x] = true;
41         visr[0] = true;
42         for (int i = x; i >= 1; i--) {
43             ans[n + x].emplace_back(i, 0);
44         }
45         for (int i = 0; i < n; i++) {
46             ans[n + x].emplace_back(0, i);
47         }
48     } else {
49         visl[1] = true;
50         visr[q[1]] = true;
51         ans[n + 1].emplace_back(1, 0);
52         for (int i = 0; i < n; i++) {
53             ans[n + 1].emplace_back(0, i);
54         }
55         for (int i = 1; i <= q[1]; i++) {
56             ans[n + 1].emplace_back(i, n - 1);
57         }
58     }
59     vector<int> fu(n), fd(n), fl(n), fr(n);
60     for (int i = 0, j = 0; i < n; i++) {
61         if (!visu[i]) {
62             fu[i] = j++;
63         }
64     }
65     for (int i = 0, j = 0; i < n; i++) {
66         if (!visd[i]) {
67             fd[i] = j++;
68         }
69     }
70     for (int i = 0, j = 0; i < n; i++) {
71         if (!visl[i]) {
72             fl[i] = j++;
73         }
74     }
75     for (int i = 0, j = 0; i < n; i++) {
76         if (!visr[i]) {
77             fr[i] = j++;
78         }
79     }
80     vector<int> np(n - 2), nq(n - 2);
81     for (int i = 0; i < n; i++) {
82         if (!visu[i]) {
83             np[fu[i]] = fd[p[i]];
84         }
85         if (!visl[i]) {
86             nq[fl[i]] = fr[q[i]];
87         }
88     }
89     auto res = solve(np, nq);
90     for (int i = 0; i < n; i++) {
91         if (!visu[i]) {
92             for (int x = i; x < fu[i] + 1; x++) {
93                 ans[i].emplace_back(0, x);
94             }
95             for (int x = i; x > fu[i] + 1; x--) {
96                 ans[i].emplace_back(0, x);
97             }
98             ans[i].emplace_back(0, fu[i] + 1);
99             for (auto [x, y] : res[fu[i]]) {
100                ans[i].emplace_back(x + 1, y + 1);

```

```
101         }
102         ans[i].emplace_back(n - 1, fd[p[i]] + 1);
103         for (int x = fd[p[i]] + 2; x <= p[i]; x++) {
104             ans[i].emplace_back(n - 1, x);
105         }
106         for (int x = fd[p[i]]; x >= p[i]; x--) {
107             ans[i].emplace_back(n - 1, x);
108         }
109     }
110     if (!visl[i]) {
111         for (int x = i; x < fl[i] + 1; x++) {
112             ans[n + i].emplace_back(x, 0);
113         }
114         for (int x = i; x > fl[i] + 1; x--) {
115             ans[n + i].emplace_back(x, 0);
116         }
117         ans[n + i].emplace_back(fl[i] + 1, 0);
118         for (auto [x, y] : res[n - 2 + fl[i]]) {
119             ans[n + i].emplace_back(x + 1, y + 1);
120         }
121         ans[n + i].emplace_back(fr[q[i]] + 1, n - 1);
122         for (int x = fr[q[i]] + 2; x <= q[i]; x++) {
123             ans[n + i].emplace_back(x, n - 1);
124         }
125         for (int x = fr[q[i]]; x >= q[i]; x--) {
126             ans[n + i].emplace_back(x, n - 1);
127         }
128     }
129     return ans;
130 }
131 int main() {
132     ios::sync_with_stdio(false);
133     cin.tie(nullptr);
134     int n;
135     cin >> n;
136     vector<int> p(n), q(n);
137     for (int i = 0; i < n; i++) {
138         cin >> p[i];
139         p[i]--;
140     }
141     for (int i = 0; i < n; i++) {
142         cin >> q[i];
143         q[i]--;
144     }
145     bool one = true;
146     for (int i = 0; i < n; i++) {
147         if (p[i] != i) {
148             one = false;
149         }
150         if (q[i] != i) {
151             one = false;
152         }
153     }
154     if (one) {
155         for (int i = 1; i <= n; i++) {
156             cout << n;
157             for (int j = 1; j <= n; j++) {
158                 cout << " " << j << " " << i;
159             }
160             cout << "\n";
161         }
162         for (int i = 1; i <= n; i++) {
163             cout << n;
164         }
```

```

165         for (int j = 1; j <= n; j++) {
166             cout << " " << i << " " << j;
167         }
168         cout << "\n";
169     }
170     return 0;
171 }
172 auto ans = solve(p, q);
173 for (auto v : ans) {
174     cout << v.size();
175     for (auto [x, y] : v) {
176         cout << " " << x + 1 << " " << y + 1;
177     }
178     cout << "\n";
179 }
180 return 0;
181 }
```

## 912: Diverse Coloring

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

In this problem, we will be working with rooted binary trees. A tree is called a rooted binary tree if it has a fixed root and every vertex has at most two children.

Let's assign a color - white or blue - to each vertex of the tree, and call this assignment a coloring of the tree. Let's call a coloring diverse if every vertex has a neighbor (a parent or a child) colored into an opposite color compared to this vertex. It can be shown that any tree with at least two vertices allows a diverse coloring.

Let's define the disbalance of a coloring as the absolute value of the difference between the number of white vertices and the number of blue vertices.

Now to the problem. Initially, the tree consists of a single vertex with the number 1 which is its root. Then, for each  $i$  from 2 to  $n$ , a new vertex  $i$  appears in the tree, and it becomes a child of vertex  $p_i$ . It is guaranteed that after each step the tree will keep being a binary tree rooted at vertex 1, that is, each vertex will have at most two children.

After every new vertex is added, print the smallest value of disbalance over all possible diverse colorings of the current tree. Moreover, after adding the last vertex with the number  $n$ , also print a diverse coloring with the smallest possible disbalance as well.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> p(n);
8     p[0] = -1;
9     mt19937 rng;
10    vector<int> d(n);
11    for (int i = 2; i <= n; i++) {
12        cin >> p[i - 1];
13        p[i - 1]--;
14        if (i == 4 && p[2] == 1 && p[3] == 1) {
15            cout << 2 << "\n";
16        } else {
17            cout << i % 2 << "\n";
18        }
19        // int x;
20        // do {
21        //     x = rng() % (i - 1);
22        // } while (d[x] >= 2);
23        // p[i - 1] = x;
24        // d[x]++;
25    }
26    vector<vector<int>> adj(n);
27    for (int i = 1; i < n; i++) {
28        adj[p[i]].push_back(i);
29        adj[i].push_back(p[i]);
30    }
31    vector<int> vis(n), color(n), bal(n), siz(n), deg(n), val(n);
32    auto dfs = [&](auto dfs, int x, int p) -> void {
33        siz[x] = 1;
34        deg[x] = (p != -1);
35        val[x] = 0;
36        bal[x] = 1;
37        for (auto y : adj[x]) {
38            if (y == p || vis[y]) continue;
39            deg[x]++;
40            color[y] = color[x] ^ 1;
41            dfs(dfs, y, x);
42            bal[x] -= bal[y];
43            siz[x] += siz[y];
44            val[x] = max(val[x], siz[y]);
45        }
46    };
47    int cnt = 0;
48    auto dfs1 = [&](auto dfs1, int x, int p) -> void {
49        vis[x] = ++cnt;
50        for (auto y : adj[x]) {
51            if (y == p || vis[y]) continue;
52            dfs1(dfs1, y, x);
53        }
54    };
55    auto find = [&](auto find, int x, int p, int s) -> int {
56        val[x] = max(val[x], s - siz[x]);
57        int res = deg[x] == 3 ? x : -1;
58        for (auto y : adj[x]) {
59            if (y == p || vis[y]) continue;
60            int v = find(find, y, x, s);
61            if (v != -1 && (res == -1 || val[v] < val[res])) res = v;
62        }
63    };

```

```

63         return res;
64     };
65     auto invert = [&](auto invert, int x, int p, int s) -> void {
66     color[x] ^= 1;
67     for (auto y : adj[x]) {
68         if (y == p || vis[y] <= s) continue;
69         invert(invert, y, x, s);
70     }
71 };
72 auto work = [&](auto work, int x) -> int {
73     dfs(dfs, x, -1);
74     int r = find(find, x, -1, siz[x]);
75     if (r == -1) {
76         dfs1(dfs1, x, -1);
77         return x;
78     }
79     // cerr << "work " << r + 1 << "\n";
80     dfs(dfs, r, -1);
81     bal[r] = 1;
82     vis[r] = ++cnt;
83     vector<pair<int, int>> ch;
84     int ok = 0;
85     for (auto y : adj[r]) {
86         int v = work(work, y);
87         if (siz[v] == 1) {
88             if (color[v] == color[r]) color[v] ^= 1;
89             bal[r] -= bal[v];
90             ok = 1;
91             continue;
92         }
93         if (bal[v] == -2) {
94             assert(siz[v] == 4);
95             color[y] ^= 1;
96             bal[v] = 0;
97             if (color[r] == color[y]) {
98                 invert(invert, v, -1, vis[r]);
99             }
100            ok = 1;
101            continue;
102        }
103        ch.emplace_back(v, y);
104    }
105    for (auto [v, y] : ch) {
106        if (bal[r] == 0 || bal[v] == 0) {
107            if (color[y] == color[r]) {
108                invert(invert, v, -1, vis[r]);
109            }
110        }
111        if (abs(bal[r] + (color[r] == color[v] ? bal[v] : -bal[v])) > 1) {
112            invert(invert, v, -1, vis[r]);
113        }
114        if (color[y] != color[r]) {
115            ok = 1;
116        }
117        bal[r] += color[r] == color[v] ? bal[v] : -bal[v];
118    }
119    assert(ok);
120    // cerr << r+1 << " " << siz[r] << " " << bal[r] << "\n";
121    return r;
122 };
123 work(work, 0);
124 for (int i = 0; i < n; i++) {
125     int ok = 0;
126     for (auto j : adj[i]) {

```

```

127         ok |= color[i] != color[j];
128     }
129     // if (!ok) {
130     // cerr << i+1 << " ";
131     // for (auto j : adj[i]) cerr << j+1 << " ";
132     // cerr << "\n";
133     // return;
134     // }
135 }
136 int res = 0;
137 for (int i = 0; i < n; i++) {
138     cout << "wb"[color[i]];
139     res += color[i];
140 }
141 cout << "\n";
142 // cerr << res << "\n";
143 }
144 int main() {
145     ios::sync_with_stdio(false);
146     cin.tie(nullptr);
147     int t;
148     cin >> t;
149     while (t--) {
150         solve();
151     }
152     return 0;
153 }
```

### 913: Unequal Adjacent Elements

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given an array  $a$  consisting of  $n$  positive integers.

Find any permutation  $p$  of  $[1, 2, \dots, n]$  such that:

$p_{i-2} < p_i$  for all  $i$ , where  $3 \leq i \leq n$ , and

$a_{p_{i-1}} \neq a_{p_i}$  for all  $i$ , where  $2 \leq i \leq n$ .

Or report that no such permutation exists.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
```

```

8     for (int i = 0; i < n; i++) {
9         cin >> a[i];
10        a[i]--;
11    }
12    vector<int> cnt(n);
13    for (int i = 0; i < n; i++) {
14        cnt[a[i]]++;
15    }
16    for (int i = 0; i < n; i++) {
17        if (cnt[i] > (n + 1) / 2) {
18            cout << "NO\n";
19            return;
20        }
21    }
22    cout << "YES\n";
23    set<pair<int, int>> s;
24    for (int i = 0; i < n; i++) {
25        s.emplace(cnt[i], i);
26    }
27    vector<vector<int>> seg;
28    vector<int> vis(n);
29    int res = 0;
30    for (int i = 0, j = 1; ; i++) {
31        if (vis[i]) continue;
32        if (j <= i) j = i + 1;
33        int mx = s.rbegin()->first;
34        if (mx == (n - res + 1) / 2) break;
35        while (vis[j] || a[i] == a[j]) j++;
36        if (i == res) seg.push_back({});
37        seg.back().push_back(j);
38        seg.back().push_back(i);
39        vis[i] = vis[j] = 1;
40        s.erase({cnt[a[i]], a[i]});
41        cnt[a[i]]--;
42        s.emplace(cnt[a[i]], a[i]);
43        s.erase({cnt[a[j]], a[j]});
44        cnt[a[j]]--;
45        s.emplace(cnt[a[j]], a[j]);
46        res += 2;
47    }
48    vector<int> mx, other;
49    int vmx = s.rbegin()->second;
50    for (int i = 0; i < n; i++) {
51        if (!vis[i]) {
52            if (vmx == a[i]) mx.push_back(i);
53            else other.push_back(i);
54        }
55    }
56    seg.push_back({});
57    for (int i = 0; i < mx.size(); i++) {
58        seg.back().push_back(mx[i]);
59        if (other.size() > i) seg.back().push_back(other[i]);
60    }
61    for (int i = seg.size() - 2; i >= 0; i--) {
62        if (a[seg[i].back()] == a[seg[i + 1][0]]) {
63            for (int j = 0; j < seg[i].size(); j += 2) {
64                swap(seg[i][j], seg[i][j + 1]);
65            }
66        }
67    }
68    vector<int> ans;
69    for (auto s : seg) {
70        ans.insert(ans.end(), s.begin(), s.end());
71    }

```

```

72     for (int i = 2; i < n; i++) {
73         assert(ans[i] > ans[i - 2]);
74     }
75     for (int i = 1; i < n; i++) {
76         assert(a[ans[i]] != a[ans[i - 1]]);
77     }
78     for (int i = 0; i < n; i++) {
79         cout << ans[i] + 1 << " \n"[i == n - 1];
80     }
81 }
82 int main() {
83     ios::sync_with_stdio(false);
84     cin.tie(nullptr);
85     int t;
86     cin >> t;
87     while (t--) {
88         solve();
89     }
90     return 0;
91 }
```

## 914: Dangerous Laser Power

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Pak Chanek has an  $n \times m$  grid of portals. The portal on the  $i$ -th row and  $j$ -th column is denoted as portal  $(i, j)$ . The portals  $(1, 1)$  and  $(n, m)$  are on the north-west and south-east corner of the grid respectively.

The portal  $(i, j)$  has two settings:

Type  $t_{i,j}$ , which is either 0 or 1.

Strength  $s_{i,j}$ , which is an integer between 1 and  $10^9$  inclusive.

When a laser enters face  $k$  of portal  $(i, j)$  with speed  $x_{\text{in}}$ , it leaves the portal going out of face  $(k + 2 + t_{i,j}) \bmod 4$  with speed  $x_{\text{out}} = \max(x_{\text{in}}, s_{i,j})$ . The portal also has to consume  $x_{\text{out}} - x_{\text{in}}$  units of energy.

Pak Chanek is very bored today. He will shoot  $4nm$  lasers with an initial speed of 1, one into each face of each portal. Each laser will travel throughout this grid of portals until it moves outside the grid or it has passed through  $10^{100}$  portals.

At the end, Pak Chanek thinks that a portal is good if and only if the total energy consumed by that portal modulo 2 is equal to its type. Given the strength settings of all portals, find a way to assign the type settings of each portal such that the number of good portals is maximised.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct SEQ;
5 void merge(SEQ &a, SEQ &b, int x) {
6     int c = 0;
7     x = max(x, b.x.front()[0]);
8     while (!a.x.empty() && a.x.back()[0] <= x) {
9         c ^= a.x.back()[1];
10        a.sum ^= a.x.back()[0] & a.x.back()[1];
11        a.x.pop_back();
12    }
13    b.x.push_front({x, c});
14    b.sum ^= x & c;
15    a.x.splice(a.x.end(), b.x);
16    a.cnt ^= b.cnt;
17    a.sum ^= b.sum;
18 }
19 # struct DSU;
20 const int dx[] = {1, 0, -1, 0};
21 const int dy[] = {0, -1, 0, 1};
22 int main() {
23     ios::sync_with_stdio(false);
24     cin.tie(nullptr);
25     int n, m;
26     cin >> n >> m;
27     vector a(n + 2, vector<int>(m + 2));
28     vector<array<int, 2>> p;
29     const int N = 4 * (n + 2) * (m + 2);
30     DSU dsu(N);
31     vector<SEQ> s(N);
32     for (int i = 0; i < N; i++) {
33         s[i].x.push_back({1, 1});
34         s[i].cnt = s[i].sum = 1;
35     }
36     for (int i = 1; i <= n; i++) {
37         for (int j = 1; j <= m; j++) {
38             cin >> a[i][j];
39             p.push_back({i, j});
40         }
41     }
42     sort(p.begin(), p.end(), [&](auto i, auto j) {
43         return a[i[0]][i[1]] < a[j[0]][j[1]];
44     });
45     auto get = [&](int x, int y, int d) {
46         return 4 * (x * (m + 2) + y) + d;
47     };
48     vector ans(n + 2, vector<int>(m + 2));
49     for (auto [x, y] : p) {
50         int res = 0;
51         for (int i = 0; i < 4; i++) {
52             int u = get(x, y, i);
53             u = dsu.leader(u);
54             res ^= s[u].sum ^ (s[u].cnt & a[x][y]);
55         }
56         ans[x][y] = res;
57         for (int i = 0; i < 4; i++) {
58             int ni = (i + res) & 3;

```

```

59         if (!dsu.same(get(x, y, i), get(x + dx[ni], y + dy[ni], ni))) {
60             int u = dsu.leader(get(x, y, i));
61             int v = dsu.leader(get(x + dx[ni], y + dy[ni], ni));
62             merge(s[u], s[v], a[x][y]);
63             dsu.merge(u, v);
64         }
65     }
66 }
67 for (int i = 1; i <= n; i++) {
68     for (int j = 1; j <= m; j++) {
69         cout << ans[i][j];
70     }
71     cout << "\n";
72 }
73 return 0;
74 }
```

## data structures

### 915: Parallel Swaps Sort

- Time limit: 7 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

You are given a permutation  $p_1, p_2, \dots, p_n$  of  $[1, 2, \dots, n]$ . You can perform the following operation some (possibly 0) times:

choose a subarray  $[l, r]$  of even length;

swap  $a_l, a_{l+1}$ ;

swap  $a_{l+2}, a_{l+3}$  (if  $l + 3 \leq r$ );

...

swap  $a_{r-1}, a_r$ .

Sort the permutation in at most  $10^6$  operations. You do not need to minimize the number of operations.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
```

```
8      cin >> n;
9      vector<int> p(n);
10     for (int i = 0; i < n; i++) {
11         cin >> p[i];
12         p[i]--;
13     }
14     vector<array<int, 2>> out;
15     vector<int> first(n), second(n);
16     iota(first.begin(), first.end(), 0);
17     vector<int> invp(n);
18     for (int i = 0; i < n; i++) {
19         invp[p[i]] = i;
20     }
21     for (int i = 0; i < n; i++) {
22         int j = i;
23         while (p[p[j]] != j) {
24             swap(invp[p[j]], invp[p[p[j]]]);
25             swap(first[j], first[p[j]]);
26             swap(p[j], p[p[j]]);
27             j = invp[j];
28         }
29     }
30     second = p;
31     auto work = [&](auto f, int t) {
32         vector<vector<int>> vec(n);
33         for (int i = 0; i < n; i++) {
34             if (f[i] != i) {
35                 int x = i, y = f[i];
36                 if (t == 1) {
37                     x = n - 1 - x;
38                     y = n - 1 - y;
39                 }
40                 if (x > y) {
41                     swap(x, y);
42                 }
43                 int turn = 0, pos = 0;
44                 if ((y - x) % 2 == 1) {
45                     turn = (y - x + 1) / 2;
46                 } else {
47                     turn = (2 * n - y - x) / 2;
48                 }
49                 if (x % 2 == 1) {
50                     turn = n + 1 - turn;
51                 }
52                 pos = turn;
53                 if (x % 2 == 0) {
54                     pos += x;
55                 } else {
56                     pos -= x + 1;
57                 }
58                 vec[turn - 1].push_back(pos - 1);
59                 swap(f[i], f[f[i]]);
60             }
61         }
62         for (int i = 0; i < n; i++) {
63             int l = i % 2;
64             int r = n;
65             if ((r - l) % 2) {
66                 r--;
67             }
68             if (l < r) {
69                 out.push_back({l, r});
70             }
71             for (auto j : vec[i]) {
```

```

72             out.push_back({j, j + 2});
73         }
74     }
75 }
76 work(first, 0);
77 work(second, 1);
78 cout << out.size() << "\n";
79 for (auto [l, r] : out) {
80     cout << l + 1 << " " << r << "\n";
81 }
82 return 0;
83 }
```

## 916: Diamond Theft

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Monocarp is the most famous thief in Berland. This time, he decided to steal two diamonds. Unfortunately for Monocarp, there are  $n$  cameras monitoring the diamonds. Each camera has two parameters,  $t_i$  and  $s_i$ . The first parameter determines whether the camera is monitoring the first diamond only ( $t_i = 1$ ), the second diamond only ( $t_i = 2$ ), or both diamonds ( $t_i = 3$ ). The second parameter determines the number of seconds the camera will be disabled after it is hacked.

Every second, Monocarp can perform one of the following three actions:

do nothing;

choose a camera and hack it; if Monocarp hacks the  $i$ -th camera, it will be disabled for the next  $s_i$  seconds (if the current second is the  $T$ -th one, the camera will be disabled from the  $(T + 1)$ -th to the  $(T + s_i)$ -th second, inclusive);

steal a diamond if all cameras monitoring it are currently disabled. Monocarp cannot steal the second diamond if he hasn't stolen the first diamond yet.

Note that Monocarp can hack a camera multiple times, even if it is currently disabled.

Your task is to determine the minimum time it will take Monocarp to steal both diamonds, beginning with the first diamond, or report that it is impossible.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
6 int main() {
7     ios::sync_with_stdio(false);
8     cin.tie(nullptr);
9     int n;
10    cin >> n;
11    vector<int> a[4];
12    for (int i = 0; i < n; i++) {
13        int t, s;
14        cin >> t >> s;
15        a[t].push_back(s);
16    }
17    for (int t = 1; t <= 3; t++) {
18        sort(a[t].begin(), a[t].end());
19    }
20    int ans = 2 * n + 3;
21    vector<int> p(n + 1);
22    iota(p.begin(), p.end(), 1);
23    shuffle(p.begin(), p.end(), rng);
24    auto check = [&](int sum, int t2) {
25        int t1 = sum - t2;
26        if (t1 < 1) {
27            return false;
28        }
29        DSU d1(t1), d2(t2);
30        for (auto s : a[1]) {
31            int x = max(0, t1 - 1 - s);
32            x = d1.find(x);
33            if (x == t1 - 1) {
34                return false;
35            }
36            d1.merge(x + 1, x);
37        }
38        for (auto s : a[3]) {
39            if (s > t2) {
40                int x = max(0, t1 - 1 - (s - t2));
41                x = d1.find(x);
42                if (x < t1 - 1) {
43                    d1.merge(x + 1, x);
44                    continue;
45                }
46            }
47            int x = max(0, t1 - 1 - s);
48            x = d1.find(x);
49            if (x == t1 - 1) {
50                return false;
51            }
52            d1.merge(x + 1, x);
53            x = max(0, t2 - 1 - s);
54            x = d2.find(x);
55            if (x == t2 - 1) {
56                return false;
57            }
58            d2.merge(x + 1, x);
59        }
60        for (auto s : a[2]) {
61            if (s > t2) {
62                int x = max(0, t1 - 1 - (s - t2));
63                x = d1.find(x);
64                if (x < t1 - 1) {
```

```

65                     d1.merge(x + 1, x);
66                     continue;
67                 }
68             }
69             int x = max(0, t2 - 1 - s);
70             x = d2.find(x);
71             if (x == t2 - 1) {
72                 return false;
73             }
74             d2.merge(x + 1, x);
75         }
76     return true;
77 };
78 for (auto t2 : p) {
79     if (!check(ans - 1, t2)) {
80         continue;
81     }
82     ans = *ranges::partition_point(ranges::iota_view(1, ans),
83                                     [&](int s) {
84                                         return !check(s, t2);
85                                     });
86 }
87 if (ans > 2 * n + 2) {
88     ans = -1;
89 }
90 cout << ans << "\n";
91 return 0;
92 }
```

## 917: Clubstep

- Time limit: 3 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

There is an extremely hard video game that is one of Chaneka's favourite video games. One of the hardest levels in the game is called Clubstep. Clubstep consists of  $n$  parts, numbered from 1 to  $n$ . Chaneka has practised the level a good amount, so currently, her familiarity value with each part  $i$  is  $a_i$ .

After this, Chaneka can do several (possibly zero) attempts on Clubstep. In each attempt, she dies on one of the  $n$  parts. If an attempt dies on part  $p$ , that means it only successfully passes through every part  $k$  for all  $1 \leq k \leq p - 1$  and it does not reach any part  $k$  for all  $p + 1 \leq k \leq n$ . An attempt that dies on part  $p$  takes  $p$  seconds.

It is known that Chaneka improves much more on the part she dies on than anything else. It is also known that during an attempt, Chaneka does not get to practise that much on the parts she does not reach. So, the effect of an attempt that dies on part  $p$  is as follows:

Chaneka's familiarity value with part  $p$  increases by 2.

Chaneka's familiarity value with each part  $k$  for all  $1 \leq k \leq p - 1$  increases by 1.

Note that each question is independent, so the attempt Chaneka does on a question does not affect the familiarity values of any other questions.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<int> a(n);
10    for (int i = 0; i < n; i++) {
11        cin >> a[i];
12    }
13    int q;
14    cin >> q;
15    vector<int> f(2 * q);
16    vector<i64> val(2 * q);
17    int tot = q;
18    iota(f.begin(), f.end(), 0);
19    vector<i64> ans(q);
20    vector<vector<pair<int, int>>> add(n);
21    vector<vector<int>> del(n);
22    for (int i = 0; i < q; i++) {
23        int l, r, x;
24        cin >> l >> r >> x;
25        l--, r--;
26        add[r].emplace_back(x, i);
27        del[l].push_back(i);
28    }
29    auto find = [&](auto self, int x) -> int {
30        if (f[x] == f[f[x]]) {
31            return f[x];
32        }
33        auto y = self(self, f[x]);
34        val[x] += val[f[x]];
35        f[x] = y;
36        return f[x];
37    };
38    auto get = [&](i64 x) {
39        find(find, x);
40        i64 ans = val[x];
41        if (x != f[x]) {
42            ans += val[f[x]];
43        }
44        return ans;
45    };
46    auto merge = [&](int &x, int y) {
47        int z = tot++;
48        f[x] = z;
49        f[y] = z;
50        x = z;
51    };
52    map<int, int> mp;
53    for (int i = n - 1; i >= 0; i--) {
54        for (auto [x, j] : add[i]) {

```

```

55         if (mp.contains(x)) {
56             merge(mp[x], j);
57         } else {
58             mp[x] = j;
59         }
60     }
61     for (auto it = mp.upper_bound(a[i]); it != mp.end(); it = mp.erase(it)) {
62         auto [x, j] = *it;
63         int y = (x + a[i]) / 2;
64         val[j] += 1LL * (i + 1) * (x - y);
65         if (mp.contains(y)) {
66             merge(mp[y], j);
67         } else {
68             mp[y] = j;
69         }
70     }
71     for (auto j : del[i]) {
72         ans[j] = get(j);
73     }
74 }
75 for (int i = 0; i < q; i++) {
76     cout << ans[i] << "\n";
77 }
78 return 0;
79 }
```

## 918: Standard Graph Problem

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given a weighted directed graph with  $n$  vertices and  $m$  edges. Each vertex in the graph can be either highlighted or normal. Initially, all vertices are normal. The cost of the graph is defined as the minimum sum of edge weights that need to be selected so that from each normal vertex one can reach at least one highlighted vertex using the selected edges only. If it is not possible to select the edges, the cost is  $-1$  instead.

Your task is to compute the cost of the graph after each of the  $q$  queries. The queries can be of two types:

+  $v_i$  makes vertex  $v_i$  highlighted; it is guaranteed that the vertex is normal before the query.

-  $v_i$  makes vertex  $v_i$  normal; it is guaranteed that the vertex is highlighted before the query.

Output the cost of the graph after each of the  $q$  queries.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 constexpr i64 inf = 1E12;
6 # struct Node;
7 void add(Node *a, i64 v) {
8     if (a) {
9         a->v += v;
10        a->tag += v;
11    }
12 }
13 void push(Node *a) {
14     add(a->l, a->tag);
15     add(a->r, a->tag);
16     a->tag = 0;
17 }
18 Node *merge(Node *a, Node *b) {
19     if (!a) {
20         return b;
21     }
22     if (!b) {
23         return a;
24     }
25     if (a->v > b->v) {
26         swap(a, b);
27     }
28     push(a);
29     push(b);
30     a->r = merge(a->r, b);
31     if (!a->l || a->l->h < a->r->h) {
32         swap(a->l, a->r);
33     }
34     a->h = a->r ? a->r->h + 1 : 0;
35     return a;
36 }
37 Node *pop(Node *a) {
38     push(a);
39     return merge(a->l, a->r);
40 }
41 # struct HLD;
42 int main() {
43     ios::sync_with_stdio(false);
44     cin.tie(nullptr);
45     int n, m, q;
46     cin >> n >> m >> q;
47     DSU dsu(2 * n - 1);
48     int cur = n;
49     vector<Node *> e(2 * n - 1);
50     vector<bool> inq(2 * n - 1);
51     for (int i = 0; i < m; i++) {
52         int u, v, c;
53         cin >> u >> v >> c;
54         u--, v--;
55         e[u] = merge(e[u], new Node{c, v});
56     }
57     for (int i = 0; i < n; i++) {
58         e[i] = merge(e[i], new Node{inf, (i + 1) % n});
59     }
60     vector<int> stk;
61     stk.push_back(0);
62     inq[0] = true;
63     vector<i64> w(2 * n - 1);
64     HLD t(2 * n - 1);
```

```

65     while (true) {
66         int x = stk.back();
67         while (e[x] && dsu.same(x, e[x]->to)) {
68             e[x] = pop(e[x]);
69         }
70         if (!e[x]) {
71             break;
72         }
73         w[x] = e[x]->v;
74         int y = dsu.find(e[x]->to);
75         if (inq[y]) {
76             int nv = cur++;
77             while (true) {
78                 int v = stk.back();
79                 stk.pop_back();
80                 add(e[v], -e[v]->v);
81                 t.addEdge(nv, v);
82                 dsu.merge(nv, v);
83                 e[nv] = merge(e[nv], e[v]);
84                 if (v == y) {
85                     break;
86                 }
87             }
88             stk.push_back(nv);
89             inq[nv] = true;
90         } else {
91             stk.push_back(y);
92             inq[y] = true;
93         }
94     }
95     int rt = cur - 1;
96     t.work(rt);
97     auto cmp = [&](int i, int j) {
98         return t.in[i] < t.in[j];
99     };
100    i64 ans = accumulate(w.begin(), w.end(), 0LL);
101    set<int, decltype(cmp)> s(cmp);
102    for (int i = rt - 1; i >= 0; i--) {
103        w[i] += w[t.parent[i]];
104    }
105    while (q--) {
106        char o;
107        int x;
108        cin >> o >> x;
109        x--;
110        if (o == '+') {
111            auto it = s.insert(x).first;
112            auto r = next(it);
113            ans -= w[x];
114            if (it != s.begin()) {
115                ans += w[t.lca(x, *prev(it))];
116            }
117            if (r != s.end()) {
118                ans += w[t.lca(x, *r)];
119            }
120            if (it != s.begin() && r != s.end()) {
121                ans -= w[t.lca(*prev(it), *r)];
122            }
123        } else {
124            auto it = s.find(x);
125            auto r = next(it);
126            ans += w[x];
127            if (it != s.begin()) {
128                ans -= w[t.lca(x, *prev(it))];

```

```

129         }
130         if (r != s.end()) {
131             ans -= w[t.lca(x, *r)];
132         }
133         if (it != s.begin() && r != s.end()) {
134             ans += w[t.lca(*prev(it), *r)];
135         }
136         s.erase(it);
137     }
138     cout << (ans >= inf ? -1 : ans) << "\n";
139 }
140 return 0;
141 }
```

## 919: MEXanization

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Let's define  $f(S)$ . Let  $S$  be a multiset (i.e., it can contain repeated elements) of non-negative integers. In one operation, you can choose any non-empty subset of  $S$  (which can also contain repeated elements), remove this subset (all elements in it) from  $S$ , and add the MEX of the removed subset to  $S$ . You can perform any number of such operations. After all the operations,  $S$  should contain exactly 1 number.  $f(S)$  is the largest number that could remain in  $S$  after any sequence of operations.

You are given an array of non-negative integers  $a$  of length  $n$ . For each of its  $n$  prefixes, calculate  $f(S)$  if  $S$  is the corresponding prefix (for the  $i$ -th prefix,  $S$  consists of the first  $i$  elements of array  $a$ ).

The MEX (minimum excluded) of an array is the smallest non-negative integer that does not belong to the array. For instance:

The MEX of  $[2, 2, 1]$  is 0, because 0 does not belong to the array.

The MEX of  $[3, 1, 0, 1]$  is 2, because 0 and 1 belong to the array, but 2 does not.

The MEX of  $[0, 3, 1, 2]$  is 4, because 0, 1, 2 and 3 belong to the array, but 4 does not.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int B = 400;
5 # struct DSU;
6 DSU dsu[B];
```

```
7 void solve() {
8     int n;
9     cin >> n;
10    vector<int> a(n);
11    for (int i = 0; i < n; i++) {
12        cin >> a[i];
13    }
14    vector<int> ans(n);
15    vector<int> cnt(n + 1);
16    vector<int> pre(n + 1);
17    int nb = (n + B) / B;
18    vector<int> pb(nb);
19    auto get = [&](int n) {
20        if (n < 0) {
21            return 0;
22        }
23        return pre[n] + (n < B ? 0 : pb[n / B - 1]);
24    };
25    for (int i = 1; i < B; i++) {
26        dsu[i].init(n + 1);
27    }
28    auto check = [&](int t) {
29        if (t > n) {
30            return false;
31        }
32        int need = 1;
33        int c0 = get(n) - get(t - 1);
34        for (int i = t - 1; i > 0; i--) {
35            if (need >= B) {
36                if (cnt[i] >= need) {
37                    c0 += cnt[i] - need;
38                } else {
39                    need = 2 * need - cnt[i];
40                }
41            } else {
42                int j = dsu[need].find(i);
43                c0 += get(i) - get(j) - (i - j) * need;
44                if (j) {
45                    need = 2 * need - cnt[j];
46                }
47                i = j;
48            }
49            if (need > n) {
50                return false;
51            }
52        }
53        return need <= c0 + cnt[0];
54    };
55    for (int i = 0, t = 1; i < n; i++) {
56        int x = min(n, a[i]);
57        int b = x / B;
58        cnt[x]++;
59        if (x && cnt[x] < B) {
60            dsu[cnt[x]].merge(x - 1, x);
61        }
62        for (int j = x; j <= min(n, b * B + B - 1); j++) {
63            pre[j]++;
64        }
65        for (int j = b; j < nb; j++) {
66            pb[j]++;
67        }
68        while (check(t)) {
69            t++;
70        }
71    }
72}
```

```

71         ans[i] = t - 1;
72     }
73     ans[0] = max(1, a[0]);
74     for (int i = 0; i < n; i++) {
75         cout << ans[i] << " \n"[i == n - 1];
76     }
77 }
78 int main() {
79     ios::sync_with_stdio(false);
80     cin.tie(nullptr);
81     int t;
82     cin >> t;
83     while (t--) {
84         solve();
85     }
86     return 0;
87 }
```

## 920: Freak Joker Process

- Time limit: 5 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

After the success of the basketball teams formed and trained by Pak Chanek last year (Basketball Together), Pak Chanek wants to measure the performance of each player that is considered as a superstar.

There are  $N$  superstar players that have been trained by Pak Chanek. At the end of the season, some calculations will be made on the performance of the  $N$  players using an international method. Each player has two values  $A_i$  and  $B_i$  where each represents the offensive and defensive value of that player.

Define  $\text{RankA}(i)$  as the offensive ranking of the  $i$ -th player, whose value is  $c + 1$  with  $c$  here representing the number of  $j$  ( $1 \leq j \leq N$ ) such that  $A_j > A_i$ . Define  $\text{RankB}(i)$  as the defensive ranking of the  $i$ -th player, whose value is  $c + 1$  with  $c$  here representing the number of  $j$  ( $1 \leq j \leq N$ ) such that  $B_j > B_i$ .

Define  $\text{RankOverall}(i)$  as the overall ranking of the  $i$ -th player, whose value is  $c + 1$  with  $c$  here representing the number of  $j$  ( $1 \leq j \leq N$ ) such that  $\text{RankA}(j) + \text{RankB}(j) < \text{RankA}(i) + \text{RankB}(i)$ .

During the next  $Q$  days, exactly one event will happen on each day. Each event is one of the three following possibilities:

1 k c - If  $c$  is +, then  $A_k$  increases by 1. If  $c$  is -, then  $A_k$  decreases by 1. ( $1 \leq k \leq N$ ;  $c$  is + or -)

2 k c - If  $c$  is +, then  $B_k$  increases by 1. If  $c$  is -, then  $B_k$  decreases by 1. ( $1 \leq k \leq N$ ;  $c$  is + or -)

3 k - Pak Chanek wants to know the value of  $\text{RankOverall}(k)$  at that moment. ( $1 \leq k \leq N$ )

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template <typename T>
5 # struct Fenwick;
6 constexpr int V = 1E5;
7 constexpr int T = 500;
8 # struct Node;
9 constexpr int L = 4E6;
10 Node t[L];
11 vector<int> stk;
12 int tot = 0;
13 int newNode() {
14     int x;
15     if (stk.empty()) {
16         x = ++tot;
17     } else {
18         x = stk.back();
19         stk.pop_back();
20     }
21     t[x] = {};
22     return x;
23 }
24 void addt(int &p, int l, int r, int x, int y) {
25     if (!p) {
26         p = newNode();
27     }
28     t[p].cnt += y;
29     if (r - l > 1) {
30         int m = (l + r) / 2;
31         if (x < m) {
32             addt(t[p].l, l, m, x, y);
33         } else {
34             addt(t[p].r, m, r, x, y);
35         }
36     }
37     if (!t[p].cnt) {
38         stk.push_back(p);
39         p = 0;
40     }
41 }
42 int sumt(int p, int l, int r, int x, int y) {
43     if (!p || l >= y || r <= x) {
44         return 0;
45     }
46     if (l >= x && r <= y) {
47         return t[p].cnt;
48     }
49     int m = (l + r) / 2;
50     return sumt(t[p].l, l, m, x, y) + sumt(t[p].r, m, r, x, y);
51 }
52 int main() {
53     ios::sync_with_stdio(false);
54     cin.tie(nullptr);
55     int N;
56     cin >> N;
57     vector<int> A(N), B(N);
58     for (int i = 0; i < N; i++) {
59         cin >> A[i];
60         A[i] = V - A[i];
```

```

61     }
62     for (int i = 0; i < N; i++) {
63         cin >> B[i];
64         B[i] = V - B[i];
65     }
66     vector<set<int>> SA(V), SB(V);
67     Fenwick<int> fenA(V), fenB(V), fenr(2 * N);
68     for (int i = 0; i < N; i++) {
69         fenA.add(A[i], 1);
70         fenB.add(B[i], 1);
71     }
72     auto get = [&](int k) {
73         return fenA.sum(A[k]) + fenB.sum(B[k]);
74     };
75     vector<int> ina(V, 1), inb(V, 1);
76     set<int> oa, ob;
77     vector<int> ta(V), tb(V);
78     auto query = [&](int k) {
79         // int ans_ = 0;
80         // for (int i = 0; i < N; i++) {
81         //     ans_ += get(i) < get(k);
82         // }
83         // return ans_ + 1;
84         int s = get(k);
85         int ans = fenr.sum(s) + 1;
86         for (auto a : oa) {
87             assert(!ina[a]);
88             int rka = fenA.sum(a);
89             if (rka >= s) {
90                 break;
91             }
92             int mxb = fenB.kth(s - rka - 1) + 1;
93             ans += sumt(ta[a], 0, V, 0, mxb);
94             // for (auto i : SA[a]) {
95             //     // ans += B[i] < mxb;
96             //     ans += get(i) < get(k);
97             // }
98         }
99         for (auto b : ob) {
100            // cerr << "b : " << b << " " << inb[b] << "\n";
101            assert(!inb[b]);
102            int rkb = fenB.sum(b);
103            if (rkb >= s) {
104                break;
105            }
106            int mxa = fenA.kth(s - rkb - 1) + 1;
107            ans += sumt(tb[b], 0, V, 0, mxa);
108            // for (auto i : SB[b]) {
109            //     if (ina[A[i]]) {
110            //         // ans += A[i] < mxa;
111            //         ans += get(i) < get(k);
112            //     }
113            // }
114        }
115        return ans;
116    };
117    auto add = [&](int k) {
118        int a = A[k], b = B[k];
119        SA[a].insert(k);
120        SB[b].insert(k);
121        if (ina[a] && inb[b]) {
122            fenr.add(get(k), 1);
123        }
124        addt(ta[a], 0, V, b, 1);

```

```

125         if (ina[a]) {
126             addt(tb[b], 0, v, a, 1);
127         }
128         if (SA[a].size() > T && ina[a]) {
129             assert(!oa.count(a));
130             oa.insert(a);
131             ina[a] = 0;
132             for (auto i : SA[a]) {
133                 addt(tb[B[i]], 0, v, a, -1);
134                 if (inb[B[i]]) {
135                     fenr.add(get(i), -1);
136                 }
137             }
138         }
139         if (SB[b].size() > T && inb[b]) {
140             assert(!ob.count(b));
141             ob.insert(b);
142             inb[b] = 0;
143             for (auto i : SB[b]) {
144                 if (ina[A[i]]) {
145                     fenr.add(get(i), -1);
146                 }
147             }
148         }
149     };
150     auto del = [&](int k) {
151         int a = A[k], b = B[k];
152         SA[a].erase(k);
153         SB[b].erase(k);
154         if (ina[a] && inb[b]) {
155             fenr.add(get(k), -1);
156         }
157         addt(ta[a], 0, v, b, -1);
158         if (ina[a]) {
159             addt(tb[b], 0, v, a, -1);
160         }
161         if (SA[a].size() <= T && !ina[a]) {
162             assert(oa.count(a));
163             oa.erase(a);
164             ina[a] = 1;
165             for (auto i : SA[a]) {
166                 addt(tb[B[i]], 0, v, a, 1);
167                 if (inb[B[i]]) {
168                     fenr.add(get(i), 1);
169                 }
170             }
171         }
172         if (SB[b].size() <= T && !inb[b]) {
173             assert(ob.count(b));
174             ob.erase(b);
175             inb[b] = 1;
176             for (auto i : SB[b]) {
177                 if (ina[A[i]]) {
178                     fenr.add(get(i), 1);
179                 }
180             }
181         }
182     };
183     for (int i = 0; i < N; i++) {
184         add(i);
185     }
186     int Q;
187     cin >> Q;
188     for (int _ = 0; _ < Q; _++) {

```

```
189     int o;
190     cin >> o;
191     if (o == 1) {
192         int k;
193         char c;
194         cin >> k >> c;
195         k--;
196         del(k);
197         fenA.add(A[k], -1);
198         if (c == '-') {
199             A[k]++;
200             if (ina[A[k]]) {
201                 for (auto i : SA[A[k]]) {
202                     if (inb[B[i]]) {
203                         int s = get(i);
204                         fenr.add(s + 1, -1);
205                         fenr.add(s, 1);
206                     }
207                 }
208             } else {
209                 if (ina[A[k]]) {
210                     for (auto i : SA[A[k]]) {
211                         if (inb[B[i]]) {
212                             int s = get(i);
213                             fenr.add(s, -1);
214                             fenr.add(s + 1, 1);
215                         }
216                     }
217                 }
218                 A[k]--;
219             }
220             fenA.add(A[k], 1);
221             add(k);
222         } else if (o == 2) {
223             int k;
224             char c;
225             cin >> k >> c;
226             k--;
227             del(k);
228             fenB.add(B[k], -1);
229             if (c == '-') {
230                 B[k]++;
231                 if (inb[B[k]]) {
232                     for (auto i : SB[B[k]]) {
233                         if (ina[A[i]]) {
234                             int s = get(i);
235                             fenr.add(s + 1, -1);
236                             fenr.add(s, 1);
237                         }
238                     }
239                 }
240             } else {
241                 if (inb[B[k]]) {
242                     for (auto i : SB[B[k]]) {
243                         if (ina[A[i]]) {
244                             int s = get(i);
245                             fenr.add(s, -1);
246                             fenr.add(s + 1, 1);
247                         }
248                     }
249                 }
250             B[k]--;
251         }
252     }
```

```

253         fenB.add(B[k], 1);
254         add(k);
255     } else {
256         int k;
257         cin >> k;
258         k--;
259         cout << query(k) << "\n";
260     }
261     // for (int i = 0; i < N; i++) {
262     //     cerr << get(i) << " ";
263     // }
264     // cerr << "| ";
265     // for (int i = 0; i < 10; i++) {
266     //     cerr << fenr.rangeSum(i, i + 1) << " ";
267     // }
268     // cerr << "\n";
269 }
270 return 0;
271 }
```

## 921: Welcome home, Chtholly

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output
- I... I survived.
- Welcome home, Chtholly.
- I kept my promise...
- I made it... I really made it!

After several days of fighting, Chtholly Nota Seniorious miraculously returned from the fierce battle.

As promised, Willem is now baking butter cake for her.

However, although Willem is skilled in making dessert, he rarely bakes butter cake.

This time, Willem made a big mistake - he accidentally broke the oven!

Fortunately, Chtholly decided to help him.

Willem puts  $n$  cakes on a roll, cakes are numbered from 1 to  $n$ , the  $i$ -th cake needs  $a_i$  seconds of baking.

Willem needs Chtholly to do  $m$  operations to bake the cakes.

Operation 1:  $l \ r \ x$

Willem asks Chtholly to check each cake in the range  $[l, r]$ , if the cake needs to be baked for more than  $x$  seconds, he would bake it for  $x$  seconds and put it back in its place. More precisely, for every  $i$  in range  $[l, r]$ , if  $a_i$  is strictly more than  $x$ ,  $a_i$  becomes equal  $a_i - x$ .

Operation 2:  $2 \mid r \mid x$

Willem asks Chtholly to count the number of cakes in the range  $[l, r]$  that needs to be cooked for exactly  $x$  seconds. More formally you should find number of such  $i$  in range  $[l, r]$ , that  $a_i = x$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, m;
9     cin >> n >> m;
10    vector<int> a(n);
11    for (int i = 0; i < n; i++) {
12        cin >> a[i];
13    }
14    const int B = sqrt(n);
15    const int nb = (n - 1 + B) / B;
16    vector<int> max(nb), add(nb);
17    constexpr int N = 1E5 + 1;
18    vector<int> rt(nb, vector<int>(N, -1));
19    DSU dsu(n);
20    auto build = [&](int b) {
21        int l = b * B, r = min(n, l + B);
22        for (int i = l; i < r; i++) {
23            if (rt[b][a[i]] == -1) {
24                rt[b][a[i]] = i;
25            } else {
26                dsu.merge(rt[b][a[i]], i);
27            }
28            max[b] = max(max[b], a[i]);
29        }
30    };
31    auto init = [&](int b) {
32        int l = b * B, r = min(n, l + B);
33        for (int i = l; i < r; i++) {
34            a[i] = a[dsu.find(i)];
35        }
36        for (int i = l; i < r; i++) {
37            rt[b][a[i]] = -1;
38            dsu.f[i] = i;
39            dsu.siz[i] = 1;
40        }
41        for (int i = l; i < r; i++) {
42            a[i] -= add[b];
43        }
44        add[b] = 0;
45        max[b] = 0;
46    };
47    for (int i = 0; i < nb; i++) {
48        build(i);

```

```
49     }
50     for (int i = 0; i < m; i++) {
51         int o, l, r, x;
52         cin >> o >> l >> r >> x;
53         l--;
54         r--;
55         int lb = l / B, rb = r / B;
56         if (lb == rb) {
57             init(lb);
58         } else {
59             init(lb);
60             init(rb);
61         }
62         if (o == 1) {
63             if (lb == rb) {
64                 for (int i = l; i <= r; i++) {
65                     if (a[i] > x) {
66                         a[i] -= x;
67                     }
68                 }
69             } else {
70                 for (int i = l; i < (lb + 1) * B; i++) {
71                     if (a[i] > x) {
72                         a[i] -= x;
73                     }
74                 }
75                 for (int b = lb + 1; b < rb; b++) {
76                     if (2 * x <= max[b]) {
77                         for (int i = 1; i <= x; i++) {
78                             if (rt[b][i + add[b]] != -1) {
79                                 if (rt[b][i + add[b] + x] == -1) {
80                                     a[rt[b][i + add[b]]] += x;
81                                     swap(rt[b][i + add[b]], rt[b][i + add[b] + x]);
82                                 } else {
83                                     dsu.merge(rt[b][i + add[b] + x], rt[b][i + add[b]]);
84                                     rt[b][i + add[b]] = -1;
85                                 }
86                             }
87                         }
88                         add[b] += x;
89                         max[b] -= x;
90                     } else if (x <= max[b]) {
91                         for (int i = x + 1; i <= max[b]; i++) {
92                             if (rt[b][i + add[b]] != -1) {
93                                 if (rt[b][i + add[b] - x] == -1) {
94                                     a[rt[b][i + add[b]]] -= x;
95                                     swap(rt[b][i + add[b]], rt[b][i + add[b] - x]);
96                                 } else {
97                                     dsu.merge(rt[b][i + add[b] - x], rt[b][i + add[b]]);
98                                     rt[b][i + add[b]] = -1;
99                                 }
100                            }
101                        }
102                        max[b] = x;
103                    }
104                }
105                for (int i = rb * B; i <= r; i++) {
106                    if (a[i] > x) {
107                        a[i] -= x;
108                    }
109                }
110            }
111        } else {
112            int ans = 0;
```

```

113         if (lb == rb) {
114             for (int i = l; i <= r; i++) {
115                 ans += (a[i] == x);
116             }
117         } else {
118             for (int i = l; i < (lb + 1) * B; i++) {
119                 ans += (a[i] == x);
120             }
121             for (int b = lb + 1; b < rb; b++) {
122                 if (x <= max[b]) {
123                     if (rt[b][x + add[b]] != -1) {
124                         ans += dsu.size(rt[b][x + add[b]]);
125                     }
126                 }
127             }
128             for (int i = rb * B; i <= r; i++) {
129                 ans += (a[i] == x);
130             }
131         }
132         cout << ans << "\n";
133     }
134     if (lb == rb) {
135         build(lb);
136     } else {
137         build(lb);
138         build(rb);
139     }
140 }
141 return 0;
142 }
```

## 922: Rivalries

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Ntarsis has an array  $a$  of length  $n$ .

The power of a subarray  $a_l \dots a_r$  ( $1 \leq l \leq r \leq n$ ) is defined as:

The largest value  $x$  such that  $a_l \dots a_r$  contains  $x$  and neither  $a_1 \dots a_{l-1}$  nor  $a_{r+1} \dots a_n$  contains  $x$ .

If no such  $x$  exists, the power is 0.

Call an array  $b$  a rival to  $a$  if the following holds:

The length of both  $a$  and  $b$  are equal to some  $n$ .

Over all  $l, r$  where  $1 \leq l \leq r \leq n$ , the power of  $a_l \dots a_r$  equals the power of  $b_l \dots b_r$ .

The elements of  $b$  are positive.

Ntarsis wants you to find a rival  $b$  to  $a$  such that the sum of  $b_i$  over  $1 \leq i \leq n$  is maximized. Help him with this task!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 template<class Info>
6 # struct SegmentTree;
7 constexpr i64 inf = 1E18;
8 # struct Info;
9 Info operator+(Info a, Info b) {
10     Info c;
11     c.cnt = a.cnt + b.cnt;
12     c.sum = a.sum + b.sum;
13     c.min = min(a.min, b.min);
14     return c;
15 }
16 void solve() {
17     int n;
18     cin >> n;
19     vector<int> a(n);
20     for (int i = 0; i < n; i++) {
21         cin >> a[i];
22     }
23     map<int, int> l, r;
24     for (int i = 0; i < n; i++) {
25         if (!l.count(a[i])) {
26             l[a[i]] = i;
27         }
28         r[a[i]] = i;
29     }
30     vector<array<int, 3>> e;
31     vector<int> b(n);
32     for (auto [x, lx] : l) {
33         e.push_back({x, lx, r[x]});
34     }
35     reverse(e.begin(), e.end());
36     map<int, int> f;
37     vector<array<int, 3>> v;
38     DSU dsu(n + 1);
39     vector<bool> fix(n);
40     for (auto [x, l, r] : e) {
41         auto it = f.lower_bound(l);
42         if (it != f.end() && it->second < r) {
43             continue;
44         }
45         v.push_back({x, l, r});
46         dsu.merge(l + 1, l);
47         dsu.merge(r + 1, r);
48         fix[l] = fix[r] = true;
49         b[l] = b[r] = x;
50         f[l] = r;
51         it--;
52         while (it != f.begin() && prev(it)->second > r) {
53             f.erase(prev(it));
54         }
55     }
56     for (auto [x, l, r] : v) {
57         for (int i = dsu.find(l); i <= r; i = dsu.find(i)) {
58             dsu.merge(i + 1, i);

```

```
59             b[i] = x;
60         }
61     }
62     i64 ans = -1;
63     i64 sum = accumulate(b.begin(), b.end(), 0LL);
64     if (count(b.begin(), b.end(), 0) == 0) {
65         ans = sum;
66     }
67     int lz = n, rz = -1;
68     for (int i = 0; i < n; i++) {
69         if (b[i] == 0) {
70             lz = min(lz, i);
71             rz = i;
72         }
73     }
74     vector<int> p(n);
75     iota(p.begin(), p.end(), 0);
76     sort(p.begin(), p.end(), [&](int i, int j) {
77         return b[i] > b[j];
78     });
79     SegmentTree<Info> seg(n);
80     for (int i = 0; i < n; i++) {
81         if (!fix[i]) {
82             seg.modify(i, {1, b[i], b[i]});
83         }
84     }
85     int Val = -1;
86     int Lv = -1, Rv = -1;
87     for (int i = 0, j = 0, k = 0; i < v.size(); i++) {
88         auto [x, l, r] = v[i];
89         if (j < i) {
90             j = i;
91         }
92         while (j + 1 < v.size() && v[j][0] - 1 == v[j + 1][0]) {
93             j++;
94         }
95         int val = v[j][0] - 1;
96         if (val == 0) {
97             continue;
98         }
99         while (k < n && b[p[k]] > val) {
100             if (!fix[p[k]]) {
101                 seg.modify(p[k], {0, 0, b[p[k]]});
102             }
103             k++;
104         }
105         int lv = min(l, lz);
106         int rv = max(r, rz);
107         i64 res = sum;
108         auto info = seg.rangeQuery(0, lv + 1);
109         if (info.cnt) {
110             res += 1LL * info.cnt * val - info.sum;
111         } else {
112             res += val - info.min;
113         }
114         info = seg.rangeQuery(rv, n);
115         if (info.cnt) {
116             res += 1LL * info.cnt * val - info.sum;
117         } else {
118             res += val - info.min;
119         }
120         if (lv < rv) {
121             info = seg.rangeQuery(lv + 1, rv);
122             res += 1LL * info.cnt * val - info.sum;
```

```
123         }
124     if (res > ans) {
125         ans = res;
126         Val = val;
127         Lv = lv;
128         Rv = rv;
129     }
130 }
131 if (Val != -1) {
132     int x = -1;
133     bool lt = false;
134     for (int i = Lv; i >= 0; i--) {
135         if (!fix[i]) {
136             if (b[i] < Val) {
137                 b[i] = Val;
138                 lt = true;
139             } else {
140                 if (x == -1 || b[i] < b[x]) {
141                     x = i;
142                 }
143             }
144         }
145     }
146     if (!lt) {
147         b[x] = Val;
148     }
149     x = -1;
150     lt = false;
151     for (int i = Rv; i < n; i++) {
152         if (!fix[i]) {
153             if (b[i] < Val) {
154                 b[i] = Val;
155                 lt = true;
156             } else {
157                 if (x == -1 || b[i] < b[x]) {
158                     x = i;
159                 }
160             }
161         }
162     }
163     if (!lt) {
164         b[x] = Val;
165     }
166     for (int i = Lv + 1; i < Rv; i++) {
167         if (!fix[i] && b[i] < Val) {
168             b[i] = Val;
169         }
170     }
171 }
172 for (int i = 0; i < n; i++) {
173     cout << b[i] << " \n"[i == n - 1];
174 }
175 }
176 int main() {
177     ios::sync_with_stdio(false);
178     cin.tie(nullptr);
179     int t;
180     cin >> t;
181     while (t--) {
182         solve();
183     }
184     return 0;
185 }
```

## 923: LuoTianyi and Cartridge

- Time limit: 3 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

LuoTianyi is watching the anime Made in Abyss. She finds that making a Cartridge is interesting. To describe the process of making a Cartridge more clearly, she abstracts the original problem and gives you the following problem.

You are given a tree  $T$  consisting of  $n$  vertices. Each vertex has values  $a_i$  and  $b_i$  and each edge has values  $c_j$  and  $d_j$ .

Now you are aim to build a tree  $T'$  as follows:

First, select  $p$  vertices from  $T$  ( $p$  is a number chosen by yourself) as the vertex set  $S'$  of  $T'$ .

Next, select  $p - 1$  edges from  $T$  one by one (you cannot select one edge more than once).

May you have chosen the  $j$ -th edge connects vertices  $x_j$  and  $y_j$  with values  $(c_j, d_j)$ , then you can choose two vertices  $u$  and  $v$  in  $S'$  that satisfy the edge  $(x_j, y_j)$  is contained in the simple path from  $u$  to  $v$  in  $T$ , and link  $u$  and  $v$  in  $T'$  by the edge with values  $(c_j, d_j)$  ( $u$  and  $v$  shouldn't be contained in one connected component before in  $T'$ ).

Let  $A$  be the minimum of values  $a_i$  in  $T'$  and  $C$  be the minimum of values  $c_i$  in  $T'$ . Let  $B$  be the sum of  $b_i$  in  $T'$  and  $D$  be the sum of values  $d_i$  in  $T'$ . Let  $\min(A, C) \cdot (B + D)$  be the cost of  $T'$ . You need to find the maximum possible cost of  $T'$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct HLD;
5 # struct DSU;
6 template<class Info>
7 # struct SegmentTree;
8 constexpr int inf = 1E9;
9 # struct Min;
10 Min operator+(Min a, Min b) {
11     return {min(a.x, b.x)};
12 }
13 template <typename T>
14 # struct Fenwick;
15 int main() {
16     ios::sync_with_stdio(false);
17     cin.tie(nullptr);
18     int n;
19     cin >> n;

```

```

20     vector<int> a(2 * n - 1), b(2 * n - 1);
21     for (int i = 0; i < n; i++) {
22         cin >> a[i];
23     }
24     for (int i = 0; i < n; i++) {
25         cin >> b[i];
26     }
27     HLD t(n);
28     vector<int> x(n - 1), y(n - 1);
29     for (int i = 0; i < n - 1; i++) {
30         cin >> x[i] >> y[i] >> a[n + i] >> b[n + i];
31         x[i]--, y[i]--;
32         t.addEdge(x[i], y[i]);
33     }
34     t.work();
35     auto vb = b;
36     sort(vb.begin(), vb.end(), greater());
37     for (int i = 0; i < 2 * n - 1; i++) {
38         b[i] = lower_bound(vb.begin(), vb.end(), b[i], greater()) - vb.begin();
39     }
40     auto cmp = [&](int x, int y) {
41         return t.in[x] < t.in[y];
42     };
43     set<int, decltype(cmp)> s(cmp);
44     vector<int> c(n), d(n);
45     for (int i = 0; i < n - 1; i++) {
46         if (t.parent[x[i]] != y[i]) {
47             swap(x[i], y[i]);
48         }
49         c[x[i]] = a[n + i];
50         d[x[i]] = b[n + i];
51     }
52     vector<int> ord(2 * n - 1);
53     iota(ord.begin(), ord.end(), 0);
54     sort(ord.begin(), ord.end(), [&](int i, int j) {
55         return a[i] > a[j];
56     });
57     SegmentTree<Min> seg(n);
58     int top = -1;
59     Fenwick<int> fc(2 * n - 1), fce(2 * n - 1);
60     Fenwick<i64> fs(2 * n - 1), fse(2 * n - 1);
61     i64 sumb = 0;
62     auto add = [&](int x, int t = 1) {
63         fc.add(x, t);
64         fs.add(x, vb[x] * t);
65     };
66     set<int, decltype(cmp)> sdown(cmp);
67     i64 sumv = 0;
68     auto addv = [&](int x) {
69         // cerr << "addvertex " << x + 1 << "\n";
70         sumv += vb[b[x]];
71         add(b[x]);
72         auto it = sdown.upper_bound(x);
73         int u = -1;
74         if (it != sdown.begin() && t.in[x] < t.out[*prev(it)]) {
75             u = *prev(it);
76         }
77         if (u != -1) {
78             auto v = seg.rangeQuery(t.in[u], t.out[u]).x;
79             if (v != inf) {
80                 add(v);
81                 sumb -= vb[v];
82             }
83         }
84     };

```

```

84     seg.modify(t.in[x], {b[x]});
85     if (u != -1) {
86         auto v = seg.rangeQuery(t.in[u], t.out[u]).x;
87         if (v != inf) {
88             add(v, -1);
89             sumb += vb[v];
90         }
91     }
92 };
93 int L = n, R = 0;
94 int cnte = 0;
95 auto adde = [&](int x) {
96     // cerr << "addedge (" << x + 1 << ", " << t.parent[x] + 1 << ")\n";
97     cnte++;
98     sumb += vb[d[x]];
99     fce.add(d[x], 1);
100    fse.add(d[x], vb[d[x]]);
101    L = min(L, t.in[x]);
102    R = max(R, t.out[x]);
103    if (top == -1 || t.dep[x] < t.dep[top]) {
104        top = x;
105    }
106    auto it = sdown.upper_bound(x);
107    if (it != sdown.begin() && t.in[x] < t.out[*prev(it)]) {
108        int u = *prev(it);
109        auto v = seg.rangeQuery(t.in[u], t.out[u]).x;
110        if (v != inf) {
111            add(v);
112            sumb -= vb[v];
113        }
114        sdown.erase(u);
115    }
116    it = sdown.lower_bound(x);
117    if (it != sdown.end() && t.in[*it] < t.out[x]) {
118        return;
119    }
120    sdown.insert(x);
121    auto v = seg.rangeQuery(t.in[x], t.out[x]).x;
122    if (v != inf) {
123        add(v, -1);
124        sumb += vb[v];
125    }
126 };
127 auto query = [&]() {
128     if (cnte >= s.size() - 1) {
129         int res = s.size() - 1;
130         i64 sum = sumv;
131         int t = fce.kth(res);
132         sum += fse.sum(t);
133         res -= fce.sum(t);
134         if (res) {
135             sum += 1LL * res * vb[t];
136         }
137         return sum;
138     }
139     int res = cnte + 1 - sdown.size();
140     i64 sum = sumb;
141     if (top != -1) {
142         if (t.in[top] > L || t.out[top] < R) {
143             top = -1;
144         }
145     }
146     int v = -1;
147     if (top != -1) {

```

```

148         res--;
149         v = (seg.rangeQuery(0, t.in[top]) + seg.rangeQuery(t.out[top], n)).x;
150         add(v, -1);
151         sum += vb[v];
152     }
153     int t = fc.kth(res);
154     sum += fs.sum(t);
155     res -= fc.sum(t);
156     if (res) {
157         sum += 1LL * res * vb[t];
158     }
159     if (v != -1) {
160         add(v);
161     }
162     // cerr << "p : " << cnte + 1 << "\n";
163     // cerr << "sumb : " << sumb << "\n";
164     // cerr << "sum : " << sum << "\n";
165     // cerr << "----\n";
166     return sum;
167 };
168 vector<bool> vis(n);
169 i64 ans = 0;
170 for (int i = 0; i < n; i++) {
171     ans = max(ans, 1LL * a[i] * vb[b[i]]);
172 }
173 DSU dsu(n);
174 for (auto i : ord) {
175     i64 min = a[i];
176     if (i < n) {
177         auto it = s.insert(i).first;
178         int lst = it == s.begin() ? *s.rbegin() : *prev(it);
179         int nxt = next(it) == s.end() ? *s.begin() : *next(it);
180         for (auto x : {lst, nxt}) {
181             int y = i;
182             x = dsu.find(x);
183             y = dsu.find(y);
184             while (x != y) {
185                 if (t.dep[x] < t.dep[y]) {
186                     swap(x, y);
187                 }
188                 if (vis[x]) {
189                     adde(x);
190                 }
191                 vis[x] = true;
192                 dsu.merge(t.parent[x], x);
193                 x = dsu.find(x);
194             }
195         }
196         addv(i);
197     } else {
198         i -= n;
199         if (vis[x[i]]) {
200             adde(x[i]);
201         }
202         vis[x[i]] = true;
203     }
204     if (!s.empty()) {
205         // cerr << "min : " << min << "\n";
206         ans = max(ans, min * query());
207     }
208 }
209 cout << ans << "\n";
210 return 0;
211 }

```

## 924: A task for substrings

- Time limit: 4 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

To do this, he took the string  $t$  and a set of  $n$  strings  $s_1, s_2, s_3, \dots, s_n$ . Philip has  $m$  queries, in the  $i$ th of them, Philip wants to take a substring of the string  $t$  from  $l_i$ th to  $r_i$ th character, and count the number of its substrings that match some string from the set. More formally, Philip wants to count the number of pairs of positions  $a, b$ , such that  $l_i \leq a \leq b \leq r_i$ , and the substring of the string  $t$  from  $a$ th to  $b$ th character coincides with some string  $s_j$  from the set.

A substring of the string  $t$  from  $a$ th to  $b$ th character is a string obtained from  $t$  by removing the  $a - 1$  character from the beginning and  $|t| - b$  characters from the end, where  $|t|$  denotes the length of the string  $t$ .

Philip has already solved this problem, but can you?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct AhoCorasick;
5 template <typename T>
6 # struct Fenwick;
7 int main() {
8     ios::sync_with_stdio(false);
9     cin.tie(nullptr);
10    int n, m;
11    cin >> n >> m;
12    string t;
13    cin >> t;
14    AhoCorasick ac1, ac2;
15    vector<string> s(n);
16    vector<int> pos(n);
17    for (int i = 0; i < n; i++) {
18        cin >> s[i];
19        pos[i] = ac1.add(s[i]);
20        ac2.add(string(s[i].rbegin(), s[i].rend()));
21    }
22    ac1.work();
23    ac2.work();
24    vector<i64> ans(m);
25    int L = t.size();
26    vector<i64> f(L + 1);
27    vector<int> sum(ac1.size());
28    vector<vector<int>> adj1(ac1.size()), adj2(ac2.size());
29    for (int i = 2; i < ac1.size(); i++) {
30        adj1[ac1.link(i)].push_back(i);
31    }
32    for (int i = 2; i < ac2.size(); i++) {

```

```

33         adj2[ac2.link(i)].push_back(i);
34     }
35     for (auto x : pos) {
36         sum[x] += 1;
37     }
38     vector<int> in1(ac1.size()), out1(ac1.size());
39     vector<int> in2(ac2.size()), out2(ac2.size());
40     int log = __lg(ac2.size());
41     vector<int> par(log + 1, vector<int>(ac2.size()));
42     int cur = 0;
43     function<void(int)> dfs1 = [&](int x) {
44         in1[x] = cur++;
45         for (auto y : adj1[x]) {
46             sum[y] += sum[x];
47             dfs1(y);
48         }
49         out1[x] = cur;
50     };
51     dfs1(1);
52     cur = 0;
53     function<void(int)> dfs2 = [&](int x) {
54         par[0][x] = ac2.link(x);
55         for (int k = 0; k < log; k++) {
56             if (!par[k][x]) {
57                 break;
58             }
59             par[k + 1][x] = par[k][par[k][x]];
60         }
61         in2[x] = cur++;
62         for (auto y : adj2[x]) {
63             dfs2(y);
64         }
65         out2[x] = cur;
66     };
67     dfs2(1);
68     vector<int> pre(L + 1), suf(L + 1);
69     pre[0] = 1;
70     for (int i = 0, p = 1; i < L; i++) {
71         p = ac1.next(p, t[i]);
72         f[i + 1] = f[i] + sum[p];
73         pre[i + 1] = p;
74     }
75     suf[L] = 1;
76     for (int i = L - 1, p = 1; i >= 0; i--) {
77         p = ac2.next(p, t[i]);
78         suf[i] = p;
79     }
80     vector<array<int, 4>> events;
81     for (auto s : s) {
82         int L = s.size();
83         vector<int> suf(L + 1);
84         suf[L] = 1;
85         for (int i = L - 1; i >= 0; i--) {
86             suf[i] = ac2.next(suf[i + 1], s[i]);
87         }
88         for (int i = 0, p = 1; i < L - 1; i++) {
89             p = ac1.next(p, s[i]);
90             int q = suf[i + 1];
91             events.push_back({in1[p], 0, in2[q], 1});
92             events.push_back({in1[p], 0, out2[q], -1});
93             events.push_back({out1[p], 0, in2[q], -1});
94             events.push_back({out1[p], 0, out2[q], 1});
95         }
96     }

```

```

97     for (int i = 0; i < m; i++) {
98         int l, r;
99         cin >> l >> r;
100        l--;
101        ans[i] = f[r] - f[l];
102        int p = pre[l], q = suf[l];
103        for (int k = log; k >= 0; k--) {
104            if (ac2.len(par[k][q]) > r - l) {
105                q = par[k][q];
106            }
107        }
108        if (ac2.len(q) > r - l) {
109            q = par[0][q];
110        }
111        events.push_back({in1[p], 1, in2[q], i});
112    }
113    sort(events.begin(), events.end());
114    Fenwick<int> fen(cur);
115    for (auto [_, t, y, k] : events) {
116        if (t == 0) {
117            fen.add(y, k);
118        } else {
119            ans[k] -= fen.sum(y + 1);
120        }
121    }
122    for (int i = 0; i < m; i++) {
123        cout << ans[i] << " \n"[i == m - 1];
124    }
125    return 0;
126 }

```

## 925: Iron Man

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Tony Stark is playing a game with his suits (they have auto-pilot now). He lives in Malibu. Malibu has  $n$  junctions numbered from 1 to  $n$ , connected with  $n - 1$  roads. One can get from a junction to any other junction using these roads (graph of Malibu forms a tree).

Tony has  $m$  suits. There's a special plan for each suit. The  $i$ -th suit will appear at the moment of time  $t_i$  in the junction  $v_i$ , and will move to junction  $u_i$  using the shortest path between  $v_i$  and  $u_i$  with the speed  $c_i$  roads per second (passing a junctions takes no time), and vanishing immediately when arriving at  $u_i$  (if it reaches  $u_i$  in time  $q$ , it's available there at moment  $q$ , but not in further moments). Also, suits move continuously (for example if  $v_i \neq u_i$ , at time  $t$  it's in the middle of a road. Please note that if  $v_i = u_i$  it means the suit will be at junction number  $v_i$  only at moment  $t_i$  and then it vanishes).

An explosion happens if at any moment of time two suits share the same exact location (it may be in a junction or somewhere on a road; while appearing, vanishing or moving).

Your task is to tell Tony the moment of the the first explosion (if there will be any).

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct HLD;
5 # struct Line;
6 int main() {
7     ios::sync_with_stdio(false);
8     cin.tie(nullptr);
9     int n, m;
10    cin >> n >> m;
11    HLD t(n);
12    for (int i = 1; i < n; i++) {
13        int u, v;
14        cin >> u >> v;
15        u--, v--;
16        t.addEdge(u, v);
17    }
18    t.work();
19    vector<vector<Line>> lines(n);
20    for (int i = 0; i < m; i++) {
21        int s, c, u, v;
22        cin >> s >> c >> u >> v;
23        u--, v--;
24        int d = t.dist(u, v);
25        int du = 0, dv = d;
26        while (t.top[u] != t.top[v]) {
27            if (t.dep[t.top[u]] > t.dep[t.top[v]]) {
28                lines[t.top[u]].push_back({s + 1. * du / c, s + 1. * (du + t.dep[u] - t.
29                    dep[t.top[u]] + 1) / c
30                    , -c, t.dep[u] + du + c * s});
31                du += t.dep[u] - t.dep[t.top[u]] + 1;
32                u = t.parent[t.top[u]];
33            } else {
34                dv -= t.dep[v] - t.dep[t.top[v]] + 1;
35                lines[t.top[v]].push_back({s + 1. * dv / c, s + 1. * (dv + t.dep[v] - t.
36                    dep[t.top[v]] + 1) / c
37                    , c, t.dep[t.top[v]] - 1 - dv - c * s});
38                v = t.parent[t.top[v]];
39            }
40            if (t.dep[u] <= t.dep[v]) {
41                lines[t.top[v]].push_back({s + 1. * du / c, s + 1. * dv / c
42                    , c, t.dep[u] - du - c * s});
43            } else {
44                lines[t.top[u]].push_back({s + 1. * du / c, s + 1. * dv / c
45                    , -c, t.dep[u] + du + c * s});
46            }
47            double ans = -1;
48            for (int i = 0; i < n; i++) {
49                if (t.top[i] != i) {
50                    continue;
51                }
52                auto &L = lines[i];
53                vector<tuple<double, int, int>> events;
```

```

54     for (int j = 0; j < L.size(); j++) {
55         events.emplace_back(L[j].l, 0, j);
56         events.emplace_back(L[j].r, 1, j);
57     }
58     sort(events.begin(), events.end());
59     double cur = 0;
60     auto cmp = [&](int i, int j) {
61         double a = L[i].k * cur + L[i].b;
62         double b = L[j].k * cur + L[j].b;
63         if (a != b) {
64             return a < b;
65         }
66         return i < j;
67     };
68     auto updateAns = [&](double x) {
69         if (ans < 0 || ans > x) {
70             ans = x;
71         }
72     };
73     auto update = [&](int i, int j) {
74         double l = max(L[i].l, L[j].l);
75         double a = L[i].k * l + L[i].b;
76         double b = L[j].k * l + L[j].b;
77         if (a == b) {
78             updateAns(l);
79             return;
80         }
81         double r = min(L[i].r, L[j].r);
82         double c = L[i].k * r + L[i].b;
83         double d = L[j].k * r + L[j].b;
84         if (c == d) {
85             updateAns(r);
86             return;
87         }
88         if ((a > b) == (c > d)) {
89             return;
90         }
91         double x = (L[i].b - L[j].b) / (L[j].k - L[i].k);
92         updateAns(x);
93     };
94     set<int, decltype(cmp)> s(cmp);
95     for (auto [x, t, i] : events) {
96         cur = x;
97         if (ans != -1 && x >= ans) {
98             break;
99         }
100        if (t == 0) {
101            auto it = s.insert(i).first;
102            if (it != s.begin()) {
103                update(*prev(it), i);
104            }
105            it++;
106            if (it != s.end()) {
107                update(*it, i);
108            }
109        } else {
110            auto it = s.erase(s.find(i));
111            if (it != s.begin() && it != s.end()) {
112                update(*prev(it), *it);
113            }
114        }
115    }
116    if (ans < 0) {

```

```

118         cout << -1 << "\n";
119         return 0;
120     }
121     cout << fixed << setprecision(10);
122     cout << ans << "\n";
123     return 0;
124 }
```

## 926: Flow Control

- Time limit: 6 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Raj has a single physical network line that connects his office to the Internet. This line bandwidth is  $b$  bytes per millisecond.

There are  $n$  users who would like to use this network line to transmit some data. The  $i$ -th of them will use the line from millisecond  $s_i$  to millisecond  $f_i$  inclusive. His initial data rate will be set to  $d_i$ . That means he will use data rate equal to  $d_i$  for millisecond  $s_i$ , and then it will change according to the procedure described below.

The flow control will happen as follows. Suppose there are  $m$  users trying to transmit some data via the given network line during millisecond  $x$ . Denote as  $t_i$  the data rate that the  $i$ -th of these  $m$  users has at the beginning of this millisecond. All  $t_i$  are non-negative integer values.

If  $m = 0$ , i. e. there are no users trying to transmit data during this millisecond, nothing happens.

If the sum of all  $t_i$  is less than or equal to  $b$ , each active user successfully completes his transmission (the  $i$ -th active user transmits  $t_i$  bytes). After that, the data rate of each active user grows by 1, i. e. each  $t_i$  is increased by 1.

If the sum of all  $t_i$  is greater than  $b$ , the congestion occurs and no data transmissions succeed this millisecond at all. If that happens, each  $t_i$  decreases twice, i. e. each  $t_i$  is replaced with  $\lfloor \frac{t_i}{2} \rfloor$ .

Raj knows all the values  $n, b, s_i, f_i$ , and  $d_i$ , he wants to calculate the total number of bytes transmitted by all the users in the aggregate.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class Info, class Tag>
5 # struct LazySegmentTree;
```

```

6  constexpr i64 inf = 1E18;
7  # struct Tag;
8  # struct Info;
9  Info operator+(Info a, Info b) {
10     Info c;
11     c.min = min(a.min, b.min);
12     c.max = max(a.max, b.max);
13     c.sum = a.sum + b.sum;
14     c.act = a.act + b.act;
15     return c;
16 }
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(nullptr);
20     int n, b;
21     cin >> n >> b;
22     i64 ans = 0;
23     vector<array<int, 3>> events;
24     for (int i = 0; i < n; i++) {
25         int s, f, d;
26         cin >> s >> f >> d;
27         s--;
28         events.push_back({s, i, d});
29         events.push_back({f, i, 0});
30     }
31     sort(events.begin(), events.end());
32     LazySegmentTree<Info, Tag> seg(n);
33     int cur = 0;
34     for (auto [t, i, d] : events) {
35         Info last = {-1, -1, -1, -1};
36         i64 lastc = -1;
37         i64 lastans = -1;
38         while (cur < t) {
39             auto info = seg.rangeQuery(0, n);
40             if (info.act == 0) {
41                 cur = t;
42                 break;
43             }
44             if (info.min == last.min && info.max == last.max && info.sum == last.sum &&
45                 info.max - info.min <= 1) {
46                 i64 dt = cur - lastc;
47                 i64 dans = ans - lastans;
48                 i64 phase = (t - cur) / dt;
49                 cur += phase * dt;
50                 ans += phase * dans;
51             }
52             last = info;
53             lastc = cur;
54             lastans = ans;
55             i64 u = (b - info.sum + info.act) / info.act;
56             if (u > t - cur) {
57                 u = t - cur;
58             }
59             if (u < 0) {
60                 u = 0;
61             }
62             cur += u;
63             ans += (info.sum + info.sum + info.act * (u - 1)) * u / 2;
64             seg.rangeApply(0, n, {u});
65             if (t == cur) {
66                 break;
67             }
68             cur += 1;
69             seg.half();
70     }
71 }

```

```

69         }
70         if (d != 0) {
71             seg.modify(i, {d, d, d, 1});
72         } else {
73             seg.modify(i, Info());
74         }
75     }
76     cout << ans << "\n";
77     return 0;
78 }
```

## 927: Treasure Hunt

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Define the beauty value of a sequence  $b_1, b_2, \dots, b_c$  as the maximum value of  $\sum_{i=1}^q b_i + \sum_{i=s}^t b_i$ , where  $q, s, t$  are all integers and  $s > q$  or  $t \leq q$ . Note that  $b_i = 0$  when  $i < 1$  or  $i > c$ ,  $\sum_{i=s}^t b_i = 0$  when  $s > t$ .

For example, when  $b = [-1, -2, -3]$ , we may have  $q = 0, s = 3, t = 2$  so the beauty value is  $0 + 0 = 0$ . And when  $b = [-1, 2, -3]$ , we have  $q = s = t = 2$  so the beauty value is  $1 + 2 = 3$ .

You are given a sequence  $a$  of length  $n$ , determine the sum of the beauty value of all non-empty subsegments  $a_l, a_{l+1}, \dots, a_r$  ( $1 \leq l \leq r \leq n$ ) of the sequence  $a$ .

Print the answer modulo 998 244 353.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #pragma GCC optimize("Ofast")
4 using i64 = long long;
5 constexpr int P = 998244353;
6 using i64 = long long;
7 // assume -P <= x < 2P
8 int norm(int x) {
9     if (x < 0) {
10         x += P;
11     }
12     if (x >= P) {
13         x -= P;
14     }
15     return x;
16 }
17 template<class T>
18 T power(T a, i64 b) {
19     T res = 1;
```

```

20     for (; b; b /= 2, a *= a) {
21         if (b % 2) {
22             res *= a;
23         }
24     }
25     return res;
26 }
27 # struct Z;
28 template<class Info, class Tag>
29 # struct LazySegmentTree;
30 constexpr i64 inf = 1E18;
31 # struct Tag;
32 # struct Info;
33 Info operator+(const Info &a, const Info &b) {
34     Info c;
35     c.n = a.n + b.n;
36     c.max = max(a.max, b.max);
37     c.min = min(a.min, b.min);
38     c.sum = a.sum + b.sum;
39     return c;
40 }
41 void solve() {
42     int n;
43     cin >> n;
44     vector<int> a(n);
45     for (int i = 0; i < n; i++) {
46         cin >> a[i];
47     }
48     Z ans = 0;
49     vector<i64> s(n + 1);
50     for (int i = 0; i < n; i++) {
51         s[i + 1] = s[i] + a[i];
52     }
53     LazySegmentTree<Info, Tag> seg(n + 1);
54     vector<int> h{-1, 0};
55     seg.modify(0, Info(0));
56     Z sum = 0;
57     for (int i = 1; i <= n; i++) {
58         while (h.size() > 1 && s[i] > s[h.back()]) {
59             int r = h.back();
60             h.pop_back();
61             int l = h.back();
62             sum -= Z(s[r]) * (r - l);
63             i64 d;
64             if (l == -1) d = s[i];
65             else d = min(s[i], s[l]);
66             d -= s[r];
67             seg.rangeApply(l + 1, i, Tag(-1, d));
68             if (l + 1 < i) {
69                 i64 v = seg.rangeQuery(l + 1, l + 2).max;
70                 int j = seg.search(0, l + 1, v);
71                 seg.rangeApply(j, l + 1, Tag(v, 0));
72             }
73         }
74         sum += Z(s[i]) * (i - h.back());
75         h.push_back(i);
76         ans += seg.rangeQuery(0, i).sum;
77         seg.modify(i, Info(0));
78         ans += sum;
79     }
80     for (int i = 0; i <= n; i++) {
81         ans -= Z(s[i]) * (n - i + 1);
82     }
83     cout << ans << "\n";

```

```

84 }
85 int main() {
86     ios::sync_with_stdio(false);
87     cin.tie(nullptr);
88     int t;
89     cin >> t;
90     while (t--) {
91         solve();
92     }
93     return 0;
94 }
```

## 928: Codeforces Scoreboard

- Time limit: 3 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are participating in a Codeforces Round with  $n$  problems.

You spend exactly one minute to solve each problem, the time it takes to submit a problem can be ignored. You can only solve at most one problem at any time. The contest starts at time 0, so you can make your first submission at any time  $t \geq 1$  minutes. Whenever you submit a problem, it is always accepted.

The scoring of the  $i$ -th problem can be represented by three integers  $k_i$ ,  $b_i$ , and  $a_i$ . If you solve it at time  $t$  minutes, you get  $\max(b_i - k_i \cdot t, a_i)$  points.

Your task is to choose an order to solve all these  $n$  problems to get the maximum possible score. You can assume the contest is long enough to solve all problems.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 using u64 = unsigned long long;
5 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
6 # struct Tree;
7 void add(Tree *t, i64 a, i64 d) {
8     if (!t) return;
9     t->ta += a;
10    t->td += d;
11    int v = 0;
12    if (t->l) v = t->l->siz;
13    t->val += a + d * v;
14 }
15 void push(Tree *t) {
16     add(t->l, t->ta, t->td);
17     int v = 1;
```

```
18     if (t->l) v += t->l->siz;
19     add(t->r, t->ta + t->td * v, t->td);
20     t->ta = t->td = 0;
21 }
22 void pull(Tree *t) {
23     t->siz = 1;
24     if (t->l) t->siz += t->l->siz;
25     if (t->r) t->siz += t->r->siz;
26 }
27 pair<Tree *, Tree *> split(Tree *t, Tree *v) {
28     if (!t) return {t, t};
29     push(t);
30     if (v->val > t->val || (v->val == t->val && v->id > t->id)) {
31         auto [l, r] = split(t->l, v);
32         t->l = r;
33         pull(t);
34         return {l, t};
35     } else {
36         auto [l, r] = split(t->r, v);
37         t->r = l;
38         pull(t);
39         return {t, r};
40     }
41 }
42 void insert(Tree *&t, Tree *x) {
43     if (!t) {
44         t = x;
45         return;
46     }
47     if (x->w > t->w) {
48         auto [l, r] = split(t, x);
49         x->l = l;
50         x->r = r;
51         pull(x);
52         t = x;
53         return;
54     }
55     push(t);
56     if (x->val > t->val || (x->val == t->val && x->id > t->id)) {
57         insert(t->l, x);
58     } else {
59         insert(t->r, x);
60     }
61     pull(t);
62 }
63 i64 get(Tree *t) {
64     if (!t) return 0;
65     i64 val = max(0LL, t->val);
66     push(t);
67     val += get(t->l);
68     val += get(t->r);
69     delete t;
70     return val;
71 }
72 void dfs(Tree *t) {
73     if (!t) return;
74     cerr << t->val << " ";
75     push(t);
76     dfs(t->l);
77     dfs(t->r);
78 }
79 void solve() {
80     int n;
81     cin >> n;
```

```

82     vector<pair<int, int>> kb(n);
83     i64 suma = 0;
84     for (int i = 0; i < n; i++) {
85         int k, b, a;
86         cin >> k >> b >> a;
87         suma += a;
88         b -= a;
89         kb[i] = {k, b};
90     }
91     sort(kb.begin(), kb.end(), greater());
92     Tree *t{};

93     int lastk = 0, cnt = 0;
94     for (auto [k, b] : kb) {
95         if (t) {
96             add(t, k - lastk, k - lastk);
97         }
98         lastk = k;
99         insert(t, new Tree(b, cnt++));
100        // cerr << "siz : " << t->siz << "\n";
101        // dfs(t);
102        // cerr << "\n";
103    }
104    add(t, -lastk, -lastk);
105    auto ans = get(t) + suma;
106    cout << ans << "\n";
107 }
108 int main() {
109     ios::sync_with_stdio(false);
110     cin.tie(nullptr);
111     int t;
112     cin >> t;
113     while (t--) {
114         solve();
115     }
116     return 0;
117 }
```

## 929: Two Subtrees

- Time limit: 9 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

You are given a rooted tree consisting of  $n$  vertices. The vertex 1 is the root. Each vertex has an integer written on it; this integer is  $val_i$  for the vertex  $i$ .

You are given  $q$  queries to the tree. The  $i$ -th query is represented by two vertices,  $u_i$  and  $v_i$ . To answer the query, consider all vertices  $w$  that lie in the subtree of  $u_i$  or  $v_i$  (if a vertex is in both subtrees, it is counted twice). For all vertices in these two subtrees, list all integers written on them, and find the integer with the maximum number of occurrences. If there are multiple integers with maximum number of occurrences, the minimum among them is the answer.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int B = 600;
5 constexpr int V = 200001;
6 constexpr int BV = 500;
7 int freq[V], cnt[V];
8 int buck[V][V / BV + 1];
9 int best = 0;
10 void add(int c) {
11     int &f = cnt[c];
12     buck[f][c / BV]--;
13     buck[f + 1][c / BV]++;
14     freq[f]--;
15     freq[+f]++;
16     best = max(best, f);
17 }
18 void rem(int c) {
19     int &f = cnt[c];
20     buck[f][c / BV]--;
21     buck[f - 1][c / BV]++;
22     freq[f]--;
23     freq[-f]++;
24     if (!freq[best]) best--;
25 }
26 int main() {
27     ios::sync_with_stdio(false);
28     cin.tie(nullptr);
29     int n;
30     cin >> n;
31     vector<int> val(n);
32     for (int i = 0; i < n; i++) {
33         cin >> val[i];
34     }
35     vector<vector<int>> adj(2 * n);
36     for (int i = 1; i < n; i++) {
37         int x, y;
38         cin >> x >> y;
39         x--, y--;
40         adj[x].push_back(y);
41         adj[y].push_back(x);
42     }
43     vector<int> in(n + 1), out(n + 1), down(n + 1, n), siz(n + 1), mx(n + 1, n);
44     vector<int> key, a;
45     int cur = 0;
46     auto dfs = [&](auto dfs, int x, int p) -> int {
47         if (p != -1) {
48             adj[x].erase(find(adj[x].begin(), adj[x].end(), p));
49         }
50         in[x] = cur++;
51         a.push_back(val[x]);
52         int res = 1;
53         int cnt = 0;
54         for (auto y : adj[x]) {
55             res += dfs(dfs, y, x);
56             cnt += (down[y] != n);
57             if (siz[down[y]] > siz[down[x]]) down[x] = down[y];
58         }
59         mx[x] = down[x];
60         out[x] = cur;

```

```

61         siz[x] = out[x] - in[x];
62         if (cnt >= 2 || res >= B) {
63             down[x] = x;
64             key.push_back(x);
65             return 0;
66         } else {
67             return res;
68         }
69     };
70     dfs(dfs, 0, -1);
71     key.push_back(n);
72     int K = key.size();
73     sort(key.begin(), key.end(), [&](int i, int j) {
74         return siz[i] < siz[j];
75     });
76     vector<int> id(n + 1, -1);
77     for (int i = 0; i < K; i++) {
78         id[key[i]] = i;
79     }
80     vector qry(K, vector<vector<tuple<int, int, int>>>(K));
81     vector ch(K, vector<vector<pair<int, int>>>(K));
82     auto query = [&](int x, int y) {
83         if (x == n && y == n) return;
84         int dx = down[x];
85         int dy = down[y];
86         int ndx = mx[x], ndy = mx[y];
87         int ax = siz[dx] + siz[ndx];
88         int ay = siz[ndx] + siz[dy];
89         if (ax > ay && y != n) dy = ndy;
90         else dx = ndx;
91         if (id[dx] < id[dy]) swap(dx, dy);
92         ch[id[dx]][id[dy]].emplace_back(id[x], id[y]);
93     };
94     for (auto x : key) {
95         for (auto y : key) {
96             query(x, y);
97             if (y == x) break;
98         }
99     }
100    int q;
101    cin >> q;
102    for (int i = 0; i < q; i++) {
103        int x, y;
104        cin >> x >> y;
105        x--, y--;
106        int a = down[x], b = down[y];
107        if (id[a] < id[b]) {
108            swap(a, b);
109            swap(x, y);
110        }
111        qry[id[a]][id[b]].emplace_back(x, y, i);
112    }
113    vector<tuple<int, int, int>> Qry;
114    Qry.reserve(q);
115    auto dfs1 = [&](auto dfs1, int x, int y) -> void {
116        Qry.insert(Qry.end(), qry[x][y].begin(), qry[x][y].end());
117        for (auto [dx, dy] : ch[x][y]) dfs1(dfs1, dx, dy);
118    };
119    dfs1(dfs1, 0, 0);
120    assert(Qry.size() == q);
121    int l1 = 0, r1 = 0, l2 = 0, r2 = 0;
122    freq[0] = V;
123    for (int i = 0; i <= V / BV; i++) buck[0][i] = V;
124    vector<int> Ans(q);

```

```

125     for (auto [x, y, i] : Qry) {
126         if (r1 <= in[x] || l1 >= out[x]) {
127             while (l1 < r1) rem(a[l1++]);
128             l1 = r1 = in[x];
129         }
130         if (r2 <= in[y] || l2 >= out[y]) {
131             while (l2 < r2) rem(a[l2++]);
132             l2 = r2 = in[y];
133         }
134         while (l1 > in[x]) add(a[--l1]);
135         while (r1 < out[x]) add(a[r1++]);
136         while (l1 < in[x]) rem(a[l1++]);
137         while (r1 > out[x]) rem(a[--r1]);
138         while (l2 > in[y]) add(a[--l2]);
139         while (r2 < out[y]) add(a[r2++]);
140         while (l2 < in[y]) rem(a[l2++]);
141         while (r2 > out[y]) rem(a[--r2]);
142         int j = 0;
143         while (!buck[best][j]) j++;
144         j *= BV;
145         while (cnt[j] != best) j++;
146         Ans[i] = j;
147     }
148     for (int i = 0; i < q; i++) {
149         cout << Ans[i] << "\n";
150     }
151     return 0;
152 }
```

### 930: Doremy's Paint 2

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Doremy has  $n$  buckets of paint which is represented by an array  $a$  of length  $n$ . Bucket  $i$  contains paint with color  $a_i$ . Initially,  $a_i = i$ .

Doremy has  $m$  segments  $[l_i, r_i]$  ( $1 \leq l_i \leq r_i \leq n$ ). Each segment describes an operation. Operation  $i$  is performed as follows:

For each  $j$  such that  $l_i < j \leq r_i$ , set  $a_j := a_{l_i}$ .

Doremy also selects an integer  $k$ . She wants to know for each integer  $x$  from 0 to  $m - 1$ , the number of distinct colors in the array after performing operations  $x \bmod m + 1, (x + 1) \bmod m + 1, \dots, (x + k - 1) \bmod m + 1$ . Can you help her calculate these values? Note that for each  $x$  individually we start from the initial array and perform only the given  $k$  operations in the given order.

Problem: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int N = 1 << 19;
5 int ans[N];
6 int max[N];
7 void build(int p, int l, int r) {
8     max[p] = r - 1;
9     if (r - l == 1) {
10         return;
11     }
12     int m = (l + r) / 2;
13     ans[p] = r - m;
14     build(2 * p, l, m);
15     build(2 * p + 1, m, r);
16 }
17 int query(int p, int l, int r, int x, int y, int v) {
18     if (l >= y || r <= x) {
19         return 0;
20     }
21     if (v >= max[p]) {
22         return 0;
23     }
24     if (r - l == 1) {
25         v = max[p];
26         return 1;
27     }
28     int m = (l + r) / 2;
29     if (l >= x && r <= y) {
30         if (v >= max[2 * p]) {
31             return query(2 * p + 1, m, r, x, y, v);
32         } else {
33             int res = query(2 * p, l, m, x, y, v) + ans[p];
34             v = max[p];
35             return res;
36         }
37     }
38     int res = query(2 * p, l, m, x, y, v);
39     res += query(2 * p + 1, m, r, x, y, v);
40     return res;
41 }
42 void pull(int p, int l, int r) {
43     max[p] = max(max[2 * p], max[2 * p + 1]);
44     int v = max[2 * p];
45     ans[p] = query(p, l, r, l, r, v);
46 }
47 void modify(int p, int l, int r, int x, int v) {
48     if (r - l == 1) {
49         // cerr << "modify " << x << " " << v << "\n";
50         max[p] = v;
51         return;
52     }
53     int m = (l + r) / 2;
54     if (x < m) {
55         modify(2 * p, l, m, x, v);
56     } else {
57         modify(2 * p + 1, m, r, x, v);
58     }
59     pull(p, l, r);
60 }
61 template<class Info,
62          class Merge = plus<Info>>
63 # struct SegmentTree;
64 # struct Max;
```

```
65 Max operator+(const Max &a, const Max &b) {
66     return max(a.x, b.x);
67 }
68 int main() {
69     ios::sync_with_stdio(false);
70     cin.tie(nullptr);
71     int n, m, k;
72     cin >> n >> m >> k;
73     vector<int> l(2 * m), r(2 * m);
74     for (int i = 0; i < m; i++) {
75         cin >> l[i] >> r[i];
76         l[i]--;
77         l[i + m] = l[i];
78         r[i + m] = r[i];
79     }
80     vector<vector<int>> adj(2 * m);
81     vector<int> pre(2 * m);
82     map<int, int> S;
83     S[0] = -1;
84     S[n] = -1;
85     auto split = [&](int x) {
86         auto it = prev(S.upper_bound(x));
87         S[x] = it->second;
88     };
89     auto get = [&](int x) {
90         auto it = prev(S.upper_bound(x));
91         return it->second;
92     };
93     auto cover = [&](int l, int r, int x) {
94         split(l);
95         split(r);
96         for (auto it = next(S.find(l)); it->first != r; ) {
97             it = S.erase(it);
98         }
99         S[l] = x;
100    };
101   for (int i = 0; i < 2 * m; i++) {
102       pre[i] = get(l[i]);
103       if (pre[i] != -1 && i - pre[i] >= k) {
104           pre[i] = -1;
105       }
106       if (pre[i] != -1) {
107           adj[pre[i]].push_back(i);
108       }
109       cover(l[i], r[i], i);
110   }
111   build(1, 0, n);
112   // auto f = r;
113   const int M = 2 * m;
114   int cur = 0;
115   vector<int> in(M), out(M);
116   function<void(int)> dfs = [&](int x) {
117       in[x] = cur++;
118       for (auto y : adj[x]) {
119           dfs(y);
120       }
121       out[x] = cur;
122   };
123   S.clear();
124   S[0] = -1;
125   S[M] = -1;
126   for (int i = 0; i < M; i++) {
127       if (pre[i] == -1) {
128           dfs(i);
129       }
130   }
```

```

129         cover(in[i], out[i], i);
130     }
131 }
132 SegmentTree<Max> seg(M);
133 auto add = [&](int i) {
134     // for (int j = i; j != -1; j = pre[j]) {
135     //     if (f[j] < f[i] || i == j) {
136     //         f[j] = f[i];
137     //         if (pre[j] == -1) {
138     //             modify(1, 0, n, l[j], f[j] - 1);
139     //         }
140     //     }
141     // }
142     int j = get(in[i]);
143     int res = seg.rangeQuery(in[j], out[j]).x;
144     seg.modify(in[i], r[i]);
145     int nres = seg.rangeQuery(in[j], out[j]).x;
146     if (nres > res) {
147         modify(1, 0, n, l[j], nres - 1);
148     }
149     if (pre[i] == -1) {
150         cover(in[i], out[i], i);
151     }
152 };
153 auto rem = [&](int i) {
154     modify(1, 0, n, l[i], l[i]);
155     for (auto j : adj[i]) {
156         modify(1, 0, n, l[j], seg.rangeQuery(in[j], out[j]).x - 1);
157         cover(in[j], out[j], j);
158     }
159 };
160 for (int i = 0; i < k; i++) {
161     add(i);
162 }
163 auto query = [&]() {
164     int v = -1;
165     return ::query(1, 0, n, 0, n, v);
166 };
167 for (int i = 0; i < m; i++) {
168     cout << query() << " \n"[i == m - 1];
169     rem(i);
170     add(i + k);
171 }
172 return 0;
173 }
```

### 931: MEX Tree Manipulation

- Time limit: 5 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Given a rooted tree, define the value of vertex  $u$  in the tree recursively as the MEX<sup>†</sup> of the values of its children. Note that it is only the children, not all of its descendants. In particular, the value of a leaf is 0.

Pak Chanek has a rooted tree that initially only contains a single vertex with index 1, which is the root. Pak Chanek is going to do  $q$  queries. In the  $i$ -th query, Pak Chanek is given an integer  $x_i$ . Pak Chanek needs to add a new vertex with index  $i + 1$  as the child of vertex  $x_i$ . After adding the new vertex, Pak Chanek needs to recalculate the values of all vertices and report the sum of the values of all vertices in the current tree.

<sup>†</sup> The MEX (minimum excluded) of an array is the smallest non-negative integer that does not belong to the array. For example, the MEX of  $[0, 1, 1, 2, 6, 7]$  is 3 and the MEX of  $[6, 9]$  is 0.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct Info;
5 Info operator+(const Info &a, const Info &b) {
6     Info c;
7     c.in = a.in;
8     if (a.out[1] == b.in) {
9         c.out[1] = b.out[1];
10        c.sum[1] = a.sum[1] + b.sum[1];
11    } else {
12        c.out[1] = b.out[0];
13        c.sum[1] = a.sum[1] + b.sum[0];
14    }
15    if (a.out[0] == b.in) {
16        c.out[0] = b.out[1];
17        c.sum[0] = a.sum[0] + b.sum[1];
18    } else {
19        c.out[0] = b.out[0];
20        c.sum[0] = a.sum[0] + b.sum[0];
21    }
22    return c;
23 }
24 int main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
27     int n;
28     cin >> n;
29     n++;
30     vector<int> p(n * 2, -1), parent(n, -1);
31     for (int i = 1; i < n; i++) {
32         cin >> parent[i];
33         parent[i]--;
34     }
35     vector<int> ch1(2 * n), ch2(2 * n);
36     vector<array<int, 20>> cnt(n);
37     vector<int> ans(2 * n, -1), sum(2 * n), lsum(2 * n);
38     vector<Info> info(2 * n);
39     function<int(int)> update = [&](int x) {
40         int in = 0;
41         while (cnt[x][in]) {
42             in++;
43         }
44         int out = in + 1;
45         while (cnt[x][out]) {

```

```

46         out++;
47     }
48     info[x] = Info(in, out, in, lsum[x]);
49     int t = x;
50     while (p[t] >= n) {
51         t = p[t];
52         info[t] = info[ch1[t]] + info[ch2[t]];
53     }
54     if (p[t] >= 0 && ans[t] != -1) {
55         lsum[p[t]] -= sum[t];
56         cnt[p[t]][ans[t]]--;
57     }
58     ans[t] = info[t].out[0];
59     sum[t] = info[t].sum[0];
60     if (p[t] >= 0) {
61         lsum[p[t]] += sum[t];
62         cnt[p[t]][ans[t]]++;
63         return update(p[t]);
64     }
65     return sum[t];
66 };
67 int cur = n;
68 vector<vector<int>> adj(n);
69 for (int i = 1; i < n; i++) {
70     adj[parent[i]].push_back(i);
71 }
72 vector<int> siz(n);
73 function<void(int)> dfs1 = [&](int x) {
74     siz[x] = 1;
75     for (auto &y : adj[x]) {
76         dfs1(y);
77         siz[x] += siz[y];
78         if (siz[y] > siz[adj[x][0]]) {
79             swap(y, adj[x][0]);
80         }
81     }
82 };
83 dfs1(0);
84 function<int(int, int)> work = [&](int l, int r) {
85     if (r == (adj[l].empty() ? -1 : adj[l][0])) {
86         for (auto x : adj[l]) {
87             if (x != adj[l][0]) {
88                 p[work(x, -1)] = l;
89             }
90         }
91         return l;
92     }
93     int x = l;
94     while (r != (adj[x].empty() ? -1 : adj[x][0]) && siz[x] * 2 >= siz[l] + (r >= 0 ?
95         siz[r] : 0)) {
96         x = adj[x][0];
97     }
98     int a = work(l, x);
99     int b = work(x, r);
100    int c = cur++;
101    ch1[c] = b;
102    ch2[c] = a;
103    p[a] = p[b] = c;
104    return c;
105 };
106 work(0, -1);
107 update(0);
108 for (int i = 1; i < n; i++) {
109     cout << update(i) << "\n";

```

```

109     }
110     return 0;
111 }
```

**dp****932: Optimizations From Chelsu**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You are given a tree with  $n$  vertices, whose vertices are numbered from 1 to  $n$ . Each edge is labeled with some integer  $w_i$ .

Define  $\text{len}(u, v)$  as the number of edges in the simple path between vertices  $u$  and  $v$ , and  $\text{gcd}(u, v)$  as the Greatest Common Divisor of all numbers written on the edges of the simple path between vertices  $u$  and  $v$ . For example,  $\text{len}(u, u) = 0$  and  $\text{gcd}(u, u) = 0$  for any  $1 \leq u \leq n$ .

You need to find the maximum value of  $\text{len}(u, v) \cdot \text{gcd}(u, v)$  over all pairs of vertices in the tree.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void solve() {
5     int n;
6     cin >> n;
7     vector<vector<pair<int, i64>>> adj(n);
8     for (int i = 1; i < n; i++) {
9         int u, v;
10        i64 w;
11        cin >> u >> v >> w;
12        u--, v--;
13        adj[u].emplace_back(v, w);
14        adj[v].emplace_back(u, w);
15    }
16    vector<int> vis(n), dep(n), siz(n), t(n);
17    vector<i64> g(n);
18    auto dfs1 = [&](auto self, int x, int p) -> void {
19        siz[x] = 1;
20        for (auto [y, w] : adj[x]) {
21            if (vis[y] || y == p) {
22                continue;
23            }
24            self(self, y, x);
25            siz[x] += siz[y];
26        }
27    };
28    int ans = 0;
29    for (int i = 1; i < n; i++) {
30        if (!vis[i]) {
31            vis[i] = 1;
32            t[i] = i;
33            dfs1(self, i, -1);
34            for (int j = 0; j < n; j++) {
35                if (vis[j]) {
36                    g[j] = t[j];
37                }
38            }
39            for (int j = 0; j < n; j++) {
40                if (vis[j]) {
41                    int d = dep[j] - dep[i];
42                    i64 gcd = __gcd(g[i], g[j]);
43                    ans = max(ans, d * gcd);
44                }
45            }
46            for (int j = 0; j < n; j++) {
47                if (vis[j]) {
48                    vis[j] = 0;
49                }
50            }
51        }
52    }
53    cout << ans;
54}
```

```

26         }
27     };
28     auto find = [&](auto self, int x, int p, int s) -> int {
29         for (auto [y, w] : adj[x]) {
30             if (vis[y] || y == p || 2 * siz[y] <= s) {
31                 continue;
32             }
33             return self(self, y, x, s);
34         }
35         return x;
36     };
37     i64 ans = 0;
38     vector<i64> vals;
39     vector<array<int, 2>> mx, bel;
40     auto dfs2 = [&](auto self, int x, int p, int r) -> void {
41         ans = max(ans, g[x] * dep[x]);
42         if (g[x] != 0) {
43             vals.push_back(g[x]);
44         }
45         for (auto [y, w] : adj[x]) {
46             if (vis[y] || y == p) {
47                 continue;
48             }
49             g[y] = gcd(g[x], w);
50             t[y] = x == r ? y : t[x];
51             dep[y] = dep[x] + 1;
52             self(self, y, x, r);
53         }
54     };
55     auto dfs3 = [&](auto self, int x, int p) -> void {
56         if (g[x] != 0) {
57             int i = lower_bound(vals.begin(), vals.end(), g[x]) - vals.begin();
58             if (dep[x] > mx[i][0]) {
59                 if (bel[i][0] != t[x]) {
60                     mx[i][1] = mx[i][0];
61                     bel[i][1] = bel[i][0];
62                 }
63                 mx[i][0] = dep[x];
64                 bel[i][0] = t[x];
65             } else if (dep[x] > mx[i][1] && t[x] != bel[i][0]) {
66                 mx[i][1] = dep[x];
67                 bel[i][1] = t[x];
68             }
69         }
70         for (auto [y, w] : adj[x]) {
71             if (vis[y] || y == p) {
72                 continue;
73             }
74             self(self, y, x);
75         }
76     };
77     auto CD = [&](auto self, int x) -> void {
78         dfs1(dfs1, x, -1);
79         x = find(find, x, -1, siz[x]);
80         vis[x] = 1;
81         g[x] = 0;
82         t[x] = x;
83         dep[x] = 0;
84         vals.clear();
85         dfs2(dfs2, x, -1, x);
86         sort(vals.begin(), vals.end());
87         vals.erase(unique(vals.begin(), vals.end()), vals.end());
88         mx.assign(vals.size(), {0, 0});
89         bel.assign(vals.size(), {-1, -1});

```

```

90         dfs3(dfs3, x, -1);
91         for (int i = 0; i < vals.size(); i++) {
92             i64 g0 = vals[i];
93             for (int j = 0; j < 2; j++) {
94                 int l0 = mx[i][j];
95                 for (int k = 1; k <= l0 && (l0 + l0 / max(1, k - 1)) * g0 > ans; k++) {
96                     i64 g1 = g0 * k;
97                     int i1 = lower_bound(vals.begin(), vals.end(), g1) - vals.begin();
98                     if (i1 < vals.size() && vals[i1] == g1) {
99                         int l1 = mx[i1][bel[i1][0]] == bel[i][j];
100                        ans = max(ans, g0 * (l0 + l1));
101                    }
102                }
103            }
104        }
105        for (auto [y, w] : adj[x]) {
106            if (vis[y]) {
107                continue;
108            }
109            self(self, y);
110        }
111    };
112    CD(CD, 0);
113    cout << ans << "\n";
114 }
115 int main() {
116     ios::sync_with_stdio(false);
117     cin.tie(nullptr);
118     int t;
119     cin >> t;
120     while (t--) {
121         solve();
122     }
123     return 0;
124 }
```

### 933: Ball-Stackable

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

With a problem title like that, there is no way this is going to be a graph problem.

Chaneka has a graph with  $n$  vertices and  $n - 1$  edges. Some of the edges are directed and some of the edges are undirected. Edge  $i$  connects vertex  $u_i$  to vertex  $v_i$ . If  $t_i = 0$ , edge  $i$  is undirected. If  $t_i = 1$ , edge  $i$  is directed in the direction from  $u_i$  to  $v_i$ . It is known that if you make all edges undirected, the graph becomes a tree<sup>†</sup>.

Chaneka wants to direct all undirected edges and colour each edge (different edges can have the same colour).

After doing that, suppose Chaneka starts a walk from an arbitrary vertex  $x$  to an arbitrary vertex  $y$  (it is

possible that  $x = y$ ) going through one or more edges. She is allowed to go through each edge either following the direction or opposite to the direction of the edge. She is also allowed to visit a vertex or an edge more than once. During the walk, Chaneka maintains a stack of balls that is initially empty before the walk. Each time Chaneka goes through an edge, she does the following:

If Chaneka goes through it in the right direction, she puts a new ball with a colour that is the same as the edge's colour to the top of the stack.

If Chaneka goes through it in the opposite direction, she removes the ball that is on the top of the stack.

A walk is stackable if and only if the stack is not empty before each time Chaneka goes through an edge in the opposite direction.

A walk is ball-stackable if and only if it is stackable and each time Chaneka goes through an edge in the opposite direction, the colour of the ball removed from the stack is the same as the colour of the edge traversed.

Is it possible to direct all undirected edges and colour each edge such that all stackable walks are also ball-stackable? If it is possible, find a construction example that uses the maximum number of different colours among all valid ways of directing and colouring. If there are multiple such solutions, output any of them.

<sup>†</sup> A tree is a connected graph with no cycles.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct DSU;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n;
9     cin >> n;
10    vector<vector<pair<int, int>>> adj(n);
11    for (int i = 1; i < n; i++) {
12        int u, v, t;
13        cin >> u >> v >> t;
14        u--, v--;
15        adj[u].emplace_back(v, t);
16        adj[v].emplace_back(u, -t);
17    }
18    vector<int> h(n);
19    auto dfs = [&](auto self, int x, int p) -> void {
20        for (auto [y, t] : adj[x]) {
21            if (p == y) {
22                continue;
23            }
24            h[y] = h[x] + t;
25            self(self, y, x);

```

```
26         }
27     };
28     dfs(dfs, 0, -1);
29     int r = min_element(h.begin(), h.end()) - h.begin();
30     auto dfs1 = [&](auto self, int x, int p) -> void {
31         for (auto [y, t] : adj[x]) {
32             if (p == y) {
33                 continue;
34             }
35             if (t == 0) {
36                 t = 1;
37             }
38             h[y] = h[x] + t;
39             self(self, y, x);
40         }
41     };
42     h[r] = 0;
43     dfs1(dfs1, r, -1);
44     int ans = 0;
45     vector<int> p(n);
46     iota(p.begin(), p.end(), 0);
47     sort(p.begin(), p.end(),
48          [&](int i, int j) {
49              return h[i] > h[j];
50          });
51     vector<array<int, 3>> edges;
52     DSU dsu(n);
53     for (int l = 0, r = 0; l < n; l = r) {
54         while (r < n && h[p[l]] == h[p[r]]) {
55             r++;
56         }
57         map<int, int> f;
58         for (int i = l; i < r; i++) {
59             int x = p[i];
60             for (auto [y, t] : adj[x]) {
61                 if (h[y] < h[x]) {
62                     continue;
63                 }
64                 int z = dsu.find(y);
65                 if (!f.count(z)) {
66                     f[z] = ++ans;
67                 }
68                 edges.push_back({x + 1, y + 1, f[z]});
69             }
70         }
71         for (int i = l; i < r; i++) {
72             int x = p[i];
73             for (auto [y, t] : adj[x]) {
74                 if (h[y] < h[x]) {
75                     continue;
76                 }
77                 dsu.merge(y, x);
78             }
79         }
80     }
81     cout << ans << "\n";
82     for (auto [x, y, z] : edges) {
83         cout << x << " " << y << " " << z << "\n";
84     }
85     return 0;
86 }
```

## 934: Jellyfish and Incription

- Time limit: 2 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

Jellyfish loves playing a game called “Incription” which is played on a directed acyclic graph with  $n$  vertices and  $m$  edges. All edges  $a \rightarrow b$  satisfy  $a < b$ .

You need to move from vertex 1 to vertex  $n$  along the directed edges, and then fight with the final boss.

You will collect cards and props in the process.

Each card has two attributes: HP and damage. If a card’s HP is  $a$  and its damage is  $b$ , then the power of the card is  $a \times b$ .

Each prop has only one attribute: power.

In addition to vertex 1 and vertex  $n$ , there are some vertices that trigger special events. The special events are:

You will get a card with  $a$  HP, and  $b$  damage.

If you have at least one card, choose one of your cards and increase its HP by  $x$ .

If you have at least one card, choose one of your cards and increase its damage by  $y$ .

You will get a prop with  $w$  power.

When you get to vertex  $n$ , you can choose at most one of your cards and multiply its damage by  $10^9$ .

The final boss is very strong, so you want to maximize the sum of the power of all your cards and props. Find the maximum possible sum of power of all your cards and props if you play the game optimally.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int V = 200;
5 constexpr int inf = 1E9;
6 template<class T>
7 void chmax(T &a, T b) {
8     if (a < b) {
9         a = b;
10    }
11 }
12 int main() {

```

```

13 ios::sync_with_stdio(false);
14 cin.tie(nullptr);
15 int n, m;
16 cin >> n >> m;
17 vector<array<int, 3>> e(n);
18 for (int i = 0; i < n; i++) {
19     int o;
20     cin >> o;
21     e[i][0] = o;
22     if (o == 1) {
23         cin >> e[i][1] >> e[i][2];
24     } else if (o > 1) {
25         cin >> e[i][1];
26     }
27 }
28 vector<vector<int>> adj(n);
29 for (int i = 0; i < m; i++) {
30     int u, v;
31     cin >> u >> v;
32     u--;
33     v--;
34     adj[u].push_back(v);
35 }
36 vector f(n, vector(V + 1, vector<int>(V + 1, -inf)));
37 vector g1(n, vector(V + 1, vector<int>(V + 1, -inf)));
38 vector g2(n, vector(V + 1, vector<int>(V + 1, -inf)));
39 vector h1(n, vector(V + 1, vector<int>(V + 1, -inf)));
40 vector h2(n, vector(V + 1, vector<int>(V + 1, -inf)));
41 vector<int> dp(n);
42 for (int i = 0; i < n; i++) {
43     for (int a = V; a >= 0; a--) {
44         for (int b = V; b >= 0; b--) {
45             chmax(dp[i], f[i][a][b]);
46         }
47     }
48     for (int a = 0; a <= V; a++) {
49         chmax(dp[i], g1[i][a][a]);
50         chmax(dp[i], g2[i][a][a]);
51         chmax(f[i][a][0], g1[i][a][a]);
52         chmax(f[i][0][a], g2[i][a][a]);
53         chmax(dp[i], h1[i][0][a]);
54         chmax(dp[i], h2[i][0][a]);
55         chmax(f[i][a][0], h1[i][0][a]);
56         chmax(f[i][0][a], h2[i][0][a]);
57         for (int b = a; b <= V; b++) {
58             chmax(h1[i][b - a][0], g1[i][a][b]);
59             chmax(h2[i][b - a][0], g2[i][a][b]);
60         }
61     }
62     chmax(f[i][0][0], dp[i]);
63     for (int a = V; a >= 0; a--) {
64         for (int b = V; b >= 0; b--) {
65             if (e[i][0] == 1) {
66                 chmax(f[i][max(a, e[i][1])][max(b, e[i][2])], f[i][a][b] + e[i][1] * e[i][2]);
67             } else if (e[i][0] == 2) {
68                 f[i][a][b] += b * e[i][1];
69             } else if (e[i][0] == 3) {
70                 f[i][a][b] += a * e[i][1];
71             } else if (e[i][0] == 4) {
72                 f[i][a][b] += e[i][1];
73             }
74         }
75     }
76     if (e[i][0] == 1) {
77         f[i][0][0] = 0;
78     }
79 }
80

```

```

76         for (int a = V; a >= 0; a--) {
77             for (int b = V; b >= a; b--) {
78                 g1[i][a][b] += e[i][1] * e[i][2];
79                 g2[i][a][b] += e[i][1] * e[i][2];
80             }
81         }
82         for (int b = e[i][1]; b <= V; b++) {
83             chmax(g1[i][e[i][1]][b], dp[i] + e[i][2] * b);
84         }
85         for (int b = e[i][2]; b <= V; b++) {
86             chmax(g2[i][e[i][2]][b], dp[i] + e[i][1] * b);
87         }
88         for (int a = 0; a <= V; a++) {
89             for (int b = V; b >= 0; b--) {
90                 chmax(h1[i][a][max(b, e[i][1])], h1[i][a][b] + e[i][1] * e[i][2]);
91                 chmax(h2[i][a][max(b, e[i][2])], h2[i][a][b] + e[i][1] * e[i][2]);
92             }
93         }
94     } else if (e[i][0] == 2) {
95         for (int a = V; a >= 0; a--) {
96             for (int b = V; b >= a; b--) {
97                 if (a + e[i][1] <= b) {
98                     chmax(g1[i][a + e[i][1]][b], g1[i][a][b]);
99                 }
100                g2[i][a][b] += e[i][1] * b;
101            }
102        }
103        for (int a = 0; a <= V; a++) {
104            for (int b = V; b >= 0; b--) {
105                if (a >= e[i][1]) {
106                    chmax(h1[i][a - e[i][1]][b], h1[i][a][b]);
107                }
108                h2[i][a][b] += e[i][1] * b;
109            }
110        }
111    } else if (e[i][0] == 3) {
112        for (int a = V; a >= 0; a--) {
113            for (int b = V; b >= a; b--) {
114                if (a + e[i][1] <= b) {
115                    chmax(g2[i][a + e[i][1]][b], g2[i][a][b]);
116                }
117                g1[i][a][b] += e[i][1] * b;
118            }
119        }
120        for (int a = 0; a <= V; a++) {
121            for (int b = V; b >= 0; b--) {
122                if (a >= e[i][1]) {
123                    chmax(h2[i][a - e[i][1]][b], h2[i][a][b]);
124                }
125                h1[i][a][b] += e[i][1] * b;
126            }
127        }
128    } else if (e[i][0] == 4) {
129        for (int a = V; a >= 0; a--) {
130            for (int b = V; b >= a; b--) {
131                g1[i][a][b] += e[i][1];
132                g2[i][a][b] += e[i][1];
133            }
134        }
135        for (int a = 0; a <= V; a++) {
136            for (int b = V; b >= 0; b--) {
137                h1[i][a][b] += e[i][1];
138                h2[i][a][b] += e[i][1];
139            }
}

```

```

140         }
141     }
142     for (auto j : adj[i]) {
143         for (int a = V; a >= 0; a--) {
144             for (int b = V; b >= 0; b--) {
145                 chmax(f[j][a][b], f[i][a][b]);
146                 chmax(h1[j][a][b], h1[i][a][b]);
147                 chmax(h2[j][a][b], h2[i][a][b]);
148             }
149         }
150         for (int a = V; a >= 0; a--) {
151             for (int b = V; b >= a; b--) {
152                 chmax(g1[j][a][b], g1[i][a][b]);
153                 chmax(g2[j][a][b], g2[i][a][b]);
154             }
155         }
156     }
157 }
158 vector<pair<int, int>> suf(n, vector<pair<int, int>>(n * V + 1, {-inf, -inf}));
159 suf[n - 1][0] = {0, 0};
160 for (int i = n - 1; i >= 0; i--) {
161     for (auto j : adj[i]) {
162         for (int a = 0; a <= n * V; a++) {
163             chmax(suf[i][a], suf[j][a]);
164         }
165     }
166     if (e[i][0] == 1) {
167         for (int a = 0; a <= n * V; a++) {
168             suf[i][a].second += e[i][1] * e[i][2];
169         }
170     } else if (e[i][0] == 2) {
171         for (int a = n * V; a >= 0; a--) {
172             if (a + e[i][1] <= n * V) {
173                 chmax(suf[i][a + e[i][1]], suf[i][a]);
174             }
175         }
176     } else if (e[i][0] == 3) {
177         for (int a = 0; a <= n * V; a++) {
178             suf[i][a].first += e[i][1];
179         }
180     } else if (e[i][0] == 4) {
181         for (int a = 0; a <= n * V; a++) {
182             suf[i][a].second += e[i][1];
183         }
184     }
185 }
186 i64 ans = dp[n - 1];
187 for (int i = 0; i < n; i++) {
188     if (e[i][0] == 1) {
189         i64 val = 0;
190         for (int a = 0; a <= n * V; a++) {
191             if (suf[i][a].first >= 0 && suf[i][a].second >= 0) {
192                 int b = suf[i][a].first;
193                 int t = suf[i][a].second - e[i][1] * e[i][2];
194                 chmax(val, t + 1LL * inf * (e[i][1] + a) * (e[i][2] + b));
195             }
196         }
197         chmax(ans, dp[i] + val);
198     }
199 }
200 cout << ans << "\n";
201 return 0;
202 }

```

### 935: Min-Sum-Max

- Time limit: 5 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

Tom is waiting for his results of Zhongkao examination. To ease the tense atmosphere, his friend, Daniel, decided to play a game with him. This game is called “Divide the Array”.

The game is about the array  $a$  consisting of  $n$  integers. Denote  $[l, r]$  as the subsegment consisting of integers  $a_l, a_{l+1}, \dots, a_r$ .

Tom will divide the array into contiguous subsegments  $[l_1, r_1], [l_2, r_2], \dots, [l_m, r_m]$ , such that each integer is in exactly one subsegment. More formally:

For all  $1 \leq i \leq m$ ,  $1 \leq l_i \leq r_i \leq n$ ;

$l_1 = 1, r_m = n$ ;

For all  $1 < i \leq m$ ,  $l_i = r_{i-1} + 1$ .

Denote  $s_i = \sum_{k=l_i}^{r_i} a_k$ , that is,  $s_i$  is the sum of integers in the  $i$ -th subsegment. For all  $1 \leq i \leq j \leq m$ , the following condition must hold:

$$\min_{i \leq k \leq j} s_k \leq \sum_{k=i}^j s_k \leq \max_{i \leq k \leq j} s_k.$$

Tom believes that the more subsegments the array  $a$  is divided into, the better results he will get. So he asks Daniel to find the maximum number of subsegments among all possible ways to divide the array  $a$ . You have to help him find it.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int inf = 1E9;
5 void solve() {
6     int n;
7     cin >> n;
8     vector<int> a(n);
9     for (int i = 0; i < n; i++) {
10         cin >> a[i];
11     }
12     vector<i64> s(n + 1);
13     for (int i = 0; i < n; i++) {
14         s[i + 1] = s[i] + a[i];

```

```

15     }
16     vector<int> pre(n + 1, vector<int>(n + 1, -inf)), suf(pre);
17     vector<int> fpre(pre), fsuf(pre);
18     int ans = 1;
19     if (s[n] == 0) {
20         ans = count(s.begin(), s.end() - 1, 0);
21     }
22     vector<int> left(n + 1, vector<int>(n + 1)), right(n + 1, vector<int>(n + 1));
23     for (int i = n; i >= 0; i--) {
24         for (int j = i + 1; j <= n; j++) {
25             right[i][j] = s[i] == s[j - 1] ? j - 1 : right[i][j - 1];
26             left[i][j] = s[i + 1] == s[j] ? i + 1 : left[i + 1][j];
27         }
28     }
29     vector<int> next(n + 1, n + 1);
30     for (int i = 0; i <= n; i++) {
31         for (int j = i + 1; j <= n; j++) {
32             if (s[i] == s[j]) {
33                 next[i] = j;
34                 break;
35             }
36         }
37     }
38     vector<i64> dp(2, vector(n + 1, vector(n + 1, vector<i64>(n, 1E18))));
39     for (int i = n; i >= 0; i--) {
40         for (int j = i + 1; j <= n; j++) {
41             if (s[i] != s[j]) {
42                 continue;
43             }
44             for (int t = 0; t < 2; t++) {
45                 dp[t][i][j][0] = 0;
46                 for (int m = i + 1; m < j; m++) {
47                     if (s[i] == s[m] && (m == next[i] || m == right[i][j])) {
48                         auto &u = dp[t][i][m];
49                         auto &v = dp[t][m][j];
50                         int x = 0, y = 0;
51                         while (x + y < n) {
52                             if (u[x] < v[y]) {
53                                 dp[t][i][j][x + y] = min(dp[t][i][j][x + y], u[x]);
54                                 x++;
55                             } else {
56                                 dp[t][i][j][x + y] = min(dp[t][i][j][x + y], v[y]);
57                                 y++;
58                             }
59                         }
60                     }
61                 }
62                 for (int m = i + 1; m < j; m++) {
63                     if ((s[m] > s[i]) == t) {
64                         auto &v = dp[!t][m][right[m][j]];
65                         int val = upper_bound(v.begin(), v.end(), abs(s[i] - s[m])) - v.begin();
66                         dp[t][i][j][val + 1] = min(dp[t][i][j][val + 1], abs(s[i] - s[m]));
67                     }
68                 }
69                 for (int x = n - 2; x >= 0; x--) {
70                     dp[t][i][j][x] = min(dp[t][i][j][x], dp[t][i][j][x + 1]);
71                 }
72             }
73         }
74     }
75     for (int i = 0; i <= n; i++) {
76         for (int j = i + 1; j <= n; j++) {

```

```

77         if (s[i] == s[j]) {
78             continue;
79         }
80         if (i == 0) {
81             pre[i][j] = 0;
82         }
83         auto &v = dp[s[j] > s[i]][i][right[i][j]];
84         pre[i][j] += upper_bound(v.begin(), v.end(), abs(s[i] - s[j])) - v.begin() +
85             1;
86         for (int k = j + 1; k <= n; k++) {
87             i64 s1 = s[j] - s[i];
88             i64 s2 = s[k] - s[j];
89             if ((s1 < 0 && s2 > 0 && s2 >= -s1) || (s1 > 0 && s2 < 0 && -s2 >= s1)) {
90                 pre[j][k] = max(pre[j][k], pre[i][j]);
91             }
92             int p = right[i][j];
93             fpre[p][j] = max(fpre[p][j], pre[i][j]);
94         }
95     }
96     for (int k = n; k >= 0; k--) {
97         for (int j = k - 1; j >= 0; j--) {
98             if (s[k] == s[j]) {
99                 continue;
100            }
101            if (k == n) {
102                suf[j][k] = 0;
103            }
104            auto &v = dp[s[j] > s[k]][left[j][k]][k];
105            suf[j][k] += upper_bound(v.begin(), v.end(), abs(s[k] - s[j])) - v.begin() +
106                1;
107            for (int i = j - 1; i >= 0; i--) {
108                i64 s1 = s[k] - s[j];
109                i64 s2 = s[j] - s[i];
110                if ((s1 < 0 && s2 > 0 && s2 >= -s1) || (s1 > 0 && s2 < 0 && -s2 >= s1)) {
111                    suf[i][j] = max(suf[i][j], suf[j][k]);
112                }
113                int p = left[j][k];
114                fsuf[j][p] = max(fsuf[j][p], suf[j][k]);
115            }
116        }
117        for (int i = 0; i <= n; i++) {
118            for (int j = i + 1; j <= n; j++) {
119                ans = max(ans, fpre[i][j] + fsuf[i][j] - 1);
120            }
121        }
122        // int realans = 0;
123        // for (int t = 0; t < (1 << (n - 1)); t++) {
124        //     int lst = 0;
125        //     vector<i64> u;
126        //     for (int i = 1; i <= n; i++) {
127        //         if (i == n || (t >> (i - 1) & 1)) {
128        //             u.push_back(s[i] - s[lst]);
129        //             lst = i;
130        //         }
131        //     }
132        //     bool ok = true;
133        //     for (int i = 0; i < u.size(); i++) {
134        //         i64 sum = 0;
135        //         i64 min = 0;
136        //         i64 max = 0;
137        //         for (int j = i; j < u.size(); j++) {
138        //             sum += u[j];

```

```

139     //             max = max(max, u[j]);
140     //             min = min(min, u[j]);
141     //             if (sum < min || sum > max) {
142     //                 ok = false;
143     //             }
144     //         }
145     //     if (ok) {
146     //         realans = max(realans, __builtin_popcount(t) + 1);
147     //         if (realans == 10) {
148     //             for (int i = 0; i < n - 1; i++) {
149     //                 cerr << (t >> i & 1);
150     //             }
151     //             cerr << "\n";
152     //             break;
153     //         }
154     //     }
155     // }
156     // }
157     // cerr << dp[1][7][10][1] << "\n";
158     // cerr << dp[0][5][11][3] << "\n";
159     // cerr << dp[1][4][12][5] << "\n";
160     cout << ans << "\n";
161 }
162 int main() {
163     ios::sync_with_stdio(false);
164     cin.tie(nullptr);
165     int t;
166     cin >> t;
167     while (t--) {
168         solve();
169     }
170     return 0;
171 }
```

### 936: Redundant Routes

- Time limit: 5 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

You are given a tree with  $n$  vertices labeled  $1, 2, \dots, n$ . The length of a simple path in the tree is the number of vertices in it.

You are to select a set of simple paths of length at least 2 each, but you cannot simultaneously select two distinct paths contained one in another. Find the largest possible size of such a set.

Formally, a set  $S$  of vertices is called a route if it contains at least two vertices and coincides with the set of vertices of a simple path in the tree. A collection of distinct routes is called a timetable. A route  $S$  in a timetable  $T$  is called redundant if there is a different route  $S' \in T$  such that  $S \subset S'$ . A timetable is called efficient if it contains no redundant routes. Find the largest possible number of routes in an efficient timetable.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<vector<int>> adj(n);
10    for (int i = 1; i < n; i++) {
11        int u, v;
12        cin >> u >> v;
13        u--, v--;
14        adj[u].push_back(v);
15        adj[v].push_back(u);
16    }
17    vector<int> p(n, -1), d(n, -1);
18    vector id(n, vector<int>(n));
19    int cur = n;
20    for (int i = 0; i < n; i++) {
21        for (int j = i + 1; j < n; j++) {
22            id[i][j] = id[j][i] = cur++;
23        }
24    }
25    vector<int> S(cur), L(cur), P(cur);
26    for (int i = 1; i < n; i++) {
27        L[i] = i;
28        P[i] = i == 1 ? -1 : i - 1;
29    }
30    vector dis(n, vector<int>(n));
31    auto bfs = [&](int s) {
32        d.assign(n, -1);
33        p.assign(n, -1);
34        queue<int> q;
35        q.push(s);
36        d[s] = 0;
37        while (!q.empty()) {
38            int x = q.front();
39            q.pop();
40            for (auto y : adj[x]) {
41                if (d[y] == -1) {
42                    d[y] = d[x] + 1;
43                    p[y] = x;
44                    q.push(y);
45                }
46            }
47        }
48        return max_element(d.begin(), d.end()) - d.begin();
49    };
50    for (int i = 0; i < n; i++) {
51        bfs(i);
52        for (int j = i + 1; j < n; j++) {
53            S[d[j]]++;
54        }
55        swap(d, dis[i]);
56    }
57    int s = bfs(0);
58    int t = bfs(s);
59    int diameter = d[t];
60    int rt = t;

```

```

61     for (int i = 0; i < diameter / 2; i++) {
62         rt = p[rt];
63     }
64     vector<int> rts;
65     if (diameter % 2 == 0) {
66         rts.push_back(rt);
67     } else {
68         rts.push_back(rt);
69         rts.push_back(p[rt]);
70         adj[rt].erase(find(adj[rt].begin(), adj[rt].end(), p[rt]));
71         adj[p[rt]].erase(find(adj[p[rt]].begin(), adj[p[rt]].end(), rt));
72     }
73     vector<int> h(n);
74     vector<vector<int>> cnt(n, vector<int>(n));
75     p.assign(n, -1);
76     auto dfs = [&](auto self, int x) -> void {
77         cnt[x][0]++;
78         for (auto y : adj[x]) {
79             if (y == p[x]) {
80                 continue;
81             }
82             p[y] = x;
83             self(self, y);
84             h[x] = max(h[x], h[y] + 1);
85             for (int i = 0; i <= h[y]; i++) {
86                 cnt[x][i + 1] += cnt[y][i];
87             }
88         };
89     };
90     for (auto r : rts) {
91         dfs(dfs, r);
92     }
93     for (int i = 0; i < n; i++) {
94         for (int j = i + 1; j < n; j++) {
95             if (h[i] == h[j]) {
96                 L[id[i][j]] = h[i] + dis[i][j];
97             }
98         }
99     }
100    for (int i = 0; i < n; i++) {
101        vector<int> have(n, -1);
102        for (int j = p[i]; j != -1; j = p[j]) {
103            have[h[j]] = j;
104        }
105        auto dfs = [&](auto self, int x, int len, int t) -> void {
106            if (h[x] == h[i] && i != x) {
107                P[id[i][x]] = t;
108                // cerr << i << " " << x << " " << t << "\n";
109            }
110            if (h[x] < len && have[h[x]] != -1 && have[h[x]] != x) {
111                len = h[x];
112                t = id[have[h[x]]][x];
113            }
114            for (auto y : adj[x]) {
115                if (y == p[x]) {
116                    continue;
117                }
118                self(self, y, len, t);
119            }
120        };
121        for (auto r : rts) {
122            dfs(dfs, r, n, n - 1);
123        }
124    }

```

```

125     for (int i = 0; i < n; i++) {
126         for (int j = i + 1; j < n; j++) {
127             if (h[i] == h[j]) {
128                 for (int x = 0; x <= h[i]; x++) {
129                     S[id[i][j]] += cnt[i][x] * cnt[j][h[i] - x];
130                 }
131                 // cerr << i + 1 << " " << j + 1 << " " << L[id[i][j]] << " " << id[i][j]
132                 // << " " << P[id[i][j]] << " " << S[id[i][j]] << "\n";
133             }
134         }
135     vector<vector<int>> e(cur);
136     for (int i = 0; i < cur; i++) {
137         if (P[i] > 0) {
138             e[P[i]].push_back(i);
139         }
140     }
141     vector<int> lst(n), dep(cur);
142     auto dfs1 = [&](auto self, int x) -> void {
143         // cerr << "x : " << x << " " << L[x] << "\n";
144         S[lst[L[x]]] -= S[x];
145         // cerr << x << " " << lst[L[x]] << "\n";
146         int t = exchange(lst[L[x]], x);
147         for (auto y : e[x]) {
148             // cerr << "(" << x << ", " << y << ") \n";
149             dep[y] = dep[x] + 1;
150             self(self, y);
151         }
152         lst[L[x]] = t;
153     };
154     dfs1(dfs1, 1);
155     vector<int> f(cur), ans(cur);
156     iota(f.begin(), f.end(), 0);
157     auto find = [&](int x) {
158         while (x != f[x]) {
159             x = f[x] = f[f[x]];
160         }
161         return x;
162     };
163     vector<int> a{1};
164     for (int i = 2; i < cur; i++) {
165         if (P[i] > 0) {
166             a.push_back(i);
167         }
168     }
169     sort(a.begin(), a.end(),
170           [&](int i, int j) {
171               return L[i] != L[j] ? L[i] > L[j] : dep[i] < dep[j];
172           });
173     vector<bool> vis(cur);
174     for (auto i : a) {
175         // cerr << i << " " << S[i] << "\n";
176         vis[i] = true;
177         int res = 0;
178         for (auto j : e[i]) {
179             if (vis[j]) {
180                 j = find(j);
181                 f[j] = i;
182                 res += ans[j];
183             }
184         }
185         res = max(res, S[i]);
186         if (P[i] != -1 && vis[P[i]]) {
187             int j = find(P[i]);

```

```

188         f[j] = i;
189         res += ans[j];
190     }
191     ans[i] = res;
192 }
193 assert(accumulate(S.begin() + 1, S.end(), 0) == n * (n - 1) / 2);
194 cout << ans[find(1)] << "\n";
195 return 0;
196 }
```

### 937: Goldberg Machine 3

- Time limit: 8 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

There is a complete rooted binary tree, that is, a rooted tree in which each vertex has either 0 or 2 children. The root of the tree is vertex 1. A node without children is called a leaf. Each leaf has a hunger value, we denote the hunger value of leaf  $v$  by  $h_v$ .

Each inner vertex of the tree has a selector pointing to one of the children of the vertex.

This tree accepts cookies. Before launching the process you can choose the initial state of each selector individually. The process is as follows:

Initially there are no cookies in vertices.

You insert cookies into the root one by one.

As long as the cookie is not in a leaf, it falls to the child defined by the selector in the current vertex. This selector then changes its state to the opposite one, i. e. it starts pointing to the other child of the vertex.

You stop inserting cookies when each leaf  $v$  has at least  $h_v$  cookies in it. In this case, we say that the tree is filled up.

You have  $q$  queries. Each query changes the value of  $h_v$  for some leaf  $v$ . You need to print  $q + 1$  numbers, the  $i$ -th of them being the minimum number of cookies required to fill up the machine after  $(i - 1)$  updates if you can pick any initial state for every selector. Since these numbers may be very large, print the answers modulo 998 244 353.

Please note that you can choose the initial state of all selectors independently between queries. However, the queries themselves are not independent: when answering the  $i$ -th query, you should also consider the effect of queries  $1, 2, \dots, i - 1$ .

Problem: [link](#)

Solution: [link](#)

```

1 #pragma GCC optimize("Ofast")
2 #include <bits/stdc++.h>
3 using namespace std;
4 using i64 = long long;
5 template<class T>
6 constexpr T power(T a, i64 b) {
7     T res = 1;
8     for (; b; b /= 2, a *= a) {
9         if (b % 2) {
10             res *= a;
11         }
12     }
13     return res;
14 }
15 constexpr i64 mul(i64 a, i64 b, i64 p) {
16     i64 res = a * b - i64(1.L * a * b / p) * p;
17     res %= p;
18     if (res < 0) {
19         res += p;
20     }
21     return res;
22 }
23 template<i64 P>
24 # struct MLong;
25 template<>
26 i64 MLong<0LL>::Mod = i64(1E18) + 9;
27 template<int P>
28 # struct MInt;
29 template<>
30 int MInt<0>::Mod = 998244353;
31 template<int V, int P>
32 constexpr MInt<P> CInv = MInt<P>(V).inv();
33 constexpr int P = 998244353;
34 using Z = MInt<P>;
35 # struct HLD;
36 using Num = vector<int>;
37 int main() {
38     ios::sync_with_stdio(false);
39     cin.tie(nullptr);
40     int n;
41     cin >> n;
42     vector<int> p(n);
43     p[0] = -1;
44     HLD t(n);
45     for (int i = 1; i < n; i++) {
46         cin >> p[i];
47         p[i]--;
48         t.addEdge(p[i], i);
49     }
50     t.work();
51     vector<int> h(n);
52     for (int i = 0; i < n; i++) {
53         cin >> h[i];
54         h[i]--;
55     }
56     int q;
57     cin >> q;
58     set<pair<Num, int>> s;
59     const int N = n + 100;
60     vector<Z> pw(N + 1);
61     pw[0] = 1;
62     for (int i = 1; i <= N; i++) {

```

```

63         pw[i] = pw[i - 1] * 2;
64     }
65     auto query = [&]() {
66         if (s.empty()) {
67             cout << 0 << "\n";
68             return;
69         }
70         const auto &num = s.rbegin()->first;
71         ans = 1;
72         for (auto i : num) {
73             ans += pw[i];
74         }
75         cout << ans << "\n";
76     };
77     auto add = [&](int x, int h) {
78         Num num;
79         for (int i = 29; i >= 0; i--) {
80             if (h >> i & 1) {
81                 num.push_back(t.dep[x] + i);
82             }
83         }
84         while (true) {
85             auto [it, ok] = s.insert({num, t.in[x]});
86             assert(ok);
87             int dep = -1;
88             int v = -1;
89             if (it != s.begin()) {
90                 auto prv = prev(it);
91                 if (prv->first == num) {
92                     int y = t.seq[prv->second];
93                     if (t.in[x] >= t.out[y]) {
94                         int l = t.lca(y, x);
95                         if (t.dep[l] > dep) {
96                             dep = t.dep[l];
97                             v = l;
98                         }
99                     }
100                }
101            }
102            auto nxt = next(it);
103            if (nxt != s.end()) {
104                if (nxt->first == num) {
105                    int y = t.seq[nxt->second];
106                    if (t.in[y] >= t.out[x]) {
107                        int l = t.lca(y, x);
108                        if (t.dep[l] > dep) {
109                            dep = t.dep[l];
110                            v = l;
111                        }
112                    }
113                }
114            }
115            if (dep == -1) {
116                break;
117            }
118            x = v;
119            num.push_back(dep);
120        }
121    };
122    auto del = [&](int x, int h) {
123        Num num;
124        for (int i = 29; i >= 0; i--) {
125            if (h >> i & 1) {
126                num.push_back(t.dep[x] + i);

```

```

127         }
128     }
129     while (true) {
130         auto it = s.find({num, t.in[x]});
131         assert(it != s.end());
132         int dep = -1;
133         int v = -1;
134         if (it != s.begin()) {
135             auto prv = prev(it);
136             if (prv->first == num) {
137                 int y = t.seq[prv->second];
138                 if (t.in[x] >= t.out[y]) {
139                     int l = t.lca(y, x);
140                     if (t.dep[l] > dep) {
141                         dep = t.dep[l];
142                         v = l;
143                     }
144                 }
145             }
146         }
147         auto nxt = next(it);
148         if (nxt != s.end()) {
149             if (nxt->first == num) {
150                 int y = t.seq[nxt->second];
151                 if (t.in[y] >= t.out[x]) {
152                     int l = t.lca(y, x);
153                     if (t.dep[l] > dep) {
154                         dep = t.dep[l];
155                         v = l;
156                     }
157                 }
158             }
159         }
160         s.erase(it);
161         if (dep == -1) {
162             break;
163         }
164         x = v;
165         num.push_back(dep);
166     }
167 };
168 for (int i = 0; i < n; i++) {
169     if (t.adj[i].empty() && h[i] >= 0) {
170         add(i, h[i]);
171     }
172 }
173 query();
174 for (int _ = 0; _ < q; _++) {
175     int x, y;
176     cin >> x >> y;
177     x--, y--;
178     if (h[x] >= 0) {
179         del(x, h[x]);
180     }
181     h[x] = y;
182     if (h[x] >= 0) {
183         add(x, h[x]);
184     }
185     query();
186 }
187 return 0;
188 }

```

## 938: Asterism Stream

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Bogocubic is playing a game with amenotiomoi. First, Bogocubic fixed an integer  $n$ , and then he gave amenotiomoi an integer  $x$  which is initially equal to 1.

In one move amenotiomoi performs one of the following operations with the same probability:

increase  $x$  by 1;

multiply  $x$  by 2.

Bogocubic wants to find the expected number of moves amenotiomoi has to do to make  $x$  greater than or equal to  $n$ . Help him find this number modulo 998 244 353.

Formally, let  $M = 998\,244\,353$ . It can be shown that the answer can be expressed as an irreducible fraction  $\frac{p}{q}$ , where  $p$  and  $q$  are integers and  $q \not\equiv 0 \pmod{M}$ . Output the integer equal to  $p \cdot q^{-1} \pmod{M}$ . In other words, output such an integer  $y$  that  $0 \leq y < M$  and  $y \cdot q \equiv p \pmod{M}$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>

```

```

29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 Z invpw[61];
36 Z invppw[61];
37 void solve() {
38     i64 n;
39     cin >> n;
40     Z ans = 0;
41     for (int k = 0; (1LL << k) < n; k++) {
42         i64 x = n - 1 - (1LL << k);
43         vector<Z> dp{1};
44         for (int t = 0; t < 60; t++) {
45             for (int i = 0; i <= min(t, k); i++) {
46                 dp.push_back(0);
47                 Z v = invppw[t - i];
48                 for (int j = dp.size() - 2; j >= 0; j--) {
49                     dp[j + 1] += dp[j] * v;
50                 }
51             }
52             dp.push_back(0);
53             for (int j = dp.size() - 2; j >= 0; j--) {
54                 dp[j + 1] += dp[j];
55             }
56             int d = x >> t & 1;
57             vector<Z> g;
58             for (int j = d; j < dp.size(); j += 2) {
59                 g.push_back(dp[j]);
60             }
61             swap(dp, g);
62         }
63         ans += dp[0] * invpw[k];
64     }
65     cout << ans << "\n";
66 }
67 int main() {
68     ios::sync_with_stdio(false);
69     cin.tie(nullptr);
70     for (int i = 0; i <= 60; i++) {
71         invpw[i] = Z(1LL << i).inv();
72         invppw[i] = Z(power(Z(2), 1LL << i)).inv();
73     }
74     int t;
75     cin >> t;
76     while (t--) {
77         solve();
78     }
79     return 0;
80 }
```

**939: Sloth**

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Sloth is bad, mkay? So we decided to prepare a problem to punish lazy guys.

You are given a tree, you should count the number of ways to remove an edge from it and then add an edge to it such that the final graph is a tree and has a perfect matching. Two ways of this operation are considered different if their removed edges or their added edges aren't the same. The removed edge and the added edge can be equal.

A perfect matching is a subset of edges such that each vertex is an endpoint of exactly one of these edges.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 array<int, 4> operator*(const array<int, 4> &a, const array<int, 4> &b) {
5     array<int, 4> c{};
6     c[0] = a[0] * b[0];
7     c[1] = a[0] * b[1] + a[1] * b[0];
8     c[2] = a[0] * b[2] + a[2] * b[0];
9     c[3] = a[0] * b[3] + a[1] * b[2] + a[2] * b[1] + a[3] * b[0];
10    return c;
11 }
12 int main() {
13     ios::sync_with_stdio(false);
14     cin.tie(nullptr);
15     int n;
16     cin >> n;
17     vector<vector<int>> adj(n);
18     for (int i = 1; i < n; i++) {
19         int u, v;
20         cin >> u >> v;
21         u--;
22         v--;
23         adj[u].push_back(v);
24         adj[v].push_back(u);
25     }
26     if (n % 2 == 1) {
27         cout << 0 << "\n";
28         return 0;
29     }
30     i64 ans = 0;
31     vector<array<int, 4>> dp(n), up(n);
32     vector<int> siz(n);
33     auto dfs1 = [&](auto self, int x, int p) -> void {
34         siz[x] = 1;
35         dp[x][0] = 1;
36         for (auto y : adj[x]) {
37             if (y == p) {
38                 continue;
39             }
40             self(self, y, x);
41             siz[x] += siz[y];
42             dp[x] = dp[x] * dp[y];
43         }
44         swap(dp[x][0], dp[x][1]);
45         swap(dp[x][2], dp[x][3]);
46         dp[x][2] += dp[x][1];
47     };

```

```

47     dfs1(dfs1, 0, -1);
48     auto dfs2 = [&](auto self, int x, int p) -> void {
49         int d = adj[x].size();
50         if (d == 0) {
51             return;
52         }
53         vector<array<int, 4>> suf(d), pre(d);
54         suf[d - 1][0] = 1;
55         pre[0][0] = 1;
56         for (int i = 0; i < d - 1; i++) {
57             int y = adj[x][i];
58             pre[i + 1] = pre[i] * (y == p ? up[x] : dp[y]);
59         }
60         for (int i = d - 1; i > 0; i--) {
61             int y = adj[x][i];
62             suf[i - 1] = suf[i] * (y == p ? up[x] : dp[y]);
63         }
64         for (int i = 0; i < d; i++) {
65             int y = adj[x][i];
66             if (y == p) {
67                 continue;
68             }
69             up[y] = pre[i] * suf[i];
70             swap(up[y][0], up[y][1]);
71             swap(up[y][2], up[y][3]);
72             up[y][2] += up[y][1];
73             self(self, y, x);
74         }
75     };
76     dfs2(dfs2, 0, -1);
77     for (int i = 1; i < n; i++) {
78         if (dp[i][0] && up[i][0]) {
79             ans += 1LL * siz[i] * (n - siz[i]);
80         } else {
81             ans += 1LL * dp[i][2] * up[i][2];
82         }
83     }
84     cout << ans << "\n";
85     return 0;
86 }

```

## 940: Panda Meetups

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The red pandas are in town to meet their relatives, the blue pandas! The town is modeled by a number line.

The pandas have already planned their meetup, but the schedule keeps changing. You are given  $q$  updates of the form  $x\ t\ c$ .

If  $c < 0$ , it means  $|c|$  more red pandas enter the number line at position  $x$  and time  $t$ . Then, each unit

of time, they can each independently move one unit in either direction across the number line, or not move at all.

If  $c > 0$ , it means that  $c$  more blue pandas check position  $x$  for red pandas at time  $t$ . If a blue panda does not meet a red panda at that specific location and time, they dejectedly leave the number line right away. If there is a red panda at a position at the same time a blue panda checks it, they form a friendship and leave the number line. Each red panda can form a friendship with at most one blue panda and vice versa.

The updates will be given in order of non-decreasing  $x$  values. After each update, please print the maximum number of friendships if the red pandas move in an optimal order based on all the updates given in the input above (and including) this update.

The order in which a red panda moves can change between updates.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 # struct Point;
5 int cur = 0;
6 bool operator<(Point a, Point b) {
7     auto posa = a.x + a.dir * cur;
8     auto posb = b.x + b.dir * cur;
9     if (posa != posb) {
10         return posa < posb;
11     }
12     return a.dir > b.dir;
13 }
14 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
15 # struct Info;
16 Info operator+(Info a, Info b) {
17     Info c;
18     c.sum = a.sum + b.sum;
19     c.mx = max(a.mx, a.sum + b.mx);
20     return c;
21 }
22 # struct Node;
23 void pull(Node *t) {
24     t->sum = t->info;
25     if (t->l) {
26         t->sum = t->l->sum + t->sum;
27     }
28     if (t->r) {
29         t->sum = t->sum + t->r->sum;
30     }
31 }
32 pair<Node *, Node *> split(Node *t, Point p) {
33     if (!t) {
34         return {t, t};
35     }
36     if (p < t->p) {

```

```
37         auto [l, r] = split(t->l, p);
38         t->l = r;
39         pull(t);
40         return {l, t};
41     } else {
42         auto [l, r] = split(t->r, p);
43         t->r = l;
44         pull(t);
45         return {t, r};
46     }
47 }
48 void insert(Node *&t, Node *x) {
49     if (!t) {
50         t = x;
51         return;
52     }
53     if (x->w < t->w) {
54         auto [l, r] = split(t, x->p);
55         t = x;
56         t->l = l;
57         t->r = r;
58         pull(t);
59         return;
60     }
61     if (x->p < t->p) {
62         insert(t->l, x);
63     } else {
64         insert(t->r, x);
65     }
66     pull(t);
67 }
68 void add(Node *&t, Point p, int x) {
69     if (p < t->p) {
70         add(t->l, p, x);
71     } else if (t->p < p) {
72         add(t->r, p, x);
73     } else {
74         t->info.sum += x;
75     }
76     pull(t);
77 }
78 Node *merge(Node *a, Node *b) {
79     if (!a) {
80         return b;
81     }
82     if (!b) {
83         return a;
84     }
85     if (a->w < b->w) {
86         a->r = merge(a->r, b);
87         pull(a);
88         return a;
89     } else {
90         b->l = merge(a, b->l);
91         pull(b);
92         return b;
93     }
94 }
95 void erase(Node *&t, Point p) {
96     if (!(t->p < p) && !(p < t->p)) {
97         t = merge(t->l, t->r);
98         return;
99     }
100    if (p < t->p) {
```

```
101     erase(t->l, p);
102 } else {
103     erase(t->r, p);
104 }
105 pull(t);
106 }
107 int main() {
108     ios::sync_with_stdio(false);
109     cin.tie(nullptr);
110     int n;
111     cin >> n;
112     vector<int> x(n), t(n), c(n);
113     for (int i = 0; i < n; i++) {
114         cin >> x[i] >> t[i] >> c[i];
115     }
116     int sumb = 0;
117     Node *tree = nullptr;
118     map<Point, int> s;
119     set<pair<i64, Point>> d;
120     auto check = [&](auto it) {
121         auto r = next(it);
122         if (it->first.dir > r->first.dir) {
123             d.emplace((r->first.x - it->first.x + 1) / 2, it->first);
124         }
125     };
126     auto check2 = [&](auto it) {
127         if (it != s.begin()) {
128             check(prev(it));
129         }
130         if (next(it) != s.end()) {
131             check(it);
132         }
133     };
134     for (int i = 0; i < n; i++) {
135         while (!d.empty() && d.begin()->first <= x[i]) {
136             auto [tm, p] = *d.begin();
137             d.erase(d.begin());
138             auto it = s.find(p);
139             if (it == s.end()) {
140                 continue;
141             }
142             auto r = next(it);
143             if (r == s.end()) {
144                 continue;
145             }
146             if (it->first.x + it->first.dir * x[i] >= r->first.x + r->first.dir * x[i]) {
147                 if (it->second + r->second < 0) {
148                     it->second += r->second;
149                     add(tree, it->first, r->second);
150                     erase(tree, r->first);
151                     s.erase(r);
152                     check2(it);
153                 } else {
154                     r->second += it->second;
155                     add(tree, r->first, it->second);
156                     erase(tree, it->first);
157                     s.erase(it);
158                     check2(r);
159                 }
160             }
161         }
162         cur = x[i];
163         if (c[i] > 0) {
164             if (s.count({t[i] + cur, -1})) {
```

```

165         add(tree, {t[i] + cur, -1}, c[i]);
166     } else {
167         Node *v = new Node;
168         v->info = v->sum = c[i];
169         v->p = {t[i] + cur, -1};
170         insert(tree, v);
171     }
172     s[{t[i] + cur, -1}] += c[i];
173     auto it = s.find({t[i] + cur, -1});
174     check2(it);
175     sumb += c[i];
176 } else {
177     if (s.count({t[i] - cur, 1})) {
178         add(tree, {t[i] - cur, 1}, c[i]);
179     } else {
180         Node *v = new Node;
181         v->info = v->sum = c[i];
182         v->p = {t[i] - cur, 1};
183         insert(tree, v);
184     }
185     s[{t[i] - cur, 1}] += c[i];
186     auto it = s.find({t[i] - cur, 1});
187     check2(it);
188 }
189 cout << sumb - tree->sum.mx << "\n";
190 }
191 return 0;
192 }
```

## 941: Old Mobile

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

During the latest mission of the starship U.S.S. Coder, Captain Jan Bitovsky was accidentally teleported to the surface of an unknown planet.

Trying to find his way back, Jan found an artifact from planet Earth's ancient civilization - a mobile device capable of interstellar calls created by Byterola. Unfortunately, there was another problem. Even though Jan, as a representative of humans, knew perfectly the old notation of the cell phone numbers, the symbols on the device's keyboard were completely worn down and invisible to the human eye. The old keyboards had exactly  $m + 1$  buttons, one for each digit from the base  $m$  numerical system, and one single backspace button allowing one to erase the last written digit (if nothing was written on the screen, then this button does nothing, but it's still counted as pressed).

Jan would like to communicate with his crew. He needs to type a certain number (also from the base  $m$  numerical system, that is, digits from 0 to  $m - 1$ ). He wants to know the expected number of button presses necessary to contact the U.S.S. Coder. Jan always chooses the most optimal buttons based on his current knowledge. Buttons are indistinguishable until pressed. Help him!

## Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int n, m;
39     cin >> n >> m;
40     vector<int> a(n);
41     for (int i = 0; i < n; i++) {
42         cin >> a[i];
43     }
44     sort(a.begin(), a.end());
45     int k = unique(a.begin(), a.end()) - a.begin();
46     vector dp(k + 1, vector(m + 1, array<array<Z, 2>, 2>{})));
47     vector sd0(k + 1, vector(m + 1, array<array<Z, 2>, 2>{})));
48     vector sd1(k + 1, vector(m + 1, array<array<Z, 2>, 2>{})));
49     vector g(k + 1, vector(m + 1, array<Z, 2>{})));
50     vector s0(k + 1, vector(m + 1, array<Z, 2>{})));
51     for (int i = k - 1; i >= 0; i--) {
52         for (int j = m; j >= i; j--) {
53             for (int b = 1; b >= 0; b--) {
54                 for (int c = 1; c >= 0; c--) {
55                     if (c == 1) {
56                         if (j > i) {
57                             dp[i][j][b][1] += 1 + g[i + 1][j][b];
58                         }
59                     } else {
60                         if (j < m) {

```

```

61             if (b == 0) {
62                 dp[i][j][0][0] += comb.inv(m - j + 1) * (1 + g[i + 1][j + 1][0]);
63                 dp[i][j][0][0] += m - j - 1;
64                 dp[i][j][0][0] += (m - j) * comb.inv(m - j) * comb.inv(m - j + 1) * (1 + (i > 0));
65                 dp[i][j][0][0] += m * comb.inv(m - j) * comb.inv(m - j + 1) * sd0[i][j][1][0];
66                 dp[i][j][0][0] -= comb.inv(m - j) * comb.inv(m - j + 1) * sd1[i][j][1][0];
67             if (j + 2 <= m) {
68                 dp[i][j][0][0] += comb.inv(m - j) * comb.inv(m - j + 1) * sd1[i][j + 2][1][1];
69                 dp[i][j][0][0] -= (j + 1) * comb.inv(m - j) * comb.inv(m - j + 1) * sd0[i][j + 2][1][1];
70             }
71         } else {
72             dp[i][j][1][0] += (m - j) + s0[i + 1][j + 1][1] * comb.inv(m - j);
73         }
74     }
75 }
76 sd0[i][j][b][c] = dp[i][j][b][c];
77 sd1[i][j][b][c] = j * dp[i][j][b][c];
78 if (j < m) {
79     sd0[i][j][b][c] += sd0[i][j + 1][b][c];
80     sd1[i][j][b][c] += sd1[i][j + 1][b][c];
81 }
82 g[i][j][b] = ((j - i) * dp[i][j][b][1] + (m - j) * dp[i][j][b][0]) * comb.
83     inv(m - i);
84 s0[i][j][b] = g[i][j][b];
85 if (j < m) {
86     s0[i][j][b] += s0[i][j + 1][b];
87 }
88 }
89 }
90 auto ans = dp[0][0][0][0] + n - k;
91 cout << ans << "\n";
92 return 0;
93 }

```

## 942: Zombies

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Polycarp plays a computer game in a post-apocalyptic setting. The zombies have taken over the world, and Polycarp with a small team of survivors is defending against hordes trying to invade their base. The zombies are invading for  $x$  minutes starting from minute 0. There are  $n$  entrances to the base, and every minute one zombie attempts to enter through every entrance.

The survivors can defend the entrances against the zombies. There are two options:

manually - shoot the zombies coming through a certain entrance;

automatically - set up an electric fence on a certain entrance to fry the zombies.

If an entrance is defended either or both ways during some minute, no zombie goes through.

Every entrance is defended by a single dedicated survivor. The  $i$ -th entrance is defended manually from minute  $l_i$  until minute  $r_i$ , non-inclusive -  $[l_i, r_i)$ .

There are  $k$  generators that can be used to defend the entrances automatically. Every entrance should be connected to exactly one generator, but a generator can be connected to multiple entrances (or even none of them). Each generator will work for exactly  $m$  consecutive minutes. Polycarp can choose when to power on each generator independently of each other, the  $m$  minute long interval should be fully inside the  $[0, x)$  time interval.

Polycarp is a weird gamer. He wants the game to be as difficult as possible for him. So he wants to connect each entrance to a generator and choose the time for each generator in such a way that as many zombies as possible enter the base. Please, help him to achieve that!

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int inf = 1E9 + 1;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, k, x, m;
9     cin >> n >> k >> x >> m;
10    i64 ans = 1LL * n * (x - m);
11    vector<int> l(n), r(n);
12    vector<int> cand{0, x - m};
13    for (int i = 0; i < n; i++) {
14        cin >> l[i] >> r[i];
15        ans -= r[i] - l[i];
16        if (l[i] <= x - m) {
17            cand.push_back(l[i]);
18        }
19        if (r[i] >= m) {
20            cand.push_back(r[i] - m);
21        }
22    }
23    sort(cand.begin(), cand.end());
24    cand.erase(unique(cand.begin(), cand.end()), cand.end());
25    cand.insert(cand.begin(), -inf);
26    cand.push_back(inf);
27    int N = cand.size();
28    vector<int> o(n);
29    iota(o.begin(), o.end(), 0);
30    sort(o.begin(), o.end(), [&](int i, int j) {
31        return l[i] + r[i] < l[j] + r[j];
32    });

```

```

33     vector<vector<i64>> f(N, vector<i64>(N));
34     for (int i = 1, L = 0; i < N; i++) {
35         while (L < n && l[o[L]] + r[o[L]] - m < 2 * cand[i]) {
36             L++;
37         }
38         int R = L;
39         i64 sum = 0;
40         for (int j = i + 1; j < N; j++) {
41             while (R < n && (j == N - 1 || l[o[R]] + r[o[R]] - m < cand[i] + cand[j])) {
42                 sum += max(0, min(cand[i] + m, r[o[R]]) - max(cand[i], l[o[R]]));
43                 R++;
44             }
45             f[i][j] += sum;
46         }
47     }
48     for (int j = N - 2, R = n; j >= 0; j--) {
49         while (R && l[o[R - 1]] + r[o[R - 1]] - m >= 2 * cand[j]) {
50             R--;
51         }
52         int L = R;
53         i64 sum = 0;
54         for (int i = j - 1; i >= 0; i--) {
55             while (L && (i == 0 || l[o[L - 1]] + r[o[L - 1]] - m >= cand[i] + cand[j])) {
56                 L--;
57                 sum += max(0, min(cand[j] + m, r[o[L]]) - max(cand[j], l[o[L]]));
58             }
59             f[i][j] += sum;
60         }
61     }
62     k++;
63     i64 lo = 0, hi = 1E13;
64     auto get = [&](i64 x) {
65         vector<i64> dp(N, -1E18);
66         dp[0] = 0;
67         vector<int> cnt(N);
68         for (int i = 0; i < N; i++) {
69             for (int j = i + 1; j < N; j++) {
70                 if (dp[j] < dp[i] + f[i][j] - x) {
71                     dp[j] = dp[i] + f[i][j] - x;
72                     cnt[j] = cnt[i] + 1;
73                 }
74             }
75         }
76         return pair(dp[N - 1], cnt[N - 1]);
77     };
78     while (lo + 1 < hi) {
79         i64 m = (lo + hi) / 2;
80         auto [val, cnt] = get(m);
81         if (cnt < k) {
82             hi = m;
83         } else {
84             lo = m;
85         }
86     }
87     auto [vl, cl] = get(lo);
88     auto [vr, cr] = get(hi);
89     ans += min(vl + lo * k, vr + hi * k);
90     cout << ans << "\n";
91     return 0;
92 }

```

### 943: Deja Vu

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Everybody knows that we have been living in the Matrix for a long time. And in the new seventh Matrix the world is ruled by beavers.

So let's take beaver Neo. Neo has so-called “deja vu” outbursts when he gets visions of events in some places he's been at or is going to be at. Let's examine the phenomenon in more detail.

We can say that Neo's city is represented by a directed graph, consisting of  $n$  shops and  $m$  streets that connect the shops. No two streets connect the same pair of shops (besides, there can't be one street from A to B and one street from B to A). No street connects a shop with itself. As Neo passes some streets, he gets visions. No matter how many times he passes street  $k$ , every time he will get the same visions in the same order. A vision is a sequence of shops.

We know that Neo is going to get really shocked if he passes the way from some shop  $a$  to some shop  $b$ , possible coinciding with  $a$ , such that the list of visited shops in the real life and in the visions coincide.

Suggest beaver Neo such path of non-zero length. Or maybe you can even count the number of such paths modulo  $1000000007$  ( $10^9 + 7$ )?..

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
```

```

22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int n, m;
38     cin >> n >> m;
39     vector e(n, vector(n, false));
40     vector v(n, vector(n, vector<int>()));
41     for (int i = 0; i < m; i++) {
42         int x, y;
43         cin >> x >> y;
44         x--, y--;
45         e[x][y] = true;
46         int k;
47         cin >> k;
48         v[x][y].resize(k);
49         for (int j = 0; j < k; j++) {
50             cin >> v[x][y][j];
51             v[x][y][j]--;
52         }
53     }
54     vector<tuple<int, int, int>> f[2][2];
55     for (int x = 0; x < n; x++) {
56         for (int y = 0; y < n; y++) {
57             if (e[x][y]) {
58                 for (int i = 0; i <= v[x][y].size(); i++) {
59                     if ((!i || v[x][y][i-1] == x) && (i == v[x][y].size() || v[x][y][i] == y)) {
60                         vector<int> l, r;
61                         l.assign(v[x][y].rend() - i, v[x][y].rend());
62                         if (!l.empty()) {
63                             bool ok = true;
64                             for (int j = 0; j+1 < l.size() && l.size() <= 2*n+1; j++) {
65                                 if (!e[l[j+1]][l[j]]) {
66                                     ok = false;
67                                     break;
68                                 }
69                                 auto &f = v[l[j+1]][l[j]];
70                                 l.insert(l.end(), f.rbegin(), f.rend());
71                             }
72                             if (!ok || l.size() > 2*n+1) {
73                                 break;
74                             }
75                         r.assign(v[x][y].begin() + i, v[x][y].end());
76                         if (!r.empty()) {
77                             bool ok = true;
78                             for (int j = 0; j+1 < r.size() && r.size() <= 2*n+1; j++) {
79                                 if (!e[r[j]][r[j+1]]) {
80                                     ok = false;
81                                     break;
82                                 }
83                                 auto &f = v[r[j]][r[j+1]];
84                             }
85                         }
86                     }
87                 }
88             }
89         }
90     }
91 }
```

```

85                     r.insert(r.end(), f.begin(), f.end());
86                 }
87                 if (!ok || r.size() > 2*n+1) {
88                     break;
89                 }
90             }
91             f[!l.empty()][!r.empty()].emplace_back(l.empty() ? x : l.back(), r
92                                         .empty() ? y : r.back(), l.size() + r.size());
93         }
94     }
95 }
96 // for (int i = 0; i < 2; i++) {
97 //     for (int j = 0; j < 2; j++) {
98 //         for (auto [x, y, l] : f[i][j]) {
99 //             cerr << i << " " << j << " " << x+1 << " " << y+1 << " " << l << "\n";
100 //         }
101 //     }
102 // }
103 // }
104 vector<Z> dp(2*n+2, vector<n, array<Z, 2>{}));
105 for (int i = 0; i < n; i++) {
106     dp[0][i][0] = 1;
107 }
108 vector<Z> ans(2*n+2);
109 for (int i = 0; i <= 2*n+1; i++) {
110     for (int u = 0; u < 2; u++) {
111         for (int v = 0; v < 2; v++) {
112             for (auto [x, y, l] : f[u][v]) {
113                 if (i+l <= 2*n+1) {
114                     dp[i+l][y][v] += dp[i][x][!u];
115                 }
116             }
117         }
118     }
119     for (int x = 0; x < n; x++) {
120         ans[i] += dp[i][x][1];
121     }
122 }
123 for (int i = 2; i <= 2*n+1; i++) {
124     cout << ans[i] << "\n";
125 }
126 return 0;
127 }

```

### 944: Willy-nilly, Crack, Into Release!

- Time limit: 3 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

You have long dreamed of working in a large IT company and finally got a job there. You have studied all existing modern technologies for a long time and are ready to apply all your knowledge in practice. But then you sit down at your desk and see a sheet of paper with the company's motto printed in large letters: abcdabcdabcdabcd....

The company's motto contains four main principles- a (Willi), b (Nilli), c (Crack), d (Release). Therefore, you consider strings of length  $n$  consisting of these four Latin letters. Unordered pairs of letters "ab", "bc", "cd", and "da" in this motto are adjacent, so we will call such pairs of symbols good. So, if you are given a string  $s$  of length  $n$ , and it is known that the unordered pair of symbols  $\{x, y\}$  is good, then you can perform one of the following operations on the string:

if  $s_n = x$ , then you are allowed to replace this symbol with  $y$ ,

if there exists  $1 \leq i < n$  such that  $s_i = x$  and  $s_{i+1} = \dots = s_n = y$ , then you are allowed to replace the  $i$ -th symbol of the string with  $y$ , and all subsequent symbols with  $x$ .

For example, the string bacdd can be replaced with one of the strings bacda, bacdc, or badcc, and the string aac can be replaced with aab or aad.

A non-empty sequence of operations for the string  $s$  will be called correct if the following two conditions are met:

after performing all operations, the string becomes  $s$  again,

no string, except for  $s$ , will occur more than once during the operations. At the same time, the string  $s$  can occur exactly twice - before the start of the operations and after performing all operations.

Now we are ready to move on to the problem statement! You have a set of strings that is initially empty. Then, each of  $q$  queries adds another string  $t_i$  to the set, or removes the string  $t_i$  from the set. After each query, you need to output the minimum and maximum size of a correct sequence of operations in which each word occurs at least once. The choice of the initial string  $s$  is up to you.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 1E18;
5 # struct Info;
6 Info merge(Info a0, Info a1, Info a2, Info a3) {
7     Info res;
8     res.emp = min(inf, a0.emp + a1.emp + a2.emp + a3.emp);
9     res.cyc = min({inf, a0.cyc + a1.emp + a2.emp + a3.emp, a0.emp + a1.cyc + a2.emp + a3.
10                 emp
11                 , a0.emp + a1.emp + a2.cyc + a3.emp, a0.emp + a1.emp + a2.emp + a3.cyc
12                 , a0.d13 + a1.d02 + a2.d13 + a3.d02 + 4});
13     res.d01 = min({inf, a0.d01 + a1.d01 + a2.emp + a3.emp + 1, a0.d03 + a1.d12 + a2.d13 +
14                 a3.d02 + 3});
15     res.d02 = min({inf, a0.d01 + a1.d02 + a2.d12 + a3.emp + 2, a0.d03 + a1.emp + a2.d23 +
16                 a3.d02 + 2});
17     res.d03 = min({inf, a0.d01 + a1.d02 + a2.d13 + a3.d23 + 3, a0.d03 + a1.emp + a2.emp +
18                 a3.d03 + 1});
19     res.d12 = min({inf, a0.emp + a1.d12 + a2.d12 + a3.emp + 1, a0.d13 + a1.d01 + a2.d23 +
20                 a3.d02 + 3});
21     res.d13 = min({inf, a0.emp + a1.d12 + a2.d13 + a3.d23 + 2, a0.d13 + a1.d01 + a2.emp +
22                 a3.d03 + 2});

```

```

17     res.d23 = min({inf, a0.emp + a1.emp + a2.d23 + a3.d23 + 1, a0.d13 + a1.d02 + a2.d12 +
18     a3.d03 + 3});
19     res.Emp = max(-inf, a0.Emp + a1.Emp + a2.Emp + a3.Emp);
20     res.Cyc = max({-inf, a0.Cyc + a1.Emp + a2.Emp + a3.Emp, a0.Emp + a1.Cyc + a2.Emp + a3.
21     Emp
22     , a0.Emp + a1.Emp + a2.Cyc + a3.Emp, a0.Emp + a1.Emp + a2.Emp + a3.Cyc
23     , a0.D13 + a1.D02 + a2.D13 + a3.D02 + 4});
24     res.D01 = max({-inf, a0.D01 + a1.D01 + a2.Emp + a3.Emp + 1, a0.D03 + a1.D12 + a2.D13 +
25     a3.D02 + 3});
26     res.D02 = max({-inf, a0.D01 + a1.D02 + a2.D12 + a3.Emp + 2, a0.D03 + a1.Emp + a2.D23 +
27     a3.D02 + 2});
28     res.D03 = max({-inf, a0.D01 + a1.D02 + a2.D13 + a3.D23 + 3, a0.D03 + a1.Emp + a2.Emp +
29     a3.D03 + 1});
30     res.D12 = max({-inf, a0.Emp + a1.D12 + a2.D12 + a3.Emp + 1, a0.D13 + a1.D01 + a2.D23 +
31     a3.D02 + 3});
32     res.D13 = max({-inf, a0.Emp + a1.D12 + a2.D13 + a3.D23 + 2, a0.D13 + a1.D01 + a2.Emp +
33     a3.D03 + 2});
34     res.D23 = max({-inf, a0.Emp + a1.Emp + a2.D23 + a3.D23 + 1, a0.D13 + a1.D02 + a2.D12 +
35     a3.D03 + 3});
36     return res;
37 }
38 Info null[21];
39 constexpr int N = 20 * 100000;
40 int trie[N][4];
41 Info info[N];
42 int cnt = 1;
43 void modify(int p, const string &t, int x, i64 val) {
44     int n = t.size();
45     if (x == n) {
46         info[p].emp = val;
47         info[p].Emp = -val;
48         return;
49     }
50     int v = t[x] - 'a';
51     if (!trie[p][v]) {
52         trie[p][v] = ++cnt;
53     }
54     modify(trie[p][v], t, x + 1, val);
55     info[p] = merge(trie[p][0] ? info[trie[p][0]] : null[n - x - 1], trie[p][1] ? info[
56         trie[p][1] : null[n - x - 1]
57     , trie[p][2] ? info[trie[p][2]] : null[n - x - 1], trie[p][3] ? info[trie[p][3]] :
58         null[n - x - 1]);
59 }
60 bool adjacent(string a, string b) {
61     int i = 0;
62     while (a[i] == b[i]) {
63         i++;
64     }
65     int x = abs(a[i] - b[i]);
66     if (x != 1 && x != 3) {
67         return false;
68     }
69     for (int j = i + 1; j < a.size(); j++) {
70         if (a[i] != b[j] || b[i] != a[j]) {
71             return false;
72         }
73     }
74     return true;
75 }
76 int main() {
77     ios::sync_with_stdio(false);
78     cin.tie(nullptr);
79     for (int i = 1; i <= 20; i++) {
80         null[i] = merge(null[i - 1], null[i - 1], null[i - 1], null[i - 1]);
81     }
82 }
```

```

71     }
72     int n, q;
73     cin >> n >> q;
74     info[1] = null[n];
75     set<string> s;
76     while (q--) {
77         string t;
78         cin >> t;
79         if (s.count(t)) {
80             modify(1, t, 0, 0);
81             s.erase(t);
82         } else {
83             modify(1, t, 0, inf);
84             s.insert(t);
85         }
86         i64 ans = info[1].cyc;
87         i64 Ans = info[1].Cyc;
88         if (Ans < 0) {
89             cout << -1 << "\n";
90             continue;
91         }
92         if (s.size() <= 1) {
93             ans = 2;
94         }
95         if (s.size() == 2) {
96             auto a = *s.begin();
97             auto b = *s.rbegin();
98             if (adjacent(a, b)) {
99                 ans = 2;
100            }
101        }
102        cout << ans << " " << Ans << "\n";
103    }
104    return 0;
105 }
```

## 945: Bosco and Particle

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Bosco is studying the behaviour of particles. He decided to investigate on the peculiar behaviour of the so-called “four-one-two” particle. He does the following:

There is a line of length  $n + 1$ , where the topmost point is position 0 and bottommost is position  $n + 1$ . The particle is initially (at time  $t = 0$ ) at position 0 and heading downwards. The particle moves at the speed of 1 unit per second. There are  $n$  oscillators at positions  $1, 2, \dots, n$ .

Each oscillator can be described by a binary string. The initial state of each oscillator is the first character of its binary string. When the particle hits with an oscillator, the particle reverses its direction if its current state is 1 and continues to move at the same direction if its current state is 0, and that oscillator

moves on to the next state (the next state of the last state is defined as the first state). Additionally, the particle always reverses its direction when it is at position 0 or  $n + 1$  at time  $t > 0$ .

Bosco would like to know the cycle length of the movement of particle. The cycle length is defined as the minimum value of  $c$  such that for any time  $t \geq 0$ , the position of the particle at time  $t$  is same as the position of the particle at time  $t + c$ . It can be proved that such value  $c$  always exists. As he realises the answer might be too large, he asks you to output your answer modulo 998244353.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 998244353;
33 using Z = MInt<P>;
34 vector<int> minp, primes;
35 void sieve(int n) {
36     minp.assign(n + 1, 0);
37     primes.clear();
38     for (int i = 2; i <= n; i++) {
39         if (minp[i] == 0) {
40             minp[i] = i;
41             primes.push_back(i);
42         }
43         for (auto p : primes) {
44             if (i * p > n) {
45                 break;
46             }
47             minp[i * p] = p;
48             if (p == minp[i]) {
49                 break;
50             }
51         }
52     }
53 }
```

```
50         }
51     }
52 }
53
54 int main() {
55     ios::sync_with_stdio(false);
56     cin.tie(nullptr);
57     int n;
58     cin >> n;
59     vector<string> s(n);
60     for (int i = 0; i < n; i++) {
61         cin >> s[i];
62     }
63     for (auto &s : s) {
64         int L = s.size();
65         vector<int> f(L + 1);
66         for (int i = 1, j = 0; i < L; i++) {
67             while (j && s[i] != s[j]) {
68                 j = f[j];
69             }
70             j += (s[i] == s[j]);
71             f[i + 1] = j;
72         }
73         if (L % (L - f[L]) == 0) {
74             s.resize(L - f[L]);
75         }
76     }
77     n = find(s.begin(), s.end(), "1") - s.begin();
78     s.resize(n);
79     if (n == 0) {
80         cout << 2 << "\n";
81         return 0;
82     }
83     vector<int> a, b;
84     for (auto s : s) {
85         if (count(s.begin(), s.end(), '0') % 2 == 1) {
86             s = s + s;
87         }
88         int up = 0;
89         int down = 0;
90         int t = 0;
91         for (auto c : s) {
92             if (c == '0') {
93                 up += 1;
94                 down += 1;
95                 t ^= 1;
96             } else {
97                 if (t == 0) {
98                     up += 2;
99                 } else {
100                     down += 2;
101                 }
102             }
103         }
104         up /= 2;
105         down /= 2;
106         a.push_back(up);
107         b.push_back(down);
108         // i64 g = gcd(up, cnt);
109         // i64 t1 = up / g;
110         // i64 t2 = cnt / g;
111         // sum = sum * t1 + 2 * down * t2;
112         // cnt = down;
113     }
}
```

```

114     int N = 2;
115     for (int i = 0; i < n; i++) {
116         N = max({N, a[i], b[i]});
117     }
118     sieve(N);
119     vector<int> cur(N + 1), min(N + 1);
120     for (int i = 0; i < n; i++) {
121         int x = a[i];
122         while (x > 1) {
123             int p = minp[x];
124             cur[p]--;
125             min[p] = min(min[p], cur[p]);
126             x /= p;
127         }
128         x = b[i];
129         while (x > 1) {
130             int p = minp[x];
131             cur[p]++;
132             x /= p;
133         }
134     }
135     Z sum = 0;
136     Z v = 2;
137     for (int i = 1; i <= N; i++) {
138         if (min[i] < 0) {
139             v *= power(Z(i), -min[i]);
140         }
141     }
142     sum += v;
143     for (int i = 0; i < n; i++) {
144         v = v / a[i] * b[i];
145         sum += v;
146     }
147     cout << sum << "\n";
148     return 0;
149 }
```

## 946: The Maximum Prefix

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You're going to generate an array  $a$  with a length of at most  $n$ , where each  $a_i$  equals either 1 or  $-1$ .

You generate this array in the following way.

First, you choose some integer  $k$  ( $1 \leq k \leq n$ ), which decides the length of  $a$ .

Then, for each  $i$  ( $1 \leq i \leq k$ ), you set  $a_i = 1$  with probability  $p_i$ , otherwise set  $a_i = -1$  (with probability  $1 - p_i$ ).

After the array is generated, you calculate  $s_i = a_1 + a_2 + a_3 + \dots + a_i$ . Specially,  $s_0 = 0$ . Then you let  $S$  equal to  $\max_{i=0}^k s_i$ . That is,  $S$  is the maximum prefix sum of the array  $a$ .

You are given  $n + 1$  integers  $h_0, h_1, \dots, h_n$ . The score of an array  $a$  with maximum prefix sum  $S$  is  $h_S$ . Now, for each  $k$ , you want to know the expected score for an array of length  $k$  modulo  $10^9 + 7$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 void solve() {
35     int n;
36     cin >> n;
37     vector<Z> p(n);
38     for (int i = 0; i < n; i++) {
39         int x, y;
40         cin >> x >> y;
41         p[i] = Z(x) / y;
42     }
43     vector<Z> h(n + 1);
44     for (int i = 0; i <= n; i++) {
45         cin >> h[i];
46     }
47     for (int k = 1; k <= n; k++) {
48         vector<Z> g(n + 1);
49         g[0] += h[0] * (1 - p[k - 1]);
50         g[1] += h[0] * (1 - p[k - 1]);
51         for (int i = 1; i <= n; i++) {
52             g[i - 1] += h[i] * p[k - 1];
53             if (i < n) {
54                 g[i + 1] += h[i] * (1 - p[k - 1]);
55             }
56         }
57     }
58 }
```

```
57         h = move(g);
58         Z ans = h[0];
59         cout << ans << " \n"[k == n];
60     }
61 }
62 int main() {
63     ios::sync_with_stdio(false);
64     cin.tie(nullptr);
65     int t;
66     cin >> t;
67     while (t--) {
68         solve();
69     }
70     return 0;
71 }
```

## 947: Code Lock

- Time limit: 7 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Lara has a safe that is locked with a circle-shaped code lock that consists of a rotating arrow, a static circumference around the arrow, an input screen, and an input button.

The circumference of the lock is split into  $k$  equal sections numbered from 1 to  $k$  in clockwise order. Arrow always points to one of the sections. Each section is marked with one of the first  $k$  letters of the English alphabet. No two sections are marked with the same letter.

Due to the lock limitations, the safe's password is a string of length  $n$  that consists of first  $k$  letters of the English alphabet only. Lara enters the password by rotating the lock's arrow and pressing the input button. Initially, the lock's arrow points to section 1 and the input screen is empty. In one second she can do one of the following actions.

Rotate the arrow one section clockwise. If the arrow was pointing at section  $x < k$  it will now point at section  $x + 1$ . If the arrow was pointing at section  $k$  it will now point at section 1.

Rotate the arrow one section counter-clockwise. If the arrow was pointing at section  $x > 1$  it will now point at section  $x - 1$ . If the arrow was pointing at section 1 it will now point at section  $k$ .

Press the input button. The letter marked on the section that the arrow points to will be added to the content of the input screen.

Lara has recently found out that the safe can be re-programmed. She can take the first  $k$  letters of the English alphabet and assign them to the sectors in any order she likes. Now she wants to re-arrange the letters in a way that will minimize the number of seconds it takes her to input the password. Compute

this minimum number of seconds and the number of ways to assign letters, for which this minimum number of seconds is achieved.

Two ways to assign letters to sectors are considered to be distinct if there exists at least one sector  $i$  that is assigned different letters.

Problem: [link](#)

Solution: [link](#)

```

1 #pragma GCC optimize("Ofast")
2 #include <bits/stdc++.h>
3 using namespace std;
4 using i64 = long long;
5 constexpr int inf = 1E9;
6 # struct Info;
7 Info operator+(Info a, Info b) {
8     if (a.min < b.min) {
9         return a;
10    } else if (b.min < a.min) {
11        return b;
12    } else {
13        return {a.min, a.cnt + b.cnt};
14    }
15 }
16 int main() {
17     ios::sync_with_stdio(false);
18     cin.tie(nullptr);
19     int k, n;
20     cin >> k >> n;
21     string S;
22     cin >> S;
23     vector<int> cnt(k, vector<int>(k));
24     for (int i = 1; i < n; i++) {
25         cnt[S[i - 1] - 'a'][S[i] - 'a'] += 1;
26         cnt[S[i] - 'a'][S[i - 1] - 'a'] += 1;
27     }
28     for (int i = 0; i < k; i++) {
29         cnt[i][i] = 0;
30     }
31     int half = (k + 1) / 2;
32     vector<int> p;
33     for (int i = 0; i < half; i++) {
34         p.push_back(i);
35         if (i + half < k) {
36             p.push_back(i + half);
37         }
38     }
39     vector<int> pw(k + 1);
40     pw[0] = 1;
41     for (int i = 1; i <= k; i++) {
42         pw[i] = pw[i - 1] * 3;
43     }
44     vector<Info> dp(1 << k);
45     vector<int> tm(1 << k, -1);
46     vector<sum> sum(k, vector<int>(1 << k));
47     for (int a = 0; a < k; a++) {
48         for (int s = 1; s < (1 << k); s++) {
49             int b = __builtin_ctz(s);
50             sum[a][s] = sum[a][s ^ (1 << b)] + cnt[a][b];
51         }
52     }

```

```

53     // i64 tot = 0;
54     Info ans;
55     vector<int> q;
56     for (int s = 0; s < (1 << k); s++) {
57         if (~s >> (S[0] - 'a') & 1) {
58             continue;
59         }
60         if (__builtin_popcount(s) == half) {
61             q.clear();
62             q.push_back(0);
63             tm[0] = s;
64             dp[0] = {0, 1};
65             for (int _ = 0; _ < q.size(); _++) {
66                 int t = q[_];
67                 int c = __builtin_popcount(t);
68                 int pos = p[c];
69                 if (c % 2 == 0) {
70                     for (int C = ~t & s; C > 0; C &= (C - 1)) {
71                         int i = __builtin_ctz(C);
72                         if (c == 0 && i != S[0] - 'a') {
73                             continue;
74                         }
75                         auto val = dp[t];
76                         val.min += (sum[i][(1 << k) - 1 - (s ^ t)] - sum[i][s ^ t]) * pos;
77                         int nt = t | 1 << i;
78                         if (tm[nt] == s) {
79                             dp[nt] = dp[nt] + val;
80                         } else {
81                             dp[nt] = val;
82                             tm[nt] = s;
83                             q.push_back(nt);
84                         }
85                     }
86                 } else {
87                     for (int C = ~t & ~s & ((1 << k) - 1); C > 0; C &= (C - 1)) {
88                         int i = __builtin_ctz(C);
89                         auto val = dp[t];
90                         val.min += (sum[i][s ^ t] - sum[i][(1 << k) - 1 - (s ^ t)]) * pos;
91                         val.min += sum[i][t & s] * k;
92                         int nt = t | 1 << i;
93                         if (tm[nt] == s) {
94                             dp[nt] = dp[nt] + val;
95                         } else {
96                             dp[nt] = val;
97                             tm[nt] = s;
98                             q.push_back(nt);
99                         }
100                     }
101                 }
102             }
103             ans = ans + dp[(1 << k) - 1];
104         }
105     }
106     ans.min += n;
107     cout << ans.min << "\n";
108     cout << ans.cnt << "\n";
109     return 0;
110 }

```

## 948: Tree Cutting

- Time limit: 2 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

You are given a tree with  $n$  vertices.

A hero  $k$  times do the following operation:

Choose some edge.

Remove it.

Take one of the two remaining parts and delete it.

Write the number of vertices in the remaining part.

You are given an initial tree and the a sequence of written numbers. Find the number of ways to make operations such that the written numbers are equal to the given numbers. Due to the answer can be big, find it by modulo 998 244 353. Two ways are considered different, if on some operation edge or remaining part are selected differently.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 template<int P>
15 # struct MInt;
16 template<int V, int P>
17 constexpr MInt<P> CInv = MInt<P>(V).inv();
18 constexpr int P = 998244353;
19 using Z = MInt<P>;
20 # struct Comb comb;
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(nullptr);
24     int n;
25     cin >> n;
26     vector<vector<int>> adj(n);
27     for (int i = 1; i < n; i++) {
28         int u, v;

```

```

29         cin >> u >> v;
30         u--, v--;
31         adj[u].push_back(v);
32         adj[v].push_back(u);
33     }
34     int k;
35     cin >> k;
36     vector<int> s(k + 1);
37     s[0] = n;
38     for (int i = 1; i <= k; i++) {
39         cin >> s[i];
40     }
41     for (int i = 0; i < k; i++) {
42         s[i] -= s[i + 1];
43     }
44     k += 1;
45     vector<int> sum(1 << k);
46     for (int mask = 1; mask < (1 << k); mask++) {
47         int x = __builtin_ctz(mask);
48         sum[mask] = sum[mask ^ (1 << x)] + s[x];
49     }
50     vector<int> dp(n, vector(k, vector(1 << k, array<Z, 2>{})));
51     vector<int> siz(n);
52     auto dfs = [&](auto self, int x, int p) -> void {
53         for (int i = 0; i < k; i++) {
54             dp[x][i][0] = 1;
55         }
56         siz[x] = 1;
57         for (auto y : adj[x]) {
58             if (y == p) {
59                 continue;
60             }
61             self(self, y, x);
62             siz[x] += siz[y];
63             for (int S = (1 << k) - 1; S >= 0; S--) {
64                 for (int i = 0; i < k; i++) {
65                     if (S >> i & 1) {
66                         continue;
67                     }
68                     int other = (1 << k) - 1 - S - (1 << i);
69                     auto val = dp[x][i][S];
70                     dp[x][i][S] = {};
71                     for (int T = other; ; T = (T - 1) & other) {
72                         for (int j = 0; j < k; j++) {
73                             if (~T >> j & 1) {
74                                 continue;
75                             }
76                             if (siz[y] == sum[T]) {
77                                 if (i > j) {
78                                     dp[x][i][S | T][0] += val[0] * dp[y][j][T ^ (1 << j)][0];
79                                     dp[x][i][S | T][1] += val[1] * dp[y][j][T ^ (1 << j)][0];
80                                 }
81                                 if (i < j) {
82                                     dp[x][i][S | T][1] += val[0] * dp[y][j][T ^ (1 << j)][j < k - 1];
83                                 }
84                             }
85                         }
86                         dp[x][i][S | T][0] += val[0] * dp[y][i][T][0];
87                         dp[x][i][S | T][1] += val[0] * dp[y][i][T][1];
88                         dp[x][i][S | T][1] += val[1] * dp[y][i][T][0];
89                     if (T == 0) {

```

```

90                     break;
91                 }
92             }
93         }
94     }
95 }
96 dfs(dfs, 0, -1);
97 Z ans = 0;
98 for (int i = 0; i < k; i++) {
100    ans += dp[0][i][(1 << k) - 1 - (1 << i)][i < k - 1];
101 }
102 cout << ans << "\n";
103 return 0;
104 }
```

### 949: Crossing the Railways

- Time limit: 4 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

Isona is in a train station. This station has two platforms, and between them there are  $m$  parallel railways that can be viewed as infinite straight lines. Each railway is identified with an integer from 1 to  $m$ , railway 1 being the closest to the first platform and railway  $m$  being the farthest. There is a 1 meter distance between consecutive railways, as well as between each platform and its closest railway.

Isona is standing on the inner border of the first platform, when she realizes that she forgot to validate her ticket! There is a validating machine on the second platform, exactly opposite her current position (thus, the distance between Isona and the validating machine is  $m + 1$  meters). There are only  $s$  seconds left to validate the ticket and the bridge designated to cross the railways is too far from the validating machine. Therefore, Isona (who is very brave and a little bit careless) will cross the railways running in a straight line perpendicular to the railways themselves. Isona can only run forward (not backward) and she can stay still. When she runs at maximum speed, she needs  $v$  seconds to traverse 1 meter. She can run at any speed less than or equal to her maximum speed.

There is only one problem:  $n$  trains are programmed to transit through the railways. The  $i$ -th train will use the railway  $r_i$ . It will start crossing the straight line between Isona and the validating machine  $a_i$  seconds from now and it will end  $b_i$  seconds from now. Of course, Isona cannot cross a railway when a train is passing. Formally, for every  $i = 1, 2, \dots, n$ , Isona is not allowed to be on railway  $r_i$  at any time  $t$  with  $a_i < t < b_i$  (but she is allowed to cross at times  $a_i$  or  $b_i$ ).

The following picture summarizes the situation. In the picture there are  $m = 4$  railways and two trains are visible; the train going through railway 3 is currently crossing the line between Isona and the validating machine.

Isona is a really good runner, but she gets tired every time she has to change her running speed. What is the minimum number of speed changes she has to perform to get to the validating machine on the other platform within  $s$  seconds from now? Note that at the beginning Isona is not running. She can start to run anytime. The instant she starts to run (i.e. her speed becomes positive) is not counted as a speed change.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 class Frac {
6 public:
7     T num;
8     T den;
9     Frac(T num, T den) : num(num), den(den) {
10         if (den < 0) {
11             den = -den;
12             num = -num;
13         }
14     }
15     Frac() : Frac(0, 1) {}
16     Frac(T num) : Frac(num, 1) {}
17     double toDouble() const {
18         return 1.0 * num / den;
19     }
20     Frac &operator+=(const Frac &rhs) {
21         num = num * rhs.den + rhs.num * den;
22         den *= rhs.den;
23         return *this;
24     }
25     Frac &operator-=(const Frac &rhs) {
26         num = num * rhs.den - rhs.num * den;
27         den *= rhs.den;
28         return *this;
29     }
30     Frac &operator*=(const Frac &rhs) {
31         num *= rhs.num;
32         den *= rhs.den;
33         return *this;
34     }
35     Frac &operator/=(const Frac &rhs) {
36         num *= rhs.den;
37         den *= rhs.num;
38         if (den < 0) {
39             num = -num;
40             den = -den;
41         }
42         return *this;
43     }
44     friend Frac operator+(Frac lhs, const Frac &rhs) {
45         return lhs += rhs;
46     }
47     friend Frac operator-(Frac lhs, const Frac &rhs) {
48         return lhs -= rhs;
49     }
50     friend Frac operator*(Frac lhs, const Frac &rhs) {
51         return lhs *= rhs;

```

```

52     }
53     friend Frac operator/(Frac lhs, const Frac &rhs) {
54         return lhs /= rhs;
55     }
56     friend Frac operator-(const Frac &a) {
57         return Frac(-a.num, a.den);
58     }
59     friend bool operator==(const Frac &lhs, const Frac &rhs) {
60         return lhs.num * rhs.den == rhs.num * lhs.den;
61     }
62     friend bool operator!=(const Frac &lhs, const Frac &rhs) {
63         return lhs.num * rhs.den != rhs.num * lhs.den;
64     }
65     friend bool operator<(const Frac &lhs, const Frac &rhs) {
66         return lhs.num * rhs.den < rhs.num * lhs.den;
67     }
68     friend bool operator>(const Frac &lhs, const Frac &rhs) {
69         return lhs.num * rhs.den > rhs.num * lhs.den;
70     }
71     friend bool operator<=(const Frac &lhs, const Frac &rhs) {
72         return lhs.num * rhs.den <= rhs.num * lhs.den;
73     }
74     friend bool operator>=(const Frac &lhs, const Frac &rhs) {
75         return lhs.num * rhs.den >= rhs.num * lhs.den;
76     }
77 };
78 using F = Frac<i64>;
79 template <typename T>
80 # struct Fenwick;
81 constexpr int inf = 1E9;
82 # struct Min;
83 int main() {
84     ios::sync_with_stdio(false);
85     cin.tie(nullptr);
86     int n, m;
87     i64 s, v;
88     cin >> n >> m >> s >> v;
89     s -= (m + 1) * v;
90     if (s < 0) {
91         cout << -1 << "\n";
92         return 0;
93     }
94     vector<vector<pair<i64, i64>>> trains(m + 1);
95     for (int i = 0; i < n; i++) {
96         i64 a, b;
97         int r;
98         cin >> a >> b >> r;
99         a -= v * r;
100        b -= v * r;
101        if (b >= 0 && a <= s) {
102            trains[r].emplace_back(a, b);
103        }
104    }
105    vector<vector<pair<i64, i64>>> ranges(m + 2);
106    ranges[0].emplace_back(0, s);
107    ranges[m + 1].emplace_back(0, s);
108    vector<pair<int, i64>> pts;
109    pts.emplace_back(0, 0);
110    pts.emplace_back(0, s);
111    pts.emplace_back(m + 1, 0);
112    pts.emplace_back(m + 1, s);
113    for (int i = 1; i <= m; i++) {
114        i64 cur = 0;
115        sort(trains[i].begin(), trains[i].end());

```

```

116     for (auto [x, y] : trains[i]) {
117         if (cur <= x) {
118             ranges[i].emplace_back(cur, x);
119         }
120         cur = max(cur, y);
121     }
122     if (cur <= s) {
123         ranges[i].emplace_back(cur, s);
124     }
125     for (auto [l, r] : ranges[i]) {
126         pts.emplace_back(i, l);
127         pts.emplace_back(i, r);
128     }
129 }
130 sort(pts.begin(), pts.end());
131 pts.erase(unique(pts.begin(), pts.end()), pts.end());
132 auto check = [&](int x, F y) {
133     if (y < 0) {
134         return false;
135     }
136     if (y > s) {
137         return false;
138     }
139     i64 v = y.num / y.den + 1;
140     auto &s = ranges[x];
141     auto it = lower_bound(s.begin(), s.end(), pair(v, -1LL));
142     if (it == s.begin()) {
143         return false;
144     }
145     it--;
146     return y <= it->second;
147 };
148 // cerr << "available ranges: \n";
149 // for (int i = 0; i <= m + 1; i++) {
150 //     for (auto [l, r] : ranges[i]) {
151 //         cerr << "[" << l << ", " << r << "] ";
152 //     }
153 //     cerr << "\n";
154 // }
155 // cerr << "---\n";
156 # struct Line;
157 vector<Line> line;
158 auto add = [&](int x, i64 y, F k) {
159     int l = x, r = x;
160     while (l > 0 && check(l - 1, y + (l - 1 - x) * k)) {
161         l--;
162     }
163     while (r < m + 1 && check(r + 1, y + (r + 1 - x) * k)) {
164         r++;
165     }
166     line.push_back({x, y, k, l, r});
167     // cerr << "line " << line.size() - 1 << ": \n";
168     // cerr << x << " " << y << " " << k.toDouble() << "\n";
169     // cerr << "[" << l << ", " << r << "] \n";
170 };
171 for (int i = 0; i < pts.size(); i++) {
172     add(pts[i].first, pts[i].second, 0);
173     for (int j = i + 1; j < pts.size(); j++) {
174         if (pts[i].first < pts[j].first && pts[i].second < pts[j].second) {
175             add(pts[i].first, pts[i].second, F(pts[j].second - pts[i].second, pts[j].first - pts[i].first));
176         }
177     }
178 }

```

```

179     int N = line.size();
180     vector<int> dp(N, inf);
181     for (int i = 0; i < N; i++) {
182         if (line[i].l == 0) {
183             dp[i] = 0;
184         }
185     }
186     vector<F> fl(N), fr(N);
187     vector<int> frx(N);
188     for (int x = 1; x <= m + 1; x++) {
189         vector<int> g(N, inf);
190         vector<int> L, R;
191         for (int i = 0; i < N; i++) {
192             if (line[i].l <= x - 1 && x - 1 <= line[i].r) {
193                 L.push_back(i);
194             }
195             if (line[i].l <= x && x <= line[i].r) {
196                 g[i] = dp[i];
197                 R.push_back(i);
198             }
199             fl[i] = line[i].y + (x - 1 - line[i].x) * line[i].k;
200             fr[i] = line[i].y + (x - line[i].x) * line[i].k;
201         }
202         auto vr = fr;
203         sort(vr.begin(), vr.end());
204         for (int i = 0; i < N; i++) {
205             frx[i] = lower_bound(vr.begin(), vr.end(), fr[i]) - vr.begin();
206         }
207         auto cmp = [&](int i, int j) {
208             return fl[i] < fl[j];
209         };
210         sort(L.begin(), L.end(), cmp);
211         sort(R.begin(), R.end(), cmp);
212         Fenwick<Min> fen(N);
213         for (int i = 0, j = 0; i < R.size(); i++) {
214             while (j < L.size() && fl[L[j]] <= fl[R[i]]) {
215                 fen.add(N - 1 - frx[L[j]], {dp[L[j]]});
216                 j++;
217             }
218             g[R[i]] = min(g[R[i]], fen.sum(N - frx[R[i]]).x + 1);
219         }
220         fen.init(N);
221         reverse(L.begin(), L.end());
222         reverse(R.begin(), R.end());
223         for (int i = 0, j = 0; i < R.size(); i++) {
224             while (j < L.size() && fl[L[j]] >= fl[R[i]]) {
225                 fen.add(frx[L[j]], {dp[L[j]]});
226                 j++;
227             }
228             // if (x == 3) {
229             //     cerr << "i : " << i << ", j : " << j << "\n";
230             // }
231             g[R[i]] = min(g[R[i]], fen.sum(frx[R[i]] + 1).x + 1);
232         }
233         // for (auto x : L) {
234         //     cerr << fl[x].toDouble() << " ";
235         // }
236         // cerr << "\n";
237         // for (auto x : R) {
238         //     cerr << fl[x].toDouble() << " ";
239         // }
240         // cerr << "\n";
241         dp = g;
242         // cerr << "x : " << x << "\n";

```

```

243         // for (int i = 0; i < N; i++) {
244         //     if (dp[i] < inf) {
245         //         cerr << i << " " << dp[i] << "\n";
246         //     }
247         // }
248         // cerr << dp[15] << " " << dp[16] << "\n";
249         // cerr << fl[15].toDouble() << " " << fr[15].toDouble() << "\n";
250         // cerr << fl[16].toDouble() << " " << fr[16].toDouble() << "\n";
251         // cerr << frx[15] << " " << frx[16] << "\n";
252     }
253     int ans = inf;
254     for (int i = 0; i < N; i++) {
255         ans = min(ans, dp[i]);
256     }
257     if (ans == inf) {
258         ans = -1;
259     }
260     cout << ans << "\n";
261     return 0;
262 }
```

## 950: Segment Covering

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

ChthollyNotaSeniorious gives DataStructures a number axis with  $m$  distinct segments on it. Let  $f(l, r)$  be the number of ways to choose an even number of segments such that the union of them is exactly  $[l, r]$ , and  $g(l, r)$  be the number of ways to choose an odd number of segments such that the union of them is exactly  $[l, r]$ .

ChthollyNotaSeniorious asked DataStructures  $q$  questions. In each query, ChthollyNotaSeniorious will give DataStructures two numbers  $l, r$ , and now he wishes that you can help him find the value  $f(l, r) - g(l, r)$  modulo 998 244 353 so that he wouldn't let her down.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, q;
8     cin >> n >> q;
9     vector<pair<int, int>> a(n);
10    for (int i = 0; i < n; i++) {
11        int x, y;
12        cin >> x >> y;
```

```

13         a[i] = {x, y};
14     }
15     sort(a.begin(), a.end());
16     vector<pair<int, int>> b;
17     for (auto [x, y] : a) {
18         if (!b.empty() && b.back().first == x) {
19             continue;
20         }
21         while (!b.empty() && b.back().second >= y) {
22             b.pop_back();
23         }
24         b.emplace_back(x, y);
25     }
26     a = move(b);
27     n = a.size();
28     int lg = __lg(n);
29     vector f(lg + 1, vector<int>(n, n));
30     for (int i = 0, j = 0; i < n; i++) {
31         while (j < n && a[j].first <= a[i].second) {
32             j++;
33         }
34         f[0][i] = j;
35     }
36     for (int k = 0; k < lg; k++) {
37         for (int i = 0; i < n; i++) {
38             if (f[k][i] < n) {
39                 f[k + 1][i] = f[k][f[k][i]];
40             }
41         }
42     }
43     for (int i = 0; i < q; i++) {
44         int l, r;
45         cin >> l >> r;
46         auto x = lower_bound(a.begin(), a.end(), pair(l, -1)) - a.begin();
47         if (x == n || a[x].first != l) {
48             cout << 0 << "\n";
49             continue;
50         }
51         if (a[x].second > r) {
52             cout << 0 << "\n";
53             continue;
54         }
55         int y = x + 1;
56         if (y == n || a[y].second > r) {
57             if (a[x].second == r) {
58                 cout << 998244352 << "\n";
59             } else {
60                 cout << 0 << "\n";
61             }
62             continue;
63         }
64         for (int j = lg; j >= 0; j--) {
65             if (f[j][x] < n && a[f[j][x]].second <= r) {
66                 x = f[j][x];
67             }
68             if (f[j][y] < n && a[f[j][y]].second <= r) {
69                 y = f[j][y];
70             }
71         }
72         if (x == y) {
73             cout << 0 << "\n";
74         } else if (a[x].second == r) {
75             cout << 998244352 << "\n";
76         } else if (a[y].second == r) {

```

```

77         cout << 1 << "\n";
78     } else {
79         cout << 0 << "\n";
80     }
81 }
82 return 0;
83 }
```

## 951: Infinite Game

- Time limit: 5 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Alice and Bob are playing an infinite game consisting of sets. Each set consists of rounds. In each round, one of the players wins. The first player to win two rounds in a set wins this set. Thus, a set always ends with the score of 2 : 0 or 2 : 1 in favor of one of the players.

Let's call a game scenario a finite string  $s$  consisting of characters 'a' and 'b'. Consider an infinite string formed with repetitions of string  $s$ :  $sss\dots$ . Suppose that Alice and Bob play rounds according to this infinite string, left to right. If a character of the string  $sss\dots$  is 'a', then Alice wins the round; if it's 'b', Bob wins the round. As soon as one of the players wins two rounds, the set ends in their favor, and a new set starts from the next round.

Let's define  $a_i$  as the number of sets won by Alice among the first  $i$  sets while playing according to the given scenario. Let's also define  $r$  as the limit of ratio  $\frac{a_i}{i}$  as  $i \rightarrow \infty$ . If  $r > \frac{1}{2}$ , we'll say that scenario  $s$  is winning for Alice. If  $r = \frac{1}{2}$ , we'll say that scenario  $s$  is tied. If  $r < \frac{1}{2}$ , we'll say that scenario  $s$  is winning for Bob.

You are given a string  $s$  consisting of characters 'a', 'b', and '?'. Consider all possible ways of replacing every '?' with 'a' or 'b' to obtain a string consisting only of characters 'a' and 'b'. Count how many of them result in a scenario winning for Alice, how many result in a tied scenario, and how many result in a scenario winning for Bob. Print these three numbers modulo 998 244 353.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 998244353;
5 using i64 = long long;
6 // assume -P <= x < 2P
```

```
7  int norm(int x) {
8      if (x < 0) {
9          x += P;
10     }
11     if (x >= P) {
12         x -= P;
13     }
14     return x;
15 }
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25 }
26 # struct Z;
27 const int T[4][2] {
28     {1, 2},
29     {0, 3},
30     {3, 0},
31     {0, 0}
32 };
33 const int D[4][2] {
34     {0, 0},
35     {1, 0},
36     {0, -1},
37     {1, -1}
38 };
39 int main() {
40     ios::sync_with_stdio(false);
41     cin.tie(nullptr);
42     string s;
43     cin >> s;
44     int n = s.size();
45     int cnt = 0;
46     array<Z, 3> ans{};
47     for (int mask = 0; mask < 256; mask++) {
48         array<int, 4> to{};
49         for (int i = 0; i < 4; i++) {
50             to[i] = mask >> (2 * i) & 3;
51         }
52         array<bool, 4> on{}, cyc{};
53         int x;
54         for (x = 0; !on[x]; x = to[x]) {
55             on[x] = true;
56         }
57         int len = 0;
58         for (; !cyc[x]; x = to[x]) {
59             cyc[x] = true;
60             len++;
61         }
62         bool ok = true;
63         for (int i = 0; i < 4; i++) {
64             if (!on[i] && to[i] != 0) {
65                 ok = false;
66             }
67         }
68         if (!ok) {
69             continue;
70     }
```

```

71         const int L = (n + 1) / 2 * len;
72         int start = 0;
73         for (int i = 0; i < 4; i++) {
74             if (on[i]) {
75                 start |= i << (2 * i);
76             }
77         }
78         vector dp(1 << 8, vector<Z>(2 * L + 1));
79         dp[start][L] = 1;
80         vector<array<int, 2>> trans(256), delta(256);
81         for (int m = 0; m < 256; m++) {
82             for (int x = 0; x < 2; x++) {
83                 int nxt = 0;
84                 for (int i = 0; i < 4; i++) {
85                     int to = m >> (2 * i) & 3;
86                     if (cyc[i]) {
87                         delta[m][x] += D[to][x];
88                     }
89                     to = T[to][x];
90                     if (!on[i]) {
91                         to = 0;
92                     }
93                     nxt |= to << (2 * i);
94                 }
95                 trans[m][x] = nxt;
96             }
97         }
98         for (int i = 0; i < n; i++) {
99             vector g(1 << 8, vector<Z>(2 * L + 1));
100            for (int m = 0; m < 256; m++) {
101                for (int j = 0; j <= 2 * L; j++) {
102                    if (dp[m][j].val() == 0) continue;
103                    for (int x = 0; x < 2; x++) {
104                        if (s[i] == 'a' + x || s[i] == '?') {
105                            g[trans[m][x]][j + delta[m][x]] += dp[m][j];
106                        }
107                    }
108                }
109            }
110            swap(dp, g);
111        }
112        for (int i = 0; i <= 2 * L; i++) {
113            if (i > L) {
114                ans[0] += dp[mask][i];
115            } else if (i == L) {
116                ans[1] += dp[mask][i];
117            } else {
118                ans[2] += dp[mask][i];
119            }
120        }
121        // if (mask == 0) {
122        //     cerr << "trans : " << trans[start][0] << "\n";
123        //     cerr << "trans : " << trans[1][0] << "\n";
124        //     cerr << "trans : " << trans[0][0] << "\n";
125        //     cerr << "trans : " << trans[1][1] << "\n";
126        //     cerr << "trans : " << trans[3][1] << "\n";
127        //     cerr << "dp : " << dp[mask][L] << "\n";
128        // }
129    }
130    // cout << cnt << "\n";
131    for (int i = 0; i < 3; i++) {
132        cout << ans[i] << "\n";
133    }
134    return 0;

```

```
135 }
```

## greedy

### 952: Half-sum

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You're given a multiset of non-negative integers  $\{a_1, a_2, \dots, a_n\}$ .

In one step you take two elements  $x$  and  $y$  of the multiset, remove them and insert their mean value  $\frac{x+y}{2}$  back into the multiset.

You repeat the step described above until you are left with only two numbers  $A$  and  $B$ . What is the maximum possible value of their absolute difference  $|A - B|$ ?

Since the answer is not an integer number, output it modulo  $10^9 + 7$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
```

```
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 void solve() {
35     int n;
36     cin >> n;
37     vector<int> a(n);
38     for (int i = 0; i < n; i++) {
39         cin >> a[i];
40     }
41     sort(a.begin(), a.end());
42     vector<int> d(n - 1);
43     for (int i = 0; i < n - 1; i++) {
44         d[i] = a[i + 1] - a[i];
45     }
46     vector<i64> ans(n);
47     ans[0] = 1E18;
48     auto check = [&](int m) {
49         vector<i64> a(n);
50         for (int i = 0; i < m; i++) {
51             a[i] += d[i];
52         }
53         for (int i = n - 2; i > m; i--) {
54             a[n - 2 - i] += d[i];
55         }
56         for (int i = n - 1; i; i--) {
57             a[i - 1] += a[i] / 2;
58             a[i] %= 2;
59         }
60         if (a < ans) {
61             ans = a;
62         }
63     };
64     check(0);
65     for (int i = 0; i < n - 1; i++) {
66         if (d[i] != 0) {
67             for (int j = i; j < n - 1 && j < i + 60; j++) {
68                 check(j);
69             }
70             break;
71         }
72     }
73     for (int i = n - 2; i >= 0; i--) {
74         if (d[i] != 0) {
75             for (int j = i; j >= 0 && j > i - 60; j--) {
76                 check(j);
77             }
78             break;
79         }
80     }
81     Z res = 0;
82     Z p = 1;
83     for (int i = 0; i < n; i++) {
84         p *= (P + 1) / 2;
85         res += ans[i] * p;
86     }
87     res = a[n - 1] - a[0] - res;
88     cout << res << "\n";
89 }
90 int main() {
91     ios::sync_with_stdio(false);
92     cin.tie(nullptr);
93     int t;
```

```
94     cin >> t;
95     while (t--) {
96         solve();
97     }
98     return 0;
99 }
```

### 953: N Machines

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

You have been invited as a production process optimization specialist to some very large company. The company has  $n$  machines at its factory, standing one behind another in the production chain. Each machine can be described in one of the following two ways:  $(+, a_i)$  or  $(*, a_i)$ .

If a workpiece with the value  $x$  is supplied to the machine of kind  $(+, a_i)$ , then the output workpiece has value  $x + a_i$ .

If a workpiece with the value  $x$  is supplied to the machine of kind  $(*, a_i)$ , then the output workpiece has value  $x \cdot a_i$ .

The whole production process is as follows. The workpiece with the value 1 is supplied to the first machine, then the workpiece obtained after the operation of the first machine is supplied to the second machine, then the workpiece obtained after the operation of the second machine is supplied to the third machine, and so on. The company is not doing very well, so now the value of the resulting product does not exceed  $2 \cdot 10^9$ .

The directors of the company are not satisfied with the efficiency of the production process and have given you a budget of  $b$  coins to optimize it.

To optimize production you can change the order of machines in the chain. Namely, by spending  $p$  coins, you can take any machine of kind  $(+, a_i)$  and move it to any place in the chain without changing the order of other machines. Also, by spending  $m$  coins, you can take any machine of kind  $(*, a_i)$  and move it to any place in the chain.

What is the maximum value of the resulting product that can be achieved if the total cost of movements that are made should not exceed  $b$  coins?

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr i64 inf = 4E18;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, b, p, m;
9     cin >> n >> b >> p >> m;
10    vector<int> x;
11    int total = 0;
12    vector<vector<int>> a;
13    vector<int> mul;
14    for (int i = 0; i < n; i++) {
15        char t;
16        int y;
17        cin >> t >> y;
18        if (t == '+') {
19            x.push_back(y);
20            total++;
21        } else if (y > 1) {
22            a.push_back(move(x));
23            mul.push_back(y);
24        }
25    }
26    a.push_back(move(x));
27    int cntm = mul.size();
28    for (int i = 0; i <= cntm; i++) {
29        sort(a[i].begin(), a[i].end());
30    }
31    vector<vector<i64>> suf(cntm + 1);
32    for (int i = 0; i <= cntm; i++) {
33        suf[i].resize(a[i].size() + 1);
34        for (int j = int(a[i].size()) - 1; j >= 0; j--) {
35            suf[i][j] = suf[i][j + 1] + a[i][j];
36        }
37    }
38    i64 ans = 0;
39    map<int, vector<int>> mpos;
40    for (int i = 0; i < cntm; i++) {
41        mpos[mul[i]].push_back(i);
42    }
43    vector<bool> move(cntm);
44    auto dfs = [&](auto self, auto it) {
45        if (it == mpos.end()) {
46            vector<i64> val(cntm + 1, 1);
47            i64 cost = 0;
48            for (int i = 0; i < cntm; i++) {
49                if (move[i]) {
50                    val[cntm] *= mul[i];
51                    cost += m;
52                } else {
53                    val[i] *= mul[i];
54                }
55            }
56            for (int i = cntm - 1; i >= 0; i--) {
57                val[i] *= val[i + 1];
58            }
59            if (cost > b) {
60                return;
61            }
62            const int rest = min((b - cost) / p, 1LL * total);
63            i64 cur = val[0];
64            for (int i = 0; i <= cntm; i++) {
```

```

65         cur += val[i] * suf[i][0];
66     }
67     auto get = [&](auto x) {
68     int cnt = 0;
69     i64 sum = 0;
70     for (int i = 0; i <= cntm; i++) {
71         if (val[i] == val[0]) {
72             continue;
73         }
74         i64 y = x / (val[0] - val[i]);
75         auto it = upper_bound(a[i].begin(), a[i].end(), y);
76         int k = it - a[i].begin();
77         cnt += a[i].size() - k;
78         sum += suf[i][k] * (val[0] - val[i]);
79     }
80     return pair(cnt, sum);
81 };
82 i64 lo = 0, hi = inf;
83 while (lo < hi) {
84     i64 x = (lo + hi) / 2;
85     if (get(x).first > rest) {
86         lo = x + 1;
87     } else {
88         hi = x;
89     }
90 }
91 auto [c, s] = get(lo);
92 cur += s + 1LL * (rest - c) * lo;
93 ans = max(ans, cur);
94 return;
95 }
96 self(self, next(it));
97 for (int i = 0; i < int(it->second.size()); i++) {
98     move[it->second[i]] = true;
99     self(self, next(it));
100 }
101 for (auto i : it->second) {
102     move[i] = false;
103 }
104 };
105 dfs(dfs, mpos.begin());
106 cout << ans << "\n";
107 return 0;
108 }

```

**math****954: Fugitive Frenzy**

- Time limit: 5 seconds
- Memory limit: 1024 megabytes
- Input file: standard input
- Output file: standard output

The city of F. can be represented as a tree. A famous fugitive is hiding in it, and today a faithful police officer decided to catch him at all costs. The police officer is stronger than the fugitive, but the fugitive

is much faster than the former. That is why the pursuit proceeds as follows. At the moment  $t = 0$  the police officer appears at the vertex with number  $s$ , and the fugitive spawns at any other vertex of his choice. After that, they take turns, starting with the police officer.

During the police officer's move, she selects any vertex adjacent to the one where she is currently located and moves there. The police officer spends one minute moving. Also, the police officer may decide to stand still instead, in which case she waits one minute at the vertex at which she started her move. If at the end of the turn the police officer ends up at the same vertex as the fugitive, she instantly catches him and the chase ends.

The fugitive's move is as follows. Let him be at vertex  $b$ , and the police officer at vertex  $p$ . Then the fugitive chooses any vertex  $b' \neq p$  such that the path between the vertices  $b$  and  $b'$  does not contain vertex  $p$  and instantly moves there. In particular, he can always choose  $b' = b$  to stay where he is. The fugitive's move takes no time.

Note that the fugitive managed to attach a radio bug to the police officer's badge a week ago, so the fugitive knows the location of the police officer at every moment (in particular, he knows the number  $s$ ). On the contrary, the police officer knows nothing about the fugitive's movements and will only be able to detect him at the very moment she catches him.

The police officer aims to catch the fugitive as fast as possible, and the fugitive aims to be caught as late as possible. Since the chase can be thought of as a game with incomplete information, participants can use mixed (probabilistic) strategies - thus, the police officer acts to minimize the expected duration of the chase, and the fugitive - to maximize it.

Find the mathematical expectation of the duration of the chase with optimal actions of the police officer and the fugitive. It can be proven that it is always finite. In particular, with optimal strategies, the probability that the chase continues indefinitely is equal to zero.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n;
8     cin >> n;
9     vector<vector<int>> adj(n);
10    for (int i = 1; i < n; i++) {
11        int u, v;
12        cin >> u >> v;
13        u--, v--;
14        adj[u].push_back(v);
15        adj[v].push_back(u);

```

```

16     }
17     vector<int> dis(n, vector<int>(n, -1));
18     for (int s = 0; s < n; s++) {
19         queue<int> q;
20         q.push(s);
21         dis[s][s] = 0;
22         while (!q.empty()) {
23             int x = q.front();
24             q.pop();
25             for (auto y : adj[x]) {
26                 if (dis[s][y] == -1) {
27                     dis[s][y] = dis[s][x] + 1;
28                     q.push(y);
29                 }
30             }
31         }
32     }
33     vector<int> leaves;
34     for (int i = 0; i < n; i++) {
35         if (adj[i].size() == 1) {
36             leaves.push_back(i);
37         }
38     }
39     int s;
40     cin >> s;
41     s--;
42     vector<double> f(n, 1);
43     for (int t = 0; t < 10000; t++) {
44         for (auto x : leaves) {
45             double num = 0;
46             double den = 0;
47             for (auto y : leaves) {
48                 if (y != x) {
49                     num += 1 + dis[x][y] / f[y];
50                     den += 1.0 / f[y];
51                 }
52             }
53             f[x] = (num - 1) / den;
54         }
55     }
56     double num = 0;
57     double den = 0;
58     for (auto y : leaves) {
59         if (y != s) {
60             num += 1 + dis[s][y] / f[y];
61             den += 1.0 / f[y];
62         }
63     }
64     double ans = (num - 1) / den;
65     cout << fixed << setprecision(10) << ans << "\n";
66     return 0;
67 }

```

## 955: Parallel Universes (Hard)

- Time limit: 4 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Heidi enjoyed performing the simulation because she knew exactly when a new universe would be formed and where, and when a non-existent link would be broken and where.

However, the multiverse itself works in mysterious ways. Well, it works using probabilities, which to some people is mysterious.

At each unit time, when a decision is made, one of the two events will happen randomly. Let's denote  $l$  as the current length of the multiverse. With a probability of  $p_{create} = 1 - \frac{l}{m}$ , a universe will be created. With a probability of  $p_{break} = \frac{l}{m}$ , a non-existent link will be broken at some position.

More specifically,

When a universe is created, it will manifest itself between any two adjacent universes or at one of the ends. Each position occurs with a probability of  $\frac{1}{l+1}$ .

When a link is broken, it could be cut between any two adjacent universes, each with a probability of  $\frac{1}{l-1}$ . After separating the multiverse into two segments, the segment NOT containing the Doctor will cease to exist.

As earlier, the Doctor remains in the same universe. However, if at some point the multiverse breaks in such a way that the Doctor finds himself at the leftmost or rightmost end of it, the TARDIS stops functioning.

In such a case, the Doctor must actually walk across the multiverse to find the tools to fix it.

We are interested in the expected value of the length of the multiverse when such an event occurs.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
```

```

25 i64 MLong<OLL>::Mod = i64(1E18) + 9;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 998244353;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 # struct Comb comb;
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38     int n, k, m;
39     cin >> n >> k >> m;
40     if (m == 1) {
41         cout << 1 << "\n";
42         return 0;
43     }
44     vector f(m, vector(m, vector<Z>(m - 1)));
45     for (int i = 1; i + 1 < m; i++) {
46         f[1][i][i - 1] = 1;
47     }
48     for (int i = 0; i < m; i++) {
49         f[0][i][m - 2] = f[i][0][m - 2] = i + 1;
50     }
51     vector<vector<Z>> a;
52     vector<Z> inv(m + 1);
53     for (int i = 1; i <= m; i++) {
54         inv[i] = comb.inv(i);
55     }
56     auto si = f;
57     auto sj = f;
58     for (int i = 1; i < m; i++) {
59         for (int j = 1; i + j < m; j++) {
60             if (i + j < m - 1) {
61                 int l = i + j + 1;
62                 auto v = inv[i + 1] * (l + 1) * inv[m - l] * m;
63                 for (int x = 0; x < m - 1; x++) {
64                     f[i + 1][j][x] += f[i][j][x];
65                     f[i + 1][j][x] -= (1 - l * inv[m]) * (j + 1) * inv[l + 1] * f[i][j + 1][x];
66                     f[i + 1][j][x] -= l * inv[m] * inv[l - 1] * si[i - 1][j][x];
67                     f[i + 1][j][x] -= l * inv[m] * inv[l - 1] * sj[i][j - 1][x];
68                     f[i + 1][j][x] *= v;
69                 }
70             } else {
71                 vector<Z> r(m - 1);
72                 for (int x = 0; x < m - 1; x++) {
73                     r[x] += f[i][j][x];
74                     r[x] -= inv[m - 1] * si[i - 1][j][x];
75                     r[x] -= inv[m - 1] * sj[i][j - 1][x];
76                 }
77                 a.push_back(r);
78             }
79             for (int x = 0; x < m - 1; x++) {
80                 si[i][j][x] = si[i - 1][j][x] + f[i][j][x];
81                 sj[i][j][x] = sj[i][j - 1][x] + f[i][j][x];
82             }
83         }
84     }
85     assert(a.size() == m - 2);
86     for (int i = 0; i < m - 2; i++) {
87         for (int j = i; j < m - 2; j++) {

```

```

88         if (a[j][i] != 0) {
89             swap(a[i], a[j]);
90             break;
91         }
92     }
93     auto inv = a[i][i].inv();
94     for (int j = 0; j < m - 1; j++) {
95         a[i][j] *= inv;
96     }
97     for (int j = 0; j < m - 2; j++) {
98         if (j == i) {
99             continue;
100        }
101        auto x = a[j][i];
102        for (int k = i; k < m - 1; k++) {
103            a[j][k] -= a[i][k] * x;
104        }
105    }
106    int i = k - 1, j = n - k;
107    Z ans = f[i][j][m - 2];
108    for (int x = 0; x < m - 2; x++) {
109        ans -= f[i][j][x] * a[x][m - 2];
110    }
111    cout << ans << "\n";
112    return 0;
113}
114

```

### 956: Survival of the Weakest (hard version)

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. It differs from the easy one only in constraints on  $n$ . You can make hacks only if you lock both versions.

Let  $a_1, a_2, \dots, a_n$  be an array of non-negative integers. Let  $F(a_1, a_2, \dots, a_n)$  be the sorted in the non-decreasing order array of  $n - 1$  smallest numbers of the form  $a_i + a_j$ , where  $1 \leq i < j \leq n$ . In other words,  $F(a_1, a_2, \dots, a_n)$  is the sorted in the non-decreasing order array of  $n - 1$  smallest sums of all possible pairs of elements of the array  $a_1, a_2, \dots, a_n$ . For example,  $F(1, 2, 5, 7) = [1+2, 1+5, 2+5] = [3, 6, 7]$ .

You are given an array of non-negative integers  $a_1, a_2, \dots, a_n$ . Determine the single element of the array  $\underbrace{F(F(\dots F}_{n-1}(a_1, a_2, \dots, a_n) \dots))$ . Since the answer can be quite large, output it modulo  $10^9 + 7$ .

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 template<class T>
5 constexpr T power(T a, i64 b) {
6     T res = 1;
7     for (; b; b /= 2, a *= a) {
8         if (b % 2) {
9             res *= a;
10        }
11    }
12    return res;
13 }
14 constexpr i64 mul(i64 a, i64 b, i64 p) {
15     i64 res = a * b - i64(1.L * a * b / p) * p;
16     res %= p;
17     if (res < 0) {
18         res += p;
19     }
20     return res;
21 }
22 template<i64 P>
23 # struct MLong;
24 template<>
25 i64 MLong<0LL>::Mod = 1;
26 template<int P>
27 # struct MInt;
28 template<>
29 int MInt<0>::Mod = 1;
30 template<int V, int P>
31 constexpr MInt<P> CInv = MInt<P>(V).inv();
32 constexpr int P = 1000000007;
33 using Z = MInt<P>;
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(nullptr);
37     int n;
38     cin >> n;
39     vector<int> a(n);
40     for (int i = 0; i < n; i++) {
41         cin >> a[i];
42     }
43     sort(a.begin(), a.end());
44     int t = n - 1;
45     Z ans = 0;
46     for (int i = 0; i < t; i++) {
47         n = a.size();
48         ans *= 2;
49         priority_queue<pair<int, int>> h;
50         vector<int> f(n);
51         for (int j = 0; j < n; j++) {
52             f[j] = j + 1;
53             if (f[j] < n) {
54                 h.emplace(-a[j] - a[f[j]], j);
55             }
56         }
57         vector<int> b;
58         while (b.size() < 64 && !h.empty()) {
59             auto [s, j] = h.top();
60             h.pop();
61             b.push_back(-s);
62             f[j] += 1;
63             if (f[j] < n) {
64                 h.emplace(-a[j] - a[f[j]], j);
```

```

65         }
66     }
67     ans += b[0];
68     int v = b[0];
69     for (auto &x : b) {
70         x -= v;
71     }
72     a = b;
73 }
74 cout << ans << "\n";
75 return 0;
76 }
```

### 957: Window Signals (easy version)

- Time limit: 7 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

This is the easy version of the problem. In this version, the constraints on  $h$  and  $w$  are lower.

A house at the sea has  $h$  floors, all of the same height. The side of the house facing the sea has  $w$  windows at equal distances from each other on every floor. Thus, the windows are positioned in cells of a rectangular grid of size  $h \times w$ .

In every window, the light can be turned either on or off, except for the given  $k$  (at most 2) windows. In these  $k$  windows the light can not be turned on, because it is broken.

In the dark, we can send a signal to a ship at sea using a configuration of lights turned on and off. However, the ship can not see the position of the lights with respect to the house. Thus, if one configuration of windows with lights on can be transformed into another using parallel translation, these configurations are considered equal. Note that only parallel translation is allowed, but neither rotations nor flips are. Moreover, a configuration without any light at all is not considered valid.

Find how many different signals the ship can receive and print this number modulo 998 244 353.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 998244353;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
```

```

8     if (x < 0) {
9         x += P;
10    }
11    if (x >= P) {
12        x -= P;
13    }
14    return x;
15 }
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25 }
26 # struct Z;
27 void solve() {
28     int h, w, k;
29     cin >> h >> w >> k;
30     if (k == 0) {
31         Z ans = power(Z(2), h * w) - power(Z(2), (h - 1) * w) - power(Z(2), h * (w - 1)) +
32             power(Z(2), (h - 1) * (w - 1));
33         cout << ans << "\n";
34         return;
35     }
36     if (k == 1) {
37         int r, c;
38         cin >> r >> c;
39         r--, c--;
40         Z ans = 0;
41         for (int a = 1; a <= h; a++) {
42             for (int b = 1; b <= w; b++) {
43                 for (auto U : {0, 1}) for (auto D : {a, a - 1}) for (auto L : {0, 1}) for
44                     (auto R : {b, b - 1}) {
45                     Z coef = (a + b - (D - U) - (R - L)) % 2 ? -1 : 1;
46                     ans += coef * power(Z(2), max(0, D - U) * max(0, R - L));
47                     if (D <= U || R <= L) continue;
48                     int x0 = r - (h - a);
49                     int y0 = c - (w - b);
50                     int x1 = r;
51                     int y1 = c;
52                     auto valid = [&](int x, int y) {
53                         return U <= x && x < D && L <= y && y < R;
54                     };
55                     if (!valid(x0, y0)) continue;
56                     if (!valid(x1, y1)) continue;
57                     if (!valid(x0, y1)) continue;
58                     if (!valid(x1, y0)) continue;
59                     int total = (D - U) * (R - L);
60                     Z res = power(Z(2), total - (h - a + 1) * (w - b + 1));
61                     ans -= coef * res;
62                 }
63             }
64         }
65     }
66     int r[2], c[2];
67     cin >> r[0] >> c[0];
68     cin >> r[1] >> c[1];
69     r[0]--, c[0]--;

```

```

70     r[1]--, c[1]--;
71     if (r[0] > r[1]) {
72         swap(r[0], r[1]);
73         swap(c[0], c[1]);
74     }
75     if (c[0] > c[1]) {
76         c[0] = w - 1 - c[0];
77         c[1] = w - 1 - c[1];
78     }
79     int dx = r[1] - r[0];
80     int dy = c[1] - c[0];
81     int N = max(h, w) + 2;
82     vector<Z> f(N + 1);
83     f[0] = 1, f[1] = 2;
84     for (int i = 2; i <= N; i++) {
85         f[i] = f[i - 1] + f[i - 2];
86     }
87     Z ans = 0;
88     for (int a = 1; a <= h; a++) {
89         for (int b = 1; b <= w; b++) {
90             for (auto U : {0, 1}) for (auto D : {a, a - 1}) for (auto L : {0, 1}) for (
91                 auto R : {b, b - 1}) {
92                 Z coef = (a + b - (D - U) - (R - L)) % 2 ? -1 : 1;
93                 ans += coef * power(Z(2), max(0, D - U) * max(0, R - L));
94                 if (D <= U || R <= L) continue;
95                 int x0 = r[0] - (h - a);
96                 int y0 = c[0] - (w - b);
97                 int x1 = r[0];
98                 int y1 = c[0];
99                 auto valid = [&](int x, int y) {
100                     return U <= x && x < D && L <= y && y < R;
101                 };
102                 if (!valid(x0, y0) && !valid(x0 + dx, y0 + dy)) continue;
103                 if (!valid(x1, y1) && !valid(x1 + dx, y1 + dy)) continue;
104                 if (!valid(x0, y1) && !valid(x0 + dx, y1 + dy)) continue;
105                 if (!valid(x1, y0) && !valid(x1 + dx, y0 + dy)) continue;
106                 if (U > D - dx && U > x0 && x1 + 1 > D - dx) continue;
107                 if (L > R - dy && L > y0 && y1 + 1 > R - dy) continue;
108                 Z res = 1;
109                 int total = (D - U) * (R - L);
110                 vector<int> freq(N + 1);
111                 for (int i = 2; i <= N; i++) {
112                     array<int, 3> cnt{};
113                     for (auto u : {0, 1}) for (auto v : {0, 1}) {
114                         for (int f = 1; f < 4; f++) for (int g = 1; g < 4; g++) {
115                             int xl = x0, xr = x1 + 1 - (i - 2) * dx;
116                             int yl = y0, yr = y1 + 1 - (i - 2) * dy;
117                             if (u) {
118                                 xl = max(xl, U);
119                                 yl = max(yl, L);
120                             }
121                             if (v) {
122                                 xr = min(xr, D - (i - 1) * dx);
123                                 yr = min(yr, R - (i - 1) * dy);
124                             }
125                             if (f & 1) xr = min(xr, x0 + dx);
126                             if (f & 2) yr = min(yr, y0 + dy);
127                             if (g & 1) xl = max(xl, x1 + 1 - (i - 1) * dx);
128                             if (g & 2) yl = max(yl, y1 + 1 - (i - 1) * dy);
129                             cnt[2 - u - v] += (__builtin_parity(f ^ g) ? -1 : 1) * max(0,
130                                         xr - xl) * max(0, yr - yl);
131                         }
132                     }
133                     cnt[1] -= 2 * cnt[0];

```

```

132             cnt[2] -= cnt[0] + cnt[1];
133             freq[i] += cnt[0];
134             freq[i - 2] += cnt[1];
135             total -= cnt[1];
136             if (i == 3) {
137                 total -= cnt[2];
138             } else if (i >= 4) {
139                 freq[i - 4] += cnt[2];
140                 total -= 2 * cnt[2];
141             }
142         }
143         for (int i = 2; i <= N; i++) {
144             res *= power(f[i], freq[i]);
145             total -= i * freq[i];
146         }
147         res *= power(f[1], total);
148         ans -= coef * res;
149     }
150 }
151 cout << ans << "\n";
152 }
153 int main() {
154     ios::sync_with_stdio(false);
155     cin.tie(nullptr);
156     int t;
157     cin >> t;
158     while (t--) {
159         solve();
160     }
161     return 0;
162 }
```

## 958: Koxia and Sequence

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Mari has three integers  $n$ ,  $x$ , and  $y$ .

Call an array  $a$  of  $n$  non-negative integers good if it satisfies the following conditions:

$$a_1 + a_2 + \dots + a_n = x, \text{ and}$$

$$a_1 | a_2 | \dots | a_n = y, \text{ where } | \text{ denotes the bitwise OR operation.}$$

The score of a good array is the value of  $a_1 \oplus a_2 \oplus \dots \oplus a_n$ , where  $\oplus$  denotes the bitwise XOR operation.

Koxia wants you to find the total bitwise XOR of the scores of all good arrays. If there are no good arrays, output 0 instead.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 998244353;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
8     if (x < 0) {
9         x += P;
10    }
11    if (x >= P) {
12        x -= P;
13    }
14    return x;
15 }
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25 }
26 # struct Z;
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(nullptr);
30     i64 n, x;
31     int y;
32     cin >> n >> x >> y;
33     if (n % 2 == 0) {
34         cout << 0 << "\n";
35         return 0;
36     }
37     i64 ans = 0;
38     for (int s = y; ; s = (s - 1) & y) {
39         for (int i = 0; i < 20; i++) {
40             if (~s >> i & 1) continue;
41             i64 v = x - (1 << i);
42             if (v < 0) continue;
43             i64 t = n * s - (1 << i);
44             if ((t & v) == v) ans ^= 1 << i;
45         }
46         if (!s) break;
47     }
48     cout << ans << "\n";
49     return 0;
50 }

```

## 959: Koxia and Bracket

- Time limit: 5 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Chiyuu has a bracket sequence<sup>†</sup>  $s$  of length  $n$ . Let  $k$  be the minimum number of characters that Chiyuu has to remove from  $s$  to make  $s$  balanced<sup>‡</sup>.

Now, Koxia wants you to count the number of ways to remove  $k$  characters from  $s$  so that  $s$  becomes balanced, modulo 998 244 353.

Note that two ways of removing characters are considered distinct if and only if the set of indices removed is different.

<sup>†</sup> A bracket sequence is a string containing only the characters “(” and “)”.

<sup>‡</sup> A bracket sequence is called balanced if one can turn it into a valid math expression by adding characters + and 1. For example, sequences (())(), (), ((())()) and the empty string are balanced, while )(, ()(), and ((())() are not.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 998244353;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
8     if (x < 0) {
9         x += P;
10    }
11    if (x >= P) {
12        x -= P;
13    }
14    return x;
15 }
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25 }
26 # struct Z;
27 vector<int> rev;
28 vector<Z> roots{0, 1};
29 void dft(vector<Z> &a) {
30     int n = a.size();
31     if (int(rev.size()) != n) {
32         int k = __builtin_ctz(n) - 1;
33         rev.resize(n);
34         for (int i = 0; i < n; i++) {
35             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
36         }
37     }
38     for (int i = 0; i < n; i++) {
39         if (rev[i] < i) {
40             swap(a[i], a[rev[i]]);
```

```

41         }
42     }
43     if (int(roots.size()) < n) {
44         int k = __builtin_ctz(roots.size());
45         roots.resize(n);
46         while ((1 << k) < n) {
47             Z e = power(Z(3), (P - 1) >> (k + 1));
48             for (int i = 1 << (k - 1); i < (1 << k); i++) {
49                 roots[2 * i] = roots[i];
50                 roots[2 * i + 1] = roots[i] * e;
51             }
52             k++;
53         }
54     }
55     for (int k = 1; k < n; k *= 2) {
56         for (int i = 0; i < n; i += 2 * k) {
57             for (int j = 0; j < k; j++) {
58                 Z u = a[i + j];
59                 Z v = a[i + j + k] * roots[k + j];
60                 a[i + j] = u + v;
61                 a[i + j + k] = u - v;
62             }
63         }
64     }
65 }
66 void idft(vector<Z> &a) {
67     int n = a.size();
68     reverse(a.begin() + 1, a.end());
69     dft(a);
70     Z inv = (1 - P) / n;
71     for (int i = 0; i < n; i++) {
72         a[i] *= inv;
73     }
74 }
75 # struct Poly;
76 vector<Z> fac, invfac;
77 Poly polyPow(int n) {
78     return Poly(n + 1, [&](int i) {
79         return fac[n] * invfac[i] * invfac[n - i];
80     });
81 }
82 Z solve(string s) {
83     int n = s.size();
84     vector<int> a{0};
85     for (auto c : s) {
86         if (c == '(') a.back()++;
87         else a.push_back(0);
88     }
89     vector<Z> b(a.size());
90     auto work = [&](auto work, int l, int r, Poly p) {
91         p = p.modxk(r - l);
92         if (r - l == 1) {
93             int x = a[l];
94             b[l] = p.size() == 0 ? 0 : p[0];
95             return make_pair(Poly{b[l]}, x);
96         }
97         int m = (l + r) / 2;
98         auto L = work(work, l, m, p);
99         auto R = work(work, m, r, (p * polyPow(L.second) - L.first).divxk(m - l));
100        return make_pair(L.first * polyPow(R.second) + R.first.mulxk(m - l), L.second + R.
101                         second);
102    };
103    auto res = work(work, 0, a.size(), Poly{1});
104    return b.back();

```

```

104 }
105 int main() {
106     ios::sync_with_stdio(false);
107     cin.tie(nullptr);
108     string s;
109     cin >> s;
110     int n = s.size();
111     fac.resize(n + 1);
112     invfac.resize(n + 1);
113     fac[0] = 1;
114     for (int i = 1; i <= n; i++) {
115         fac[i] = fac[i - 1] * i;
116     }
117     invfac[n] = fac[n].inv();
118     for (int i = n; i; i--) {
119         invfac[i - 1] = invfac[i] * i;
120     }
121     vector<int> pre(n + 1);
122     for (int i = 0; i < n; i++) {
123         pre[i + 1] = pre[i] + (s[i] == '(' ? 1 : -1);
124     }
125     int x = min_element(pre.begin(), pre.end()) - pre.begin();
126     auto ans = solve(s.substr(x));
127     string t;
128     for (int i = x - 1; i >= 0; i--) {
129         if (s[i] == '(') t += ')';
130         else t += '(';
131     }
132     ans *= solve(t);
133     cout << ans << "\n";
134     return 0;
135 }
```

## 960: Anti-median (Hard Version)

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the hard version of the problem. The only difference between the two versions is the constraint on  $n$ . You can make hacks only if all versions of the problem are solved.

Let's call an array  $a$  of odd length  $2m + 1$  (with  $m \geq 1$ ) bad, if element  $a_{m+1}$  is equal to the median of this array. In other words, the array is bad if, after sorting it, the element at  $m + 1$ -st position remains the same.

Let's call a permutation  $p$  of integers from 1 to  $n$  anti-median, if every its subarray of odd length  $\geq 3$  is not bad.

You are already given values of some elements of the permutation. Find the number of ways to set unknown values to obtain an anti-median permutation. As this number can be very large, find it modulo  $10^9 + 7$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 1000000007;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
8     if (x < 0) {
9         x += P;
10    }
11    if (x >= P) {
12        x -= P;
13    }
14    return x;
15}
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25}
26 # struct Z;
27 constexpr int N = 4E6;
28 vector<Z> fac(N + 1), invfac(N + 1);
29 Z binom(int n, int m) {
30     if (n < m || m < 0) {
31         return 0;
32     }
33     return fac[n] * invfac[m] * invfac[n - m];
34}
35 Z get(int n, int a, int b, int c, int d) {
36     if ((c - a) % 2 != 0) {
37         return 0;
38     }
39     if (c > a || d < b) {
40         return 0;
41     }
42     if (a + b < 0 || a + b >= 2 * n) {
43         return 0;
44     }
45     if (c + d < 0 || c + d >= 2 * n) {
46         return 0;
47     }
48     Z ans = binom((a - c + d - b) / 2, (a - c) / 2);
49     ans -= binom((a - c + d - b) / 2, (a - (-d - 2)) / 2);
50     ans -= binom((a - c + d - b) / 2, (a - (-d + 2 * n)) / 2);
51     return ans;
52}
53 int init = 0;
54 vector<Z> pre;
55 Z get1(int n, int a, int b) {
56     if (a + b < 0 || a + b >= 2 * n) {
57         return 0;
58     }
59     int cur = (2 * n - 4 - (b - a)) / 2;
60     assert(cur >= 0);

```

```

61     if (!init) {
62         init = 1;
63         pre.resize(cur + 1);
64         for (int i = 0; i <= cur; i++) {
65             pre[i] = binom(cur, i);
66             if (i) {
67                 pre[i] += pre[i - 1];
68             }
69         }
70     }
71     Z ans = 0;
72     int s = cur + (a + b) / 2;
73     if (s >= 0) {
74         ans += pre[min(s / 2, cur)];
75     }
76     s = cur - (a + b) / 2 - 2;
77     if (s >= 0) {
78         ans -= pre[s / 2];
79     }
80     s = cur + (a + b) / 2 - n;
81     if (s >= 0) {
82         ans -= pre[s / 2];
83     }
84     s = cur + (a + b) / 2 - n - 1;
85     if (s >= 0) {
86         ans -= pre[s / 2];
87     }
88     return ans;
89 }
90 Z work(const vector<int> &p) {
91     const int n = p.size();
92     Z ans = 0;
93     for (int t = 0; t < 2; t++) {
94         vector<pair<array<int, 2>, Z>> dp;
95         for (int i = t; i < n; i += 2) {
96             if (p[0] == -1 || p[0] == i) {
97                 dp.emplace_back(array{i, i}, 1);
98             }
99         }
100        for (int i = 1; i < n - 1; i++) {
101            if (p[i] == -1) {
102                continue;
103            }
104            vector<pair<array<int, 2>, Z>> f;
105            if (p[i] % 2 == t) {
106                int x, y;
107                y = p[i], x = p[i] - 2 * i;
108                Z res = 0;
109                for (auto [q, v] : dp) {
110                    res += get(n, q[0], q[1], x, y - 2) * v;
111                }
112                f.emplace_back(array{x, y}, res);
113                res = 0;
114                x = p[i], y = p[i] + 2 * i;
115                for (auto [q, v] : dp) {
116                    res += get(n, q[0], q[1], x + 2, y) * v;
117                }
118                f.emplace_back(array{x, y}, res);
119            } else {
120                int x, y;
121                x = -1 - p[i], y = x + 2 * i;
122                Z res = 0;
123                for (auto [q, v] : dp) {
124                    res += get(n, q[0], q[1], x + 2, y) * v;

```

```

125             }
126             f.emplace_back(array{x, y}, res);
127             res = 0;
128             y = 2 * n - 1 - p[i], x = y - 2 * i;
129             for (auto [q, v] : dp) {
130                 res += get(n, q[0], q[1], x, y - 2) * v;
131             }
132             f.emplace_back(array{x, y}, res);
133         }
134         dp = f;
135     }
136     if (p[n - 1] == -1) {
137         init = 0;
138         for (auto [q, v] : dp) {
139             ans += get1(n, q[0], q[1]) * v;
140         }
141     } else {
142         for (auto [q, v] : dp) {
143             ans += get(n, q[0], q[1], 1 - p[n - 1], 2 * n - 3 - p[n - 1]) * v;
144         }
145     }
146 }
147 return ans;
148 }
149 void solve() {
150     int n;
151     cin >> n;
152     vector<int> p(n, -1);
153     for (int i = 0; i < n; i++) {
154         int x;
155         cin >> x;
156         x--;
157         if (x >= 0) {
158             p[x] = i;
159         }
160     }
161     auto ans = work(p);
162     cout << ans << "\n";
163 }
164 int main() {
165     ios::sync_with_stdio(false);
166     cin.tie(nullptr);
167     fac[0] = 1;
168     for (int i = 1; i <= N; i++) {
169         fac[i] = fac[i - 1] * i;
170     }
171     invfac[N] = fac[N].inv();
172     for (int i = N; i; i--) {
173         invfac[i - 1] = invfac[i] * i;
174     }
175     int t;
176     cin >> t;
177     while (t--) {
178         solve();
179     }
180     return 0;
181 }

```

### 961: Anti-median (Easy Version)

- Time limit: 2 seconds

- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is the easy version of the problem. The only difference between the two versions is the constraint on  $n$ . You can make hacks only if all versions of the problem are solved.

Let's call an array  $a$  of odd length  $2m + 1$  (with  $m \geq 1$ ) bad, if element  $a_{m+1}$  is equal to the median of this array. In other words, the array is bad if, after sorting it, the element at  $m + 1$ -st position remains the same.

Let's call a permutation  $p$  of integers from 1 to  $n$  anti-median, if every its subarray of odd length  $\geq 3$  is not bad.

You are already given values of some elements of the permutation. Find the number of ways to set unknown values to obtain an anti-median permutation. As this number can be very large, find it modulo  $10^9 + 7$ .

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 1000000007;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
8     if (x < 0) {
9         x += P;
10    }
11    if (x >= P) {
12        x -= P;
13    }
14    return x;
15 }
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25 }
26 # struct Z;
27 constexpr int N = 4E6;
28 vector<Z> fac(N + 1), invfac(N + 1);
29 Z binom(int n, int m) {
30     if (n < m || m < 0) {
31         return 0;
32     }
33     return fac[n] * invfac[m] * invfac[n - m];
34 }
```

```

35 Z work(vector<int> p) {
36     const int n = p.size();
37     Z ans = 0;
38     for (int t = 0; t < 2; t++) {
39         vector<Z> dp(n, vector<Z>(n));
40         for (int i = t; i < n; i += 2) {
41             dp[i][i] = (p[i] == 0 || p[i] == -1);
42         }
43         for (int i = n - 1 - (n % 2 == t); i >= 0; i -= 2) {
44             int x = 0;
45             for (int j = i; j < n; j += 2) {
46                 x++;
47                 if (i > j) {
48                     continue;
49                 }
50                 int k = i < 2 ? !t : i - 2;
51                 if (x >= n - 1 && (p[k] == -1 || p[k] == n - 1)) {
52                     ans += dp[i][j];
53                     continue;
54                 }
55                 if (p[k] == -1 || p[k] == x) {
56                     dp[k][j] += dp[i][j];
57                 }
58                 k = j + 2 >= n ? n - 1 - (n % 2 == !t) : j + 2;
59                 if (p[k] == -1 || p[k] == x) {
60                     dp[i][k] += dp[i][j];
61                 }
62             }
63             for (int j = n - 1 - (n % 2 == !t); j >= 0; j -= 2) {
64                 x++;
65                 if (i > j) {
66                     continue;
67                 }
68                 if (x >= n - 1 && j >= 2 && (p[j - 2] == -1 || p[j - 2] == n - 1)) {
69                     ans += dp[i][j];
70                     continue;
71                 }
72                 int k = i < 2 ? !t : i - 2;
73                 if (p[k] == -1 || p[k] == x) {
74                     dp[k][j] += dp[i][j];
75                 }
76                 if (j >= 2) {
77                     k = j - 2;
78                     if (p[k] == -1 || p[k] == x) {
79                         dp[i][k] += dp[i][j];
80                     }
81                 }
82             }
83         }
84         for (int i = !t; i < n; i += 2) {
85             int x = i / 2 + 1;
86             for (int j = t; j < n; j += 2) {
87                 x++;
88                 if (i > j) {
89                     continue;
90                 }
91                 if (x >= n - 1 && i + 2 < n && (p[i + 2] == -1 || p[i + 2] == n - 1)) {
92                     ans += dp[i][j];
93                     continue;
94                 }
95                 int k;
96                 if (i + 2 < n) {
97                     k = i + 2;
98                     if (p[k] == -1 || p[k] == x) {

```

```

99                     dp[k][j] += dp[i][j];
100                }
101            }
102            k = j + 2 >= n ? n - 1 - (n % 2 == !t) : j + 2;
103            if (p[k] == -1 || p[k] == x) {
104                dp[i][k] += dp[i][j];
105            }
106        }
107        for (int j = n - 1 - (n % 2 == !t); j > i; j -= 2) {
108            x++;
109            if (i > j) {
110                continue;
111            }
112            if (x >= n - 1 && i + 2 < n && (p[i + 2] == -1 || p[i + 2] == n - 1)) {
113                ans += dp[i][j];
114                continue;
115            }
116            int k;
117            if (i + 2 < n) {
118                k = i + 2;
119                if (p[k] == -1 || p[k] == x) {
120                    dp[k][j] += dp[i][j];
121                }
122            }
123            if (j >= 2) {
124                k = j - 2;
125                if (p[k] == -1 || p[k] == x) {
126                    dp[i][k] += dp[i][j];
127                }
128            }
129        }
130        // for (int i = 0; i < n; i++) {
131        //     for (int j = 0; j < n; j++) {
132        //         cerr << dp[i][j] << " \n"[j == n - 1];
133        //     }
134        // }
135    }
136    return ans;
137}
138 void solve() {
139     int n;
140     cin >> n;
141     vector<int> p(n);
142     for (int i = 0; i < n; i++) {
143         cin >> p[i];
144         if (p[i] != -1) {
145             p[i]--;
146         }
147     }
148     auto ans = work(p);
149     cout << ans << "\n";
150 }
151 int main() {
152     ios::sync_with_stdio(false);
153     cin.tie(nullptr);
154     fac[0] = 1;
155     for (int i = 1; i <= N; i++) {
156         fac[i] = fac[i - 1] * i;
157     }
158     invfac[N] = fac[N].inv();
159     for (int i = N; i; i--) {
160         invfac[i - 1] = invfac[i] * i;
161     }
162 }
```

```
163     int t;
164     cin >> t;
165     while (t--) {
166         solve();
167     }
168     return 0;
169 }
```

## misc

### 962: Indefinite Clownfish

- Time limit: 3.5 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Pak Chanek has just bought an empty fish tank and he has been dreaming to fill it with his favourite kind of fish, clownfish. Pak Chanek likes clownfish because of their ability to change their genders on demand. Because of the size of his fish tank, Pak Chanek wants to buy exactly  $k$  clownfish to fill the tank.

Pak Chanek goes to the local fish shop. The shop provides  $n$  clownfish numbered from 1 to  $n$ , with clownfish  $i$  having a size of  $a_i$ . Initially, every clownfish in the store does not have an assigned gender, but has the ability to be assigned to two possible clownfish genders, female or male.

The store has a procedure which Pak Chanek should follow to buy clownfish. The shop owner will point at each clownfish sequentially from 1 to  $n$  and for each clownfish, she asks Pak Chanek whether to buy it or not. Each time Pak Chanek is asked, he must declare whether to buy the currently asked clownfish or not, before the shop owner moves on to the next clownfish. If Pak Chanek declares to buy the currently asked clownfish, he must also declare the gender to be assigned to that clownfish immediately. When assigning the gender for the currently asked clownfish, the following conditions must be satisfied:

If Pak Chanek assigns it to be female and he has bought a female clownfish before, then the size of the current one must be exactly 1 bigger than the last female one.

If Pak Chanek assigns it to be male and he has bought a male clownfish before, then the size of the current one must be exactly 1 smaller than the last male one.

Pak Chanek wants to buy exactly  $k$  clownfish such that:

There is at least one female clownfish and one male clownfish.

Among the  $k$  clownfish Pak Chanek buys, the mean size of the female clownfish is equal to the mean size of the male clownfish.

Let  $l$  and  $r$  respectively be the minimum and maximum index of a clownfish Pak Chanek buys. What is the minimum possible value of  $r - l$ ?

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
5     ios::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, k;
8     cin >> n >> k;
9     if (k % 2 != 0) {
10         cout << -1 << "\n";
11         return 0;
12     }
13     vector<int> a(n);
14     for (int i = 0; i < n; i++) {
15         cin >> a[i];
16         a[i]--;
17     }
18     auto get = [&]() {
19         vector f(2, vector(3, vector<int>(n, -1)));
20         vector p(2, vector(3, vector(20, vector<int>(n, -1))));
21         vector<int> lst(n, -1);
22         for (int i = 0; i < n; i++) {
23             for (int x = 0; x < 3; x++) {
24                 int y = a[i] - 1 + x;
25                 if (y >= 0 && y < n) {
26                     f[0][x][i] = lst[y];
27                 }
28             }
29             lst[a[i]] = i;
30         }
31         lst.assign(n, -1);
32         for (int i = n - 1; i >= 0; i--) {
33             for (int x = 0; x < 3; x++) {
34                 int y = a[i] - 1 + x;
35                 if (y >= 0 && y < n) {
36                     f[1][x][i] = lst[y];
37                 }
38             }
39             lst[a[i]] = i;
40         }
41         for (int x = 0; x < 2; x++) {
42             for (int y = 0; y < 3; y++) {
43                 p[x][y][0] = f[x][y];
44                 for (int i = 0; i < 19; i++) {
45                     for (int j = 0; j < n; j++) {
46                         if (p[x][y][i][j] != -1) {
47                             p[x][y][i + 1][j] = p[x][y][i][p[x][y][i][j]];
48                         }
49                     }
50                 }
51             }
52         }
53     };
54 }
```

```

53     int ans = n;
54     auto get = [&](int s, int x, int y, int k) {
55         if (k == 0) {
56             return s;
57         }
58         for (int i = __lg(k); i >= 0; i--) {
59             if (k >> i & 1) {
60                 if (s != -1) {
61                     s = p[x][y][i][s];
62                 }
63             }
64         }
65         if (x == 1 && s == -1) {
66             s = n;
67         }
68         return s;
69     };
70     if (k >= 4) {
71         for (int x = 0; x < n; x++) {
72             int y = f[1][2][x];
73             int z = f[0][2][x];
74             int w = f[1][1][x];
75             if (y == -1 || z == -1 || w == -1) {
76                 continue;
77             }
78             int t = (k - 4) / 2;
79             int j = *ranges::partition_point(ranges::iota_view(0, t),
80                                             [&](int v) {
81                                                 return get(x, 0, 0, v) > get(z, 0, 2, t - v);
82                                             });
83             int L = -1;
84             for (auto v : {j, j - 1}) {
85                 if (0 <= v && v <= t) {
86                     L = max(L, min(get(x, 0, 0, v), get(z, 0, 2, t - v)));
87                 }
88             }
89             j = *ranges::partition_point(ranges::iota_view(0, t),
90                                         [&](int v) {
91                                             return get(w, 1, 0, v) < get(y, 1, 2, t - v);
92                                         });
93             int R = n;
94             for (auto v : {j, j - 1}) {
95                 if (0 <= v && v <= t) {
96                     R = min(R, max(get(w, 1, 0, v), get(y, 1, 2, t - v)));
97                 }
98             }
99             if (L != -1 && R != n) {
100                 ans = min(ans, R - L);
101             }
102         }
103     }
104     for (int x = 0; x < n; x++) {
105         int y = f[1][1][x];
106         if (y == -1) {
107             continue;
108         }
109         int t = (k - 2) / 2;
110         int L = get(x, 0, 2, t);
111         int j = *ranges::partition_point(ranges::iota_view(0, t),
112                                         [&](int v) {
113                                             return get(x, 1, 0, v) < get(y, 1, 2, t - v);
114                                         });
115         int R = n;
116         for (auto v : {j, j - 1}) {

```

```

117         if (0 <= v && v <= t) {
118             R = min(R, max(get(x, 1, 0, v), get(y, 1, 2, t - v)));
119         }
120     }
121     if (L != -1 && R != n) {
122         ans = min(ans, R - L);
123     }
124 }
125 for (int x = 0; x < n; x++) {
126     int y = f[0][1][x];
127     if (y == -1) {
128         continue;
129     }
130     int t = (k - 2) / 2;
131     int R = get(x, 1, 2, t);
132     int j = *ranges::partition_point(ranges::iota_view(0, t),
133                                     [&](int v) {
134                                         return get(x, 0, 0, v) > get(y, 0, 2, t - v);
135                                     });
136     int L = -1;
137     for (auto v : {j, j - 1}) {
138         if (0 <= v && v <= t) {
139             L = max(L, min(get(x, 0, 0, v), get(y, 0, 2, t - v)));
140         }
141     }
142     if (L != -1 && R != n) {
143         ans = min(ans, R - L);
144     }
145 }
146 return ans;
147 };
148 auto ans = get();
149 for (auto &x : a) {
150     x = n - 1 - x;
151 }
152 ans = min(ans, get());
153 if (ans == n) {
154     ans = -1;
155 }
156 cout << ans << "\n";
157 return 0;
158 }

```

### 963: Minimums or Medians

- Time limit: 4 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Vika has a set of all consecutive positive integers from 1 to  $2n$ , inclusive.

Exactly  $k$  times Vika will choose and perform one of the following two actions:

take two smallest integers from her current set and remove them;

take two median integers from her current set and remove them.

Recall that medians are the integers located exactly in the middle of the set if you write down its elements in increasing order. Note that Vika's set always has an even size, thus the pair of median integers is uniquely defined. For example, two median integers of the set  $\{1, 5, 6, 10, 15, 16, 18, 23\}$  are 10 and 15.

How many different sets can Vika obtain in the end, after  $k$  actions? Print this number modulo 998 244 353. Two sets are considered different if some integer belongs to one of them but not to the other.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int P = 998244353;
5 using i64 = long long;
6 // assume -P <= x < 2P
7 int norm(int x) {
8     if (x < 0) {
9         x += P;
10    }
11    if (x >= P) {
12        x -= P;
13    }
14    return x;
15 }
16 template<class T>
17 T power(T a, i64 b) {
18     T res = 1;
19     for (; b; b /= 2, a *= a) {
20         if (b % 2) {
21             res *= a;
22         }
23     }
24     return res;
25 }
26 # struct Z;
27 # struct Comb comb;
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(nullptr);
31     int n, k;
32     cin >> n >> k;
33     Z ans = 0;
34     for (int i = 0; i <= k; i++) {
35         if (i == k) {
36             ans += 1;
37         } else if (2 * i < n) {
38             // for (int j = 0; i + j <= k && 2 * i + j < n; j++) {
39             //     ans += comb.binom(i, k - i - j);
40             // }
41         } else {
42             ans += comb.binom(n - k - 1 + k - i, k - i);
43         }
44     }
45     for (int s = 0; s <= k; s++) {

```

```

46     int mini = k - s;
47     int maxi = min({s, (n - 1) / 2, n - 1 - s, k - 1});
48     if (mini <= maxi) {
49         ans += comb.binom(maxi + 1, k - s + 1);
50         ans -= comb.binom(mini, k - s + 1);
51         // for (int i = mini; i <= maxi; i++) {
52         //     ans += comb.binom(i, k - s);
53         // }
54     }
55 }
56 cout << ans << "\n";
57 return 0;
58 }
```

## 964: Olympic Team Building

- Time limit: 2 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

Iron and Werewolf are participating in a chess Olympiad, so they want to practice team building. They gathered  $n$  players, where  $n$  is a power of 2, and they will play sports. Iron and Werewolf are among those  $n$  people.

One of the sports is tug of war. For each  $1 \leq i \leq n$ , the  $i$ -th player has strength  $s_i$ . Elimination rounds will be held until only one player remains - we call that player the absolute winner.

In each round:

Assume that  $m > 1$  players are still in the game, where  $m$  is a power of 2.

The  $m$  players are split into two teams of equal sizes (i. e., with  $m/2$  players in each team). The strength of a team is the sum of the strengths of its players.

If the teams have equal strengths, Iron chooses who wins; otherwise, the stronger team wins.

Every player in the losing team is eliminated, so  $m/2$  players remain.

Iron already knows each player's strength and is wondering who can become the absolute winner and who can't if he may choose how the teams will be formed in each round, as well as the winning team in case of equal strengths.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 int main() {
```

```

5   ios::sync_with_stdio(false);
6   cin.tie(nullptr);
7   int n;
8   cin >> n;
9   vector<i64> s(n);
10  for (int i = 0; i < n; i++) {
11      cin >> s[i];
12  }
13  i64 sum = accumulate(s.begin(), s.end(), 0LL);
14  vector<i64> vals = s;
15  sort(vals.begin(), vals.end());
16  int cnt = 0;
17  auto check = [&](auto check, int k, i64 asum, vector<i64> a) -> int {
18      int n = a.size();
19      if (k == n) {
20          return asum * 2 >= sum;
21      }
22      if (k == 2) {
23          for (int i = 0; i < n; i++) {
24              for (int j = i + 1; j < n; j++) {
25                  i64 bsum = a[i] + a[j];
26                  if (bsum > asum) break;
27                  if (i + 1 < j && a[i + 1] + a[j] <= asum) continue;
28                  if (j + 1 < n && a[i] + a[j + 1] <= asum) continue;
29                  auto na = a;
30                  na.erase(na.begin() + j);
31                  na.erase(na.begin() + i);
32                  if (check(check, 4, asum + bsum, na)) return 1;
33              }
34          }
35          return 0;
36      }
37      if (k == 4) {
38          cnt++;
39          for (int i = 0; i < n; i++) {
40              for (int j = i + 1; j < n; j++) {
41                  for (int x = j + 1; x < n; x++) {
42                      for (int y = x + 1; y < n; y++) {
43                          i64 bsum = a[i] + a[j] + a[x] + a[y];
44                          if (bsum > asum) break;
45                          if (i + 1 < j && a[i + 1] + a[j] + a[x] + a[y] <= asum)
46                              continue;
47                          if (j + 1 < x && a[i] + a[j + 1] + a[x] + a[y] <= asum)
48                              continue;
49                          if (x + 1 < y && a[i] + a[j] + a[x + 1] + a[y] <= asum)
50                              continue;
51                          if (y + 1 < n && a[i] + a[j] + a[x] + a[y + 1] <= asum)
52                              continue;
53                          auto na = a;
54                          na.erase(na.begin() + y);
55                          na.erase(na.begin() + x);
56                          na.erase(na.begin() + j);
57                          na.erase(na.begin() + i);
58                          if (check(check, 8, asum + bsum, na)) return 1;
59                      }
60                  }
61              }
62          }
63          return 0;
64      }
65      if (k == 8) {
66          vector<i64> sums[9];
67          if (4 * asum < sum) return 0;
68          assert(n == 24);
69      }
70  }

```

```

65         vector<i64> val(1 << 12);
66         for (int s = 0; s < (1 << 12); s++) {
67             if (s) {
68                 int u = __builtin_ctz(s);
69                 val[s] = val[s ^ (1 << u)] + a[u];
70             }
71             if (int c = __builtin_popcount(s); c <= 8) {
72                 sums[c].push_back(val[s]);
73             }
74         }
75         for (int i = 0; i <= 8; i++) {
76             sort(sums[i].begin(), sums[i].end());
77         }
78         for (int s = 0; s < (1 << 12); s++) {
79             if (s) {
80                 int u = __builtin_ctz(s);
81                 val[s] = val[s ^ (1 << u)] + a[12 + u];
82             }
83             if (int c = __builtin_popcount(s); c <= 8) {
84                 auto &v = sums[8 - c];
85                 auto it = upper_bound(v.begin(), v.end(), asum - val[s]);
86                 if (it != v.begin() && (asum + val[s] + *(it - 1)) * 2 >= sum) return
87                     1;
88             }
89         }
90     }
91     return 0;
92 };
93 int lo = 1, hi = n - 1;
94 while (lo < hi) {
95     int x = (lo + hi) / 2;
96     vector<i64> res = vals;
97     res.erase(res.begin() + x);
98     res.erase(res.begin() + x - 1);
99     cnt = 0;
100    if (check(check, 2, vals[x - 1] + vals[x], res)) hi = x;
101    else lo = x + 1;
102 }
103 for (int i = 0; i < n; i++) {
104     cout << (s[i] >= vals[lo]);
105 }
106 cout << "\n";
107 return 0;
108 }
```

### 965: Doremy's Perfect DS Class (Hard Version)

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The only difference between this problem and the other two versions is the maximum number of queries. In this version, you are allowed to ask at most 20 queries. You can make hacks only if all versions of the problem are solved.

This is an interactive problem.

“Everybody! Doremy’s Perfect Data Structure Class is about to start! Come and do your best if you want to have as much IQ as me!” In today’s Data Structure class, Doremy is teaching everyone a powerful data structure - Doremy tree! Now she gives you a quiz to prove that you are paying attention in class.

Given an array  $a$  of length  $m$ , Doremy tree supports the query  $Q(l, r, k)$ , where  $1 \leq l \leq r \leq m$  and  $1 \leq k \leq m$ , which returns the number of distinct integers in the array  $\lfloor \frac{a_l}{k} \rfloor, \lfloor \frac{a_{l+1}}{k} \rfloor, \dots, \lfloor \frac{a_r}{k} \rfloor$ .

Doremy has a secret permutation  $p$  of integers from 1 to  $n$ . You can make queries, in one query, you give 3 integers  $l, r, k$  ( $1 \leq l \leq r \leq n, 1 \leq k \leq n$ ) and receive the value of  $Q(l, r, k)$  for the array  $p$ . Can you find the index  $y$  ( $1 \leq y \leq n$ ) such that  $p_y = 1$  in at most **20** queries?

Note that the permutation  $p$  is fixed before any queries are made.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 map<tuple<int, int, int>, int> F;
5 int n;
6 int Q(int l, int r, int k) {
7     if (l == r) {
8         return 0;
9     }
10    if (r - l == n) {
11        int s = 0;
12        for (int i = 1; i <= n; i++) {
13            s += i / k;
14        }
15        return s;
16    }
17    if (F.count({l, r, k})) {
18        return F[{l, r, k}];
19    }
20    cout << "? " << l + 1 << " " << r << " " << k << endl;
21    int res;
22    cin >> res;
23    return F[{l, r, k}] = res;
24}
25 int main() {
26     ios::sync_with_stdio(false);
27     cin.tie(nullptr);
28     cin >> n;
29     int pn = -1;
30     int s = n / 2 + 1;
31     int l = 0, r = n;
32     while (r - l > 1 + (n % 2 == 0 && n > 500)) {
33         int m = (l + r) / 2;
34         if (n % 2 == 1 || pn != -1) {
35             int ql = Q(0, m, 2);
36             int qr = Q(m, n, 2);
37             int lenl = m;
38             int lenr = n - m;
39             if (0 <= pn && pn < m) {
40                 ql--;
41             }
42             if (pn <= m && pn >= n - m) {
43                 qr--;
44             }
45             if (ql == qr) {
46                 cout << "!" << endl;
47                 exit(0);
48             }
49         }
50     }
51 }
```

```

41             lenl--;
42         }
43     if (m <= pn && pn < n) {
44         qr--;
45         lenr--;
46     }
47     int both = ql + qr - s;
48     assert((lenl - both) % 2 != (lenr - both) % 2);
49     // cerr << lenl - both - (ql - both) * 2 << " " << lenr - both - (qr - both) *
50     //      2 << "\n";
51     if ((lenl - both) % 2 != 0) {
52         r = m;
53     } else {
54         l = m;
55     }
56 } else {
57     int ql = Q(0, m, 2);
58     int qr = Q(m, n, 2);
59     int lenl = m;
60     int lenr = n - m;
61     int both = ql + qr - s;
62     int vl = lenl + both - ql * 2;
63     int vr = lenr + both - qr * 2;
64     assert(vl <= 0);
65     assert(vr <= 0);
66     assert(vl + vr == -2);
67     if (vl == -2) {
68         r = m;
69         continue;
70     }
71     if (vr == -2) {
72         l = m;
73         continue;
74     }
75     int t = 0;
76     if (m >= 2) {
77         if (Q(0, m, n) == 1) {
78             t = 1;
79         }
80     } else {
81         if (Q(m, n, n) == 2) {
82             t = 1;
83         }
84     }
85     if (t == 0) {
86         pn = l;
87         l = m;
88     } else {
89         pn = m;
90         r = m;
91     }
92     s--;
93 }
94 if (r - l == 2) {
95     int m = l + 1;
96     if (pn == -1) {
97         if (m >= 2 ? Q(0, m, n) == 1 : Q(m, n, n) == 2) {
98             r = m;
99         } else {
100             l = m;
101         }
102     } else {
103         if (Q(0, r, 2) - Q(0, l, 2) == 2) {

```

```

104         if (Q(m, n, 2) == Q(r, n, 2)) {
105             r = m;
106         } else {
107             l = m;
108         }
109     } else {
110         if (Q(0, m, 2) == Q(0, l, 2)) {
111             l = m;
112         } else {
113             r = m;
114         }
115     }
116 }
117 cout << "! " << l + 1 << endl;
118 return 0;
120 }
```

## 966: Minecraft Series

- Time limit: 6 seconds
- Memory limit: 512 megabytes
- Input file: standard input
- Output file: standard output

Little Misha goes to the programming club and solves nothing there. It may seem strange, but when you find out that Misha is filming a Minecraft series, everything will fall into place...

Misha is inspired by Manhattan, so he built a city in Minecraft that can be imagined as a table of size  $n \times m$ .  $k$  students live in a city, the  $i$ -th student lives in the house, located at the intersection of the  $x_i$ -th row and the  $y_i$ -th column. Also, each student has a degree of his aggressiveness  $w_i$ . Since the city turned out to be very large, Misha decided to territorially limit the actions of his series to some square  $s$ , which sides are parallel to the coordinate axes. The length of the side of the square should be an integer from 1 to  $\min(n, m)$  cells.

According to the plot, the main hero will come to the city and accidentally fall into the square  $s$ . Possessing a unique degree of aggressiveness 0, he will be able to show his leadership qualities and assemble a team of calm, moderate and aggressive students.

In order for the assembled team to be versatile and close-knit, degrees of aggressiveness of all students of the team must be pairwise distinct and must form a single segment of consecutive integers. Formally, if there exist students with degrees of aggressiveness  $l, l + 1, \dots, -1, 1, \dots, r - 1, r$  inside the square  $s$ , where  $l \leq 0 \leq r$ , the main hero will be able to form a team of  $r - l + 1$  people (of course, he is included in this team).

Notice, that it is not required to take all students from square  $s$  to the team.

Misha thinks that the team should consist of at least  $t$  people. That is why he is interested, how many squares are there in the table in which the main hero will be able to form a team of at least  $t$  people. Help him to calculate this.

Problem: [link](#)

Tutorial: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 constexpr int K = 1E6 + 2;
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, m, k, t;
9     cin >> n >> m >> k >> t;
10    const int N = n * m;
11    vector<vector<int>> ap(N), an(N);
12    bitset<K> fp, fn;
13    for (int i = 1; i <= t; i++) {
14        fp[i] = fn[i] = 1;
15    }
16    vector<int> cp(t + 1), cn(t + 1);
17    cp[0] = cn[0] = 1;
18    for (int i = 0; i < k; i++) {
19        int x, y, z;
20        cin >> x >> y >> z;
21        x--, y--;
22        if (z && z > -t && z < t) {
23            if (z > 0) {
24                ap[x * m + y].push_back(z);
25            } else {
26                an[x * m + y].push_back(-z);
27            }
28        }
29    }
30    auto add = [&](int x, int y) {
31        for (auto i : ap[x * m + y]) {
32            if (!cp[i]++) {
33                fp[i] = 0;
34            }
35        }
36        for (auto i : an[x * m + y]) {
37            if (!cn[i]++) {
38                fn[i] = 0;
39            }
40        }
41    };
42    auto del = [&](int x, int y) {
43        for (auto i : ap[x * m + y]) {
44            if (!--cp[i]) {
45                fp[i] = 1;
46            }
47        }
48        for (auto i : an[x * m + y]) {
49            if (!--cn[i]) {
50                fn[i] = 1;
51            }
52        }
53    };

```

```

53     };
54     auto check = [&]() {
55         return fp._Find_first() + fn._Find_first() >= t + 1;
56     };
57     vector<int> len(n, vector<int>(m));
58     i64 ans = 0;
59     for (int i = 0; i < n; i++) {
60         int x0 = i, x1 = i, y0 = 0, y1 = 0;
61         int l = 0;
62         fp = 1;
63         fp = fn = ~fp;
64         fill(cp.begin() + 1, cp.end(), 0);
65         fill(cn.begin() + 1, cn.end(), 0);
66         for (int j = 0; j < m; j++) {
67             if (l && (i == 0 || l >= len[i - 1][j])) {
68                 l--;
69             }
70             l = min({l, n - i, m - j});
71             while (y1 < j + l) {
72                 for (int i = x0; i < x1; i++) {
73                     add(i, y1);
74                 }
75                 y1++;
76             }
77             while (y0 < j) {
78                 for (int i = x0; i < x1; i++) {
79                     del(i, y0);
80                 }
81                 y0++;
82             }
83             while (x1 > i + l) {
84                 x1--;
85                 for (int i = y0; i < y1; i++) {
86                     del(x1, i);
87                 }
88             }
89             while (!l || !check()) {
90                 l++;
91                 if (i + l > n || j + l > m) {
92                     break;
93                 }
94                 while (y1 < j + l) {
95                     for (int i = x0; i < x1; i++) {
96                         add(i, y1);
97                     }
98                     y1++;
99                 }
100                while (x1 < i + l) {
101                    for (int i = y0; i < y1; i++) {
102                        add(x1, i);
103                    }
104                    x1++;
105                }
106            }
107            ans += min(n - i, m - j) - l + 1;
108            len[i][j] = l;
109        }
110    }
111    cout << ans << "\n";
112    return 0;
113 }

```

## 967: Joking (Hard Version)

- Time limit: 1 second
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

The only difference between this problem and the hard version is the maximum number of questions.

This is an interactive problem.

There is a hidden integer  $1 \leq x \leq n$  which you have to find. In order to find it you can ask at most 53 questions.

In each question you can choose a non-empty integer set  $S$  and ask if  $x$  belongs to  $S$  or not, after each question, if  $x$  belongs to  $S$ , you'll receive "YES", otherwise "NO".

But the problem is that not all answers are necessarily true (some of them are joking), it's just guaranteed that for each two consecutive questions, at least one of them is answered correctly.

Additionally to the questions, you can make at most 2 guesses for the answer  $x$ . Each time you make a guess, if you guess  $x$  correctly, you receive ":" and your program should terminate, otherwise you'll receive ":".

As a part of the joking, we will not fix the value of  $x$  in the beginning. Instead, it can change throughout the interaction as long as all the previous responses are valid as described above.

Note that your answer guesses are always answered correctly. If you ask a question before and after a guess, at least one of these two questions is answered correctly, as normal.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 void dfs(int a, int b, int k) {
5     if (k == 0) {
6         if (a + b > 2) {
7             cerr << a << " " << b << "\n";
8         }
9         return;
10    }
11    if (a <= 2 * b) {
12        int x = (a + 1) / 2, y = b / 2;
13        dfs(x + y, a - x, k - 1);
14        dfs(a - x + b - y, x, k - 1);
15    } else {
16        dfs(a, 0, k - 1);
17        dfs(b, a, k - 1);
18    }

```

```

19 }
20 int dp[121][121];
21 int gx[121][121], gy[121][121];
22 int main() {
23     ios::sync_with_stdio(false);
24     cin.tie(nullptr);
25     memset(dp, 0x3f, sizeof(dp));
26     for (int s = 0; s <= 120; s++) {
27         for (int i = 0; i <= s; i++) {
28             const int j = s - i;
29             if (s <= 2) {
30                 dp[i][j] = 0;
31             } else {
32                 for (int x = 0; x <= i; x++) {
33                     for (int y = 0; y <= j; y++) {
34                         int t = max(dp[x + y][i - x], dp[i - x + j - y][x]) + 1;
35                         if (t < dp[i][j]) {
36                             dp[i][j] = t;
37                             gx[i][j] = x;
38                             gy[i][j] = y;
39                         }
40                     }
41                 }
42             }
43         }
44     }
45     int n;
46     cin >> n;
47     vector<int> T, F;
48     for (int i = 1; i <= n; i++) {
49         T.push_back(i);
50     }
51     while (T.size() + F.size() > 2) {
52         int x, y;
53         if (T.size() + F.size() <= 120) {
54             x = gx[T.size()][F.size()];
55             y = gy[T.size()][F.size()];
56         } else {
57             x = T.size() / 2;
58             y = F.size() / 2;
59         }
60         vector T1(T.begin(), T.begin() + x), T0(T.begin() + x, T.end());
61         vector F1(F.begin(), F.begin() + y), F0(F.begin() + y, F.end());
62         cout << "? " << x + y;
63         for (auto x : T1) {
64             cout << " " << x;
65         }
66         for (auto x : F1) {
67             cout << " " << x;
68         }
69         cout << endl;
70         string s;
71         cin >> s;
72         T = F = {};
73         if (s == "YES") {
74             for (auto x : T1) {
75                 T.push_back(x);
76             }
77             for (auto x : F1) {
78                 T.push_back(x);
79             }
80             for (auto x : T0) {
81                 F.push_back(x);
82             }

```

```

83         } else {
84             for (auto x : T0) {
85                 T.push_back(x);
86             }
87             for (auto x : F0) {
88                 T.push_back(x);
89             }
90             for (auto x : T1) {
91                 F.push_back(x);
92             }
93         }
94     }
95     for (auto x : T) {
96         cout << "! " << x << endl;
97         string s;
98         cin >> s;
99         if (s == ":)") {
100             return 0;
101         }
102     }
103     for (auto x : F) {
104         cout << "! " << x << endl;
105         string s;
106         cin >> s;
107         if (s == ":)") {
108             return 0;
109         }
110     }
111 }
112 }
```

## trees

### 968: Roads in E City

- Time limit: 3 seconds
- Memory limit: 256 megabytes
- Input file: standard input
- Output file: standard output

This is an interactive problem.

As is well known, the city “E” has never had its roads repaired in its a thousand and a half years old history. And only recently the city administration repaired some of them.

It is known that in total in the city “E” there are  $n$  intersections and  $m$  roads, which can be used in both directions, numbered with integers from 1 to  $m$ . The  $i$ -th road connects intersections with numbers  $a_i$  and  $b_i$ .

Among all  $m$  roads, some subset of the roads has been repaired, but you do not know which one. The only information you could get from the city’s road services is that you can get from any intersection to any other intersection by driving only on the roads that have been repaired.

You are a young entrepreneur, and decided to organize a delivery service of fresh raw meat in the city "E" (in this city such meat is called "steaks", it is very popular among the locals). You have already recruited a staff of couriers, but the couriers are willing to travel only on repaired roads. Now you have to find out which roads have already been repaired.

The city administration has given you the city for a period of time, so you can make different queries of one of three types:

Block the road with the number  $x$ . In this case, movement on the road for couriers will be forbidden. Initially all roads are unblocked.

Unblock the road with the number  $x$ . In this case, couriers will be able to move on the road  $x$  if it is repaired.

Try to deliver the order to the intersection with the number  $y$ . In this case, one of your couriers will start moving from intersection with number  $s$  you don't know and deliver the order to intersection with number  $y$  if there is a path on unblocked repaired roads from intersection  $s$  to intersection  $y$ . It is guaranteed that intersection  $s$  will be chosen beforehand.

Unfortunately, the city is placed at your complete disposal for a short period of time, so you can make no more than  $100 \cdot m$  requests.

Problem: [link](#)

Solution: [link](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
5 bool query(int x, int y) {
6     for (int i = 0; i < 40; i++) {
7         cout << "? " << (rng() % 2 ? x : y) + 1 << endl;
8         int res;
9         cin >> res;
10        if (res == 0) {
11            return false;
12        }
13    }
14    return true;
15 }
16 void solve() {
17     int n, m;
18     cin >> n >> m;
19     vector<int> u(m), v(m);
20     for (int i = 0; i < m; i++) {
21         cin >> u[i] >> v[i];
22         u[i]--, v[i]--;
23     }
24     vector<bool> ok(m);
25     for (int i = 0; i < m; i++) {
26         cout << "- " << i + 1 << endl;
27         if (!query(u[i], v[i])) {
28             ok[i] = true;
29             cout << "+ " << i + 1 << endl;
30         }
31     }
32 }
```

```
30         }
31     }
32     vector<vector<pair<int, int>>> adj(n);
33     for (int i = 0; i < m; i++) {
34         if (ok[i]) {
35             adj[u[i]].emplace_back(v[i], i);
36             adj[v[i]].emplace_back(u[i], i);
37         }
38     }
39     vector<int> parent(n), dep(n), pe(n);
40     function<void(int)> dfs = [&](int x) {
41         for (auto [y, i] : adj[x]) {
42             if (y == parent[x]) {
43                 continue;
44             }
45             pe[y] = i;
46             parent[y] = x;
47             dep[y] = dep[x] + 1;
48             dfs(y);
49         }
50     };
51     parent[0] = -1;
52     dfs(0);
53     for (int i = 0; i < m; i++) {
54         if (!ok[i]) {
55             if (dep[u[i]] < dep[v[i]]) {
56                 swap(u[i], v[i]);
57             }
58             cout << "- " << pe[u[i]] + 1 << endl;
59             cout << "+ " << i + 1 << endl;
60             if (query(u[i], v[i])) {
61                 ok[i] = true;
62             }
63             cout << "+ " << pe[u[i]] + 1 << endl;
64             cout << "- " << i + 1 << endl;
65         }
66     }
67     cout << "!";
68     for (int i = 0; i < m; i++) {
69         cout << " " << ok[i];
70     }
71     cout << endl;
72     int res;
73     cin >> res;
74 }
75 int main() {
76     ios::sync_with_stdio(false);
77     cin.tie(nullptr);
78     int t;
79     cin >> t;
80     while (t--) {
81         solve();
82     }
83     return 0;
84 }
```

## Appendix: some common template code

### MInt and MLong

The M in the struct name is for “modular arithmetic”. Whenever a problem asks for “submit answer modulo 1e9+7 or 998244353”, this is the data structure to use. This is the most popular template by far, appearing in ~10% of problems.

```

1  struct MInt {
2      int x;
3      constexpr MInt() : x{} {}
4      constexpr MInt(i64 x) : x{norm(x % P)} {}
5
6      constexpr int norm(int x) const {
7          if (x < 0) {
8              x += P;
9          }
10         if (x >= P) {
11             x -= P;
12         }
13         return x;
14     }
15     constexpr int val() const {
16         return x;
17     }
18     explicit constexpr operator int() const {
19         return x;
20     }
21     constexpr MInt operator-() const {
22         MInt res;
23         res.x = norm(P - x);
24         return res;
25     }
26     constexpr MInt inv() const {
27         assert(x != 0);
28         return power(*this, P - 2);
29     }
30     constexpr MInt &operator*=(MInt rhs) {
31         x = 1LL * x * rhs.x % P;
32         return *this;
33     }
34     constexpr MInt &operator+=(MInt rhs) {
35         x = norm(x + rhs.x);
36         return *this;
37     }
38     constexpr MInt &operator-=(MInt rhs) {
39         x = norm(x - rhs.x);
40         return *this;
41     }
42     constexpr MInt &operator/=(MInt rhs) {
43         return *this *= rhs.inv();
44     }
45     friend constexpr MInt operator*(MInt lhs, MInt rhs) {
46         MInt res = lhs;
47         res *= rhs;
48         return res;
49     }
50     friend constexpr MInt operator+(MInt lhs, MInt rhs) {
51         MInt res = lhs;
52         res += rhs;
53     }

```

```
53         return res;
54     }
55     friend constexpr MInt operator-(MInt lhs, MInt rhs) {
56         MInt res = lhs;
57         res -= rhs;
58         return res;
59     }
60     friend constexpr MInt operator/(MInt lhs, MInt rhs) {
61         MInt res = lhs;
62         res /= rhs;
63         return res;
64     }
65     friend constexpr istream &operator>>(istream &is, MInt &a) {
66         i64 v;
67         is >> v;
68         a = MInt(v);
69         return is;
70     }
71     friend constexpr ostream &operator<<(ostream &os, const MInt &a) {
72         return os << a.val();
73     }
74     friend constexpr bool operator==(MInt lhs, MInt rhs) {
75         return lhs.val() == rhs.val();
76     }
77     friend constexpr bool operator!=(MInt lhs, MInt rhs) {
78         return lhs.val() != rhs.val();
79     }
80 }
81
82 struct MLong {
83     i64 x;
84     constexpr MLong() : x{} {}
85     constexpr MLong(i64 x) : x{norm(x % getMod())} {}
86
87     static i64 Mod;
88     constexpr static i64 getMod() {
89         if (P > 0) {
90             return P;
91         } else {
92             return Mod;
93         }
94     }
95     constexpr static void setMod(i64 Mod_) {
96         Mod = Mod_;
97     }
98     constexpr i64 norm(i64 x) const {
99         if (x < 0) {
100             x += getMod();
101         }
102         if (x >= getMod()) {
103             x -= getMod();
104         }
105         return x;
106     }
107     constexpr i64 val() const {
108         return x;
109     }
110     explicit constexpr operator i64() const {
111         return x;
112     }
113     constexpr MLong operator-() const {
114         MLong res;
115         res.x = norm(getMod() - x);
116         return res;
```

```

117     }
118     constexpr MLong inv() const {
119         assert(x != 0);
120         return power(*this, getMod() - 2);
121     }
122     constexpr MLong &operator*=(MLong rhs) & {
123         x = mul(x, rhs.x, getMod());
124         return *this;
125     }
126     constexpr MLong &operator+=(MLong rhs) & {
127         x = norm(x + rhs.x);
128         return *this;
129     }
130     constexpr MLong &operator-=(MLong rhs) & {
131         x = norm(x - rhs.x);
132         return *this;
133     }
134     constexpr MLong &operator/=(MLong rhs) & {
135         return *this *= rhs.inv();
136     }
137     friend constexpr MLong operator*(MLong lhs, MLong rhs) {
138         MLong res = lhs;
139         res *= rhs;
140         return res;
141     }
142     friend constexpr MLong operator+(MLong lhs, MLong rhs) {
143         MLong res = lhs;
144         res += rhs;
145         return res;
146     }
147     friend constexpr MLong operator-(MLong lhs, MLong rhs) {
148         MLong res = lhs;
149         res -= rhs;
150         return res;
151     }
152     friend constexpr MLong operator/(MLong lhs, MLong rhs) {
153         MLong res = lhs;
154         res /= rhs;
155         return res;
156     }
157     friend constexpr istream &operator>>(istream &is, MLong &a) {
158         i64 v;
159         is >> v;
160         a = MLong(v);
161         return is;
162     }
163     friend constexpr ostream &operator<<(ostream &os, const MLong &a) {
164         return os << a.val();
165     }
166     friend constexpr bool operator==(MLong lhs, MLong rhs) {
167         return lhs.val() == rhs.val();
168     }
169     friend constexpr bool operator!=(MLong lhs, MLong rhs) {
170         return lhs.val() != rhs.val();
171     }
172 }
```

## Comb

For combinatorial math like binomials.

```

1  struct Comb {
2      int n;
3      vector<Z> _fac;
4      vector<Z> _invfac;
5      vector<Z> _inv;
6
7      Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
8      Comb(int n) : Comb() {
9          init(n);
10     }
11
12     void init(int m) {
13         if (m <= n) return;
14         _fac.resize(m + 1);
15         _invfac.resize(m + 1);
16         _inv.resize(m + 1);
17
18         for (int i = n + 1; i <= m; i++) {
19             _fac[i] = _fac[i - 1] * i;
20         }
21         _invfac[m] = _fac[m].inv();
22         for (int i = m; i > n; i--) {
23             _invfac[i - 1] = _invfac[i] * i;
24             _inv[i] = _invfac[i] * _fac[i - 1];
25         }
26         n = m;
27     }
28
29     Z fac(int m) {
30         if (m > n) init(2 * m);
31         return _fac[m];
32     }
33     Z invfac(int m) {
34         if (m > n) init(2 * m);
35         return _invfac[m];
36     }
37     Z inv(int m) {
38         if (m > n) init(2 * m);
39         return _inv[m];
40     }
41     Z binom(int n, int m) {
42         if (n < m || m < 0) return 0;
43         return fac(n) * invfac(m) * invfac(n - m);
44     }
45 }
```

## DSU

The short implementation of DSU, also known as “union-find”.

```

1  struct DSU {
2      vector<int> f, siz;
3
4      DSU() {}
5      DSU(int n) {
6          init(n);
7      }
8
9      void init(int n) {
10         f.resize(n);
```

```

11         iota(f.begin(), f.end(), 0);
12         siz.assign(n, 1);
13     }
14
15     int find(int x) {
16         while (x != f[x]) {
17             x = f[x] = f[f[x]];
18         }
19         return x;
20     }
21
22     bool same(int x, int y) {
23         return find(x) == find(y);
24     }
25
26     bool merge(int x, int y) {
27         x = find(x);
28         y = find(y);
29         if (x == y) {
30             return false;
31         }
32         siz[x] += siz[y];
33         f[y] = x;
34         return true;
35     }
36
37     int size(int x) {
38         return siz[find(x)];
39     }
40 }
```

## Fenwick

Fenwick is also known as a Binary Index Tree.

```

1 struct Fenwick {
2     int n;
3     vector<T> a;
4
5     Fenwick(int n = 0) {
6         init(n);
7     }
8
9     void init(int n) {
10        this->n = n;
11        a.assign(n, T());
12    }
13
14     void add(int x, T v) {
15         for (int i = x + 1; i <= n; i += i & -i) {
16             a[i - 1] += v;
17         }
18     }
19
20     T sum(int x) {
21         auto ans = T();
22         for (int i = x; i > 0; i -= i & -i) {
23             ans += a[i - 1];
24         }
25         return ans;
26     }
}
```

```

27     T rangeSum(int l, int r) {
28         return sum(r) - sum(l);
29     }
30
31     int kth(T k) {
32         int x = 0;
33         for (int i = 1 << __lg(n); i; i /= 2) {
34             if (x + i <= n && k >= a[x + i - 1]) {
35                 x += i;
36                 k -= a[x - 1];
37             }
38         }
39         return x;
40     }
41 }
42 }
```

**HLD****HLD**

```

1 struct HLD {
2     int n;
3     vector<int> siz, top, dep, parent, in, out, seq;
4     vector<vector<int>> adj;
5     int cur;
6
7     HLD() {}
8     HLD(int n) {
9         init(n);
10    }
11    void init(int n) {
12        this->n = n;
13        siz.resize(n);
14        top.resize(n);
15        dep.resize(n);
16        parent.resize(n);
17        in.resize(n);
18        out.resize(n);
19        seq.resize(n);
20        cur = 0;
21        adj.assign(n, {});
22    }
23    void addEdge(int u, int v) {
24        adj[u].push_back(v);
25        adj[v].push_back(u);
26    }
27    void work(int root = 0) {
28        top[root] = root;
29        dep[root] = 0;
30        parent[root] = -1;
31        dfs1(root);
32        dfs2(root);
33    }
34    void dfs1(int u) {
35        if (parent[u] != -1) {
36            adj[u].erase(find(adj[u].begin(), adj[u].end(), parent[u]));
37        }
38        siz[u] = 1;
39        for (auto &v : adj[u]) {
```

```

41         parent[v] = u;
42         dep[v] = dep[u] + 1;
43         dfs1(v);
44         siz[u] += siz[v];
45         if (siz[v] > siz[adj[u][0]]) {
46             swap(v, adj[u][0]);
47         }
48     }
49 }
50 void dfs2(int u) {
51     in[u] = cur++;
52     seq[in[u]] = u;
53     for (auto v : adj[u]) {
54         top[v] = v == adj[u][0] ? top[u] : v;
55         dfs2(v);
56     }
57     out[u] = cur;
58 }
59 int lca(int u, int v) {
60     while (top[u] != top[v]) {
61         if (dep[top[u]] > dep[top[v]]) {
62             u = parent[top[u]];
63         } else {
64             v = parent[top[v]];
65         }
66     }
67     return dep[u] < dep[v] ? u : v;
68 }
69 int dist(int u, int v) {
70     return dep[u] + dep[v] - 2 * dep[lca(u, v)];
71 }
72
73 int jump(int u, int k) {
74     if (dep[u] < k) {
75         return -1;
76     }
77
78     int d = dep[u] - k;
79
80     while (dep[top[u]] > d) {
81         u = parent[top[u]];
82     }
83
84     return seq[in[u] - dep[u] + d];
85 }
86
87 bool isAncestor(int u, int v) {
88     return in[u] <= in[v] && in[v] < out[u];
89 }
90
91 int rootedParent(int u, int v) {
92     swap(u, v);
93     if (u == v) {
94         return u;
95     }
96     if (!isAncestor(u, v)) {
97         return parent[u];
98     }
99     auto it = upper_bound(adj[u].begin(), adj[u].end(), v, [&](int x, int y) {
100         return in[x] < in[y];
101     }) - 1;
102     return *it;
103 }
104 }
```

```

105     int rootedSize(int u, int v) {
106         if (u == v) {
107             return n;
108         }
109         if (!isAncestor(v, u)) {
110             return siz[v];
111         }
112         return n - siz[rootedParent(u, v)];
113     }
114
115     int rootedLca(int a, int b, int c) {
116         return lca(a, b) ^ lca(b, c) ^ lca(c, a);
117     }
118 }
119 }
```

## LazySegmentTree

### Lazy Segment Tree

```

1 struct LazySegmentTree {
2     int n;
3     vector<Info> info;
4     vector<Tag> tag;
5     LazySegmentTree() : n(0) {}
6     LazySegmentTree(int n_, Info v_ = Info()) {
7         init(n_, v_);
8     }
9     template<class T>
10    LazySegmentTree(vector<T> init_) {
11        init(init_);
12    }
13    void init(int n_, Info v_ = Info()) {
14        init(vector(n_, v_));
15    }
16    template<class T>
17    void init(vector<T> init_) {
18        n = init_.size();
19        info.assign(4 << __lg(n), Info());
20        tag.assign(4 << __lg(n), Tag());
21        function<void(int, int, int)> build = [&](int p, int l, int r) {
22            if (r - l == 1) {
23                info[p] = init_[l];
24                return;
25            }
26            int m = (l + r) / 2;
27            build(2 * p, l, m);
28            build(2 * p + 1, m, r);
29            pull(p);
30        };
31        build(1, 0, n);
32    }
33    void pull(int p) {
34        info[p] = info[2 * p] + info[2 * p + 1];
35    }
36    void apply(int p, const Tag &v) {
37        info[p].apply(v);
38        tag[p].apply(v);
39    }
40    void push(int p) {
41        apply(2 * p, tag[p]);
```

```
42         apply(2 * p + 1, tag[p]);
43         tag[p] = Tag();
44     }
45     void modify(int p, int l, int r, int x, const Info &v) {
46         if (r - l == 1) {
47             info[p] = v;
48             return;
49         }
50         int m = (l + r) / 2;
51         push(p);
52         if (x < m) {
53             modify(2 * p, l, m, x, v);
54         } else {
55             modify(2 * p + 1, m, r, x, v);
56         }
57         pull(p);
58     }
59     void modify(int p, const Info &v) {
60         modify(1, 0, n, p, v);
61     }
62     Info rangeQuery(int p, int l, int r, int x, int y) {
63         if (l >= y || r <= x) {
64             return Info();
65         }
66         if (l >= x && r <= y) {
67             return info[p];
68         }
69         int m = (l + r) / 2;
70         push(p);
71         return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
72     }
73     Info rangeQuery(int l, int r) {
74         return rangeQuery(1, 0, n, l, r);
75     }
76     void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
77         if (l >= y || r <= x) {
78             return;
79         }
80         if (l >= x && r <= y) {
81             apply(p, v);
82             return;
83         }
84         int m = (l + r) / 2;
85         push(p);
86         rangeApply(2 * p, l, m, x, y, v);
87         rangeApply(2 * p + 1, m, r, x, y, v);
88         pull(p);
89     }
90     void rangeApply(int l, int r, const Tag &v) {
91         return rangeApply(1, 0, n, l, r, v);
92     }
93     template<class F>
94     int findFirst(int p, int l, int r, int x, int y, F pred) {
95         if (l >= y || r <= x || !pred(info[p])) {
96             return -1;
97         }
98         if (r - l == 1) {
99             return l;
100        }
101        int m = (l + r) / 2;
102        push(p);
103        int res = findFirst(2 * p, l, m, x, y, pred);
104        if (res == -1) {
105            res = findFirst(2 * p + 1, m, r, x, y, pred);
106        }
107    }
108 }
```

```
106         }
107     }
108 }
109 template<class F>
110 int findFirst(int l, int r, F pred) {
111     return findFirst(1, 0, n, l, r, pred);
112 }
113 template<class F>
114 int findLast(int p, int l, int r, int x, int y, F pred) {
115     if (l >= y || r <= x || !pred(info[p])) {
116         return -1;
117     }
118     if (r - l == 1) {
119         return l;
120     }
121     int m = (l + r) / 2;
122     push(p);
123     int res = findLast(2 * p + 1, m, r, x, y, pred);
124     if (res == -1) {
125         res = findLast(2 * p, l, m, x, y, pred);
126     }
127     return res;
128 }
129 template<class F>
130 int findLast(int l, int r, F pred) {
131     return findLast(1, 0, n, l, r, pred);
132 }
133 }
```