

Contents

1 Misc	1	4.1.13 Rational Number Binary Search	14
1.1 Contest	1	4.1.14 Farey Sequence	14
1.1.1 Makefile	1	4.2 Combinatorics	14
1.2 How Did We Get Here?	2	4.2.1 Matroid Intersection	14
1.2.1 Macros	2	4.2.2 De Bruijn Sequence	15
1.2.2 Fast I/O	2	4.2.3 Multinomial	15
1.2.2.1 Kotlin	2	4.3 Theorems	15
1.2.3 Bump Allocator	2	Kirchhoff's Theorem	15
1.3 Tools	2	Tutte's Matrix	15
1.3.1 Floating Point Binary Search	2	Cayley's Formula	15
1.3.2 SplitMix64	2	Erdős-Gallai Theorem	15
1.3.3 <random>	2	Gale-Ryser Theorem	15
1.3.4 x86 Stack Hack	2	Burnside's Lemma	15
1.3.5 ctypes	3	Gram-Schmidt Process	15
1.4 Algorithms	3	5 Numeric	15
1.4.1 Bit Hacks	3	5.1 Barrett Reduction	15
1.4.2 Aliens Trick	3	5.2 Long Long Multiplication	16
1.4.3 Hilbert Curve	3	5.3 Fast Fourier Transform	16
1.4.4 Longest Increasing Subsequence	3	5.4 Fast Walsh-Hadamard Transform	16
1.4.5 Mo's Algorithm on Tree	3	5.5 Subset Convolution	16
2 Data Structures	3	5.6 Linear Recurrences	16
2.1 GNU PBDS	3	5.6.1 Berlekamp-Massey Algorithm	16
2.2 Segment Tree (ZKW)	3	5.6.2 Linear Recurrence Calculation	16
2.3 Line Container	3	5.7 Matrices	16
2.4 Li-Chao Tree	4	5.7.1 Determinant	16
2.5 adamant HLD	4	5.7.2 Inverse	17
2.6 van Emde Boas Tree	4	5.7.3 Characteristic Polynomial	17
2.7 Wavelet Matrix	5	5.7.4 Solve Linear Equation	17
2.8 Link-Cut Tree	5	5.8 Polynomial Interpolation	18
3 Graph	6	5.9 Simplex Algorithm	18
3.1 Modeling	6	6 Geometry	19
3.2 Matching/Flows	6	6.1 Point	19
3.2.1 Dinic's Algorithm	6	6.1.1 Quaternion	19
3.2.2 Minimum Cost Flow	7	6.1.2 Spherical Coordinates	19
3.2.3 Gomory-Hu Tree	7	6.2 Segments	19
3.2.4 Global Minimum Cut	7	6.3 Convex Hull	19
3.2.5 Bipartite Minimum Cover	7	6.3.1 3D Hull	19
3.2.6 Edmonds' Algorithm	8	6.4 Angular Sort	20
3.2.7 Minimum Weight Matching	8	6.5 Convex Hull Tangent	20
3.2.8 Stable Marriage	9	6.6 Convex Polygon Minkowski Sum	20
3.2.9 Kuhn-Munkres algorithm	9	6.7 Point In Polygon	20
3.3 Shortest Path Faster Algorithm	9	6.7.1 Convex Version	20
3.4 Strongly Connected Components	10	6.7.2 Offline Multiple Points Version	21
3.4.1 2-Satisfiability	10	6.8 Closest Pair	21
3.5 Biconnected Components	10	6.9 Minimum Enclosing Circle	21
3.5.1 Articulation Points	10	6.10 Delaunay Triangulation	22
3.5.2 Bridges	10	6.10.1 Quadratic Time Version	22
3.6 Triconnected Components	10	6.11 Half Plane Intersection	22
3.7 Centroid Decomposition	11	7 Strings	23
3.8 Minimum Mean Cycle	11	7.1 Knuth-Morris-Pratt Algorithm	23
3.9 Directed MST	11	7.2 Aho-Corasick Automaton	23
3.10 Maximum Clique	12	7.3 Suffix Array	23
3.11 Dominator Tree	12	7.4 Suffix Tree	24
3.12 Manhattan Distance MST	12	7.5 Cocke-Younger-Kasami Algorithm	24
3.13 Virtual Tree	13	7.6 Z Value	24
4 Math	13	7.7 Manacher's Algorithm	25
4.1 Number Theory	13	7.8 Lyndon Factorization	25
4.1.1 Mod Struct	13	7.9 Palindromic Tree	25
4.1.2 Miller-Rabin	13		
4.1.3 Linear Sieve	13		
4.1.4 Get Factors	13		
4.1.5 Binary GCD	13		
4.1.6 Extended GCD	13		
4.1.7 Chinese Remainder Theorem	14		
4.1.8 Baby-Step Giant-Step	14		
4.1.9 Pohlig-Hellman Algorithm	14		
4.1.10 Pollard's Rho	14		
4.1.11 Tonelli-Shanks Algorithm	14		
4.1.12 Chinese Sieve	14		

1 Misc

1.1 Contest

1.1.1 Makefile

```
1 .PRECIOUS: ./p%
3 %: p%
    ulimit -s unlimited && setarch -R ./p%
5 p%: p%.cpp
    g++ -o $@ $< -std=gnu++20 -Wall -Wextra -Wshadow \
7      -g -fsanitize=address,undefined
```

1.2 How Did We Get Here?

1.2.1 Macros

```
1 #define _GLIBCXX_DEBUG 1 // for debug mode
2 #define _GLIBCXX_SANITIZE_VECTOR 1 // for asan on vectors
3 #pragma GCC optimize("O3", "unroll-loops")
4 #pragma GCC optimize("fast-math")
5 #pragma GCC target("avx,avx2,abm,bmi,bmi2") // tip: `lscpu`
// before a loop
6 #pragma GCC unroll 16 // 0 or 1 -> no unrolling
7 #pragma GCC ivdep
```

1.2.2 Fast I/O

```
1 struct scanner {
2     static constexpr size_t LEN = 32 << 20;
3     char *buf, *buf_ptr, *buf_end;
4     scanner()
5         : buf(new char[LEN]), buf_ptr(buf + LEN),
6           buf_end(buf + LEN) {}
7     ~scanner() { delete[] buf; }
8     char getc() {
9         if (buf_ptr == buf_end) [[unlikely]]
10             buf_end = buf + fread_unlocked(buf, 1, LEN, stdin),
11             buf_ptr = buf;
12         return *(buf_ptr++);
13     }
14     char seek(char del) {
15         char c;
16         while ((c = getc()) < del) {}
17         return c;
18     }
19     void read(int &t) {
20         bool neg = false;
21         char c = seek('-');
22         if (c == '-') neg = true, t = 0;
23         else t = c ^ '0';
24         while ((c = getc()) >= '0') t = t * 10 + (c ^ '0');
25         if (neg) t = -t;
26     }
27 };
28
29 struct printer {
30     static constexpr size_t CPI = 21, LEN = 32 << 20;
31     char *buf, *buf_ptr, *buf_end, *tbuf;
32     char *int_buf, *int_buf_end;
33     printer()
34         : buf(new char[LEN]), buf_ptr(buf),
35           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
36           int_buf_end(int_buf + CPI - 1) {}
37     ~printer() {
38         flush();
39         delete[] buf, delete[] int_buf;
40     }
41     void flush() {
42         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
43         buf_ptr = buf;
44     }
45     void write_(const char &c) {
46         *buf_ptr = c;
47         if (++buf_ptr == buf_end) [[unlikely]]
48             flush();
49     }
50     void write_(const char *s) {
51         for (; *s != '\0'; ++s) write_(*s);
52     }
53     void write(int x) {
54         if (x < 0) write_('-'), x = -x;
55         if (x == 0) [[unlikely]]
56             return write_('0');
57         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
58             *tbuf = '0' + char(x % 10);
59         write_(++tbuf);
60     }
61 };
62
63 struct kotlin {
64     static constexpr size_t LEN = 32 << 20;
65     char *buf, *buf_ptr, *buf_end, *tbuf;
66     char *int_buf, *int_buf_end;
67     kotlin()
68         : buf(new char[LEN]), buf_ptr(buf),
69           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
70           int_buf_end(int_buf + CPI - 1) {}
71     ~kotlin() {
72         flush();
73         delete[] buf, delete[] int_buf;
74     }
75     void flush() {
76         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
77         buf_ptr = buf;
78     }
79     void write_(const char &c) {
80         *buf_ptr = c;
81         if (++buf_ptr == buf_end) [[unlikely]]
82             flush();
83     }
84     void write_(const char *s) {
85         for (; *s != '\0'; ++s) write_(*s);
86     }
87     void write(int x) {
88         if (x < 0) write_('-'), x = -x;
89         if (x == 0) [[unlikely]]
90             return write_('0');
91         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
92             *tbuf = '0' + char(x % 10);
93         write_(++tbuf);
94     }
95 };
96
97 struct kotlin2 {
98     static constexpr size_t LEN = 32 << 20;
99     char *buf, *buf_ptr, *buf_end, *tbuf;
100     char *int_buf, *int_buf_end;
101     kotlin2()
102         : buf(new char[LEN]), buf_ptr(buf),
103           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
104           int_buf_end(int_buf + CPI - 1) {}
105     ~kotlin2() {
106         flush();
107         delete[] buf, delete[] int_buf;
108     }
109     void flush() {
110         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
111         buf_ptr = buf;
112     }
113     void write_(const char &c) {
114         *buf_ptr = c;
115         if (++buf_ptr == buf_end) [[unlikely]]
116             flush();
117     }
118     void write_(const char *s) {
119         for (; *s != '\0'; ++s) write_(*s);
120     }
121     void write(int x) {
122         if (x < 0) write_('-'), x = -x;
123         if (x == 0) [[unlikely]]
124             return write_('0');
125         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
126             *tbuf = '0' + char(x % 10);
127         write_(++tbuf);
128     }
129 };
130
131 struct kotlin3 {
132     static constexpr size_t LEN = 32 << 20;
133     char *buf, *buf_ptr, *buf_end, *tbuf;
134     char *int_buf, *int_buf_end;
135     kotlin3()
136         : buf(new char[LEN]), buf_ptr(buf),
137           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
138           int_buf_end(int_buf + CPI - 1) {}
139     ~kotlin3() {
140         flush();
141         delete[] buf, delete[] int_buf;
142     }
143     void flush() {
144         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
145         buf_ptr = buf;
146     }
147     void write_(const char &c) {
148         *buf_ptr = c;
149         if (++buf_ptr == buf_end) [[unlikely]]
150             flush();
151     }
152     void write_(const char *s) {
153         for (; *s != '\0'; ++s) write_(*s);
154     }
155     void write(int x) {
156         if (x < 0) write_('-'), x = -x;
157         if (x == 0) [[unlikely]]
158             return write_('0');
159         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
160             *tbuf = '0' + char(x % 10);
161         write_(++tbuf);
162     }
163 };
164
165 struct kotlin4 {
166     static constexpr size_t LEN = 32 << 20;
167     char *buf, *buf_ptr, *buf_end, *tbuf;
168     char *int_buf, *int_buf_end;
169     kotlin4()
170         : buf(new char[LEN]), buf_ptr(buf),
171           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
172           int_buf_end(int_buf + CPI - 1) {}
173     ~kotlin4() {
174         flush();
175         delete[] buf, delete[] int_buf;
176     }
177     void flush() {
178         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
179         buf_ptr = buf;
180     }
181     void write_(const char &c) {
182         *buf_ptr = c;
183         if (++buf_ptr == buf_end) [[unlikely]]
184             flush();
185     }
186     void write_(const char *s) {
187         for (; *s != '\0'; ++s) write_(*s);
188     }
189     void write(int x) {
190         if (x < 0) write_('-'), x = -x;
191         if (x == 0) [[unlikely]]
192             return write_('0');
193         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
194             *tbuf = '0' + char(x % 10);
195         write_(++tbuf);
196     }
197 };
198
199 struct kotlin5 {
200     static constexpr size_t LEN = 32 << 20;
201     char *buf, *buf_ptr, *buf_end, *tbuf;
202     char *int_buf, *int_buf_end;
203     kotlin5()
204         : buf(new char[LEN]), buf_ptr(buf),
205           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
206           int_buf_end(int_buf + CPI - 1) {}
207     ~kotlin5() {
208         flush();
209         delete[] buf, delete[] int_buf;
210     }
211     void flush() {
212         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
213         buf_ptr = buf;
214     }
215     void write_(const char &c) {
216         *buf_ptr = c;
217         if (++buf_ptr == buf_end) [[unlikely]]
218             flush();
219     }
220     void write_(const char *s) {
221         for (; *s != '\0'; ++s) write_(*s);
222     }
223     void write(int x) {
224         if (x < 0) write_('-'), x = -x;
225         if (x == 0) [[unlikely]]
226             return write_('0');
227         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
228             *tbuf = '0' + char(x % 10);
229         write_(++tbuf);
230     }
231 };
232
233 struct kotlin6 {
234     static constexpr size_t LEN = 32 << 20;
235     char *buf, *buf_ptr, *buf_end, *tbuf;
236     char *int_buf, *int_buf_end;
237     kotlin6()
238         : buf(new char[LEN]), buf_ptr(buf),
239           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
240           int_buf_end(int_buf + CPI - 1) {}
241     ~kotlin6() {
242         flush();
243         delete[] buf, delete[] int_buf;
244     }
245     void flush() {
246         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
247         buf_ptr = buf;
248     }
249     void write_(const char &c) {
250         *buf_ptr = c;
251         if (++buf_ptr == buf_end) [[unlikely]]
252             flush();
253     }
254     void write_(const char *s) {
255         for (; *s != '\0'; ++s) write_(*s);
256     }
257     void write(int x) {
258         if (x < 0) write_('-'), x = -x;
259         if (x == 0) [[unlikely]]
260             return write_('0');
261         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
262             *tbuf = '0' + char(x % 10);
263         write_(++tbuf);
264     }
265 };
266
267 struct kotlin7 {
268     static constexpr size_t LEN = 32 << 20;
269     char *buf, *buf_ptr, *buf_end, *tbuf;
270     char *int_buf, *int_buf_end;
271     kotlin7()
272         : buf(new char[LEN]), buf_ptr(buf),
273           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
274           int_buf_end(int_buf + CPI - 1) {}
275     ~kotlin7() {
276         flush();
277         delete[] buf, delete[] int_buf;
278     }
279     void flush() {
280         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
281         buf_ptr = buf;
282     }
283     void write_(const char &c) {
284         *buf_ptr = c;
285         if (++buf_ptr == buf_end) [[unlikely]]
286             flush();
287     }
288     void write_(const char *s) {
289         for (; *s != '\0'; ++s) write_(*s);
290     }
291     void write(int x) {
292         if (x < 0) write_('-'), x = -x;
293         if (x == 0) [[unlikely]]
294             return write_('0');
295         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
296             *tbuf = '0' + char(x % 10);
297         write_(++tbuf);
298     }
299 };
300
301 struct kotlin8 {
302     static constexpr size_t LEN = 32 << 20;
303     char *buf, *buf_ptr, *buf_end, *tbuf;
304     char *int_buf, *int_buf_end;
305     kotlin8()
306         : buf(new char[LEN]), buf_ptr(buf),
307           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
308           int_buf_end(int_buf + CPI - 1) {}
309     ~kotlin8() {
310         flush();
311         delete[] buf, delete[] int_buf;
312     }
313     void flush() {
314         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
315         buf_ptr = buf;
316     }
317     void write_(const char &c) {
318         *buf_ptr = c;
319         if (++buf_ptr == buf_end) [[unlikely]]
320             flush();
321     }
322     void write_(const char *s) {
323         for (; *s != '\0'; ++s) write_(*s);
324     }
325     void write(int x) {
326         if (x < 0) write_('-'), x = -x;
327         if (x == 0) [[unlikely]]
328             return write_('0');
329         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
330             *tbuf = '0' + char(x % 10);
331         write_(++tbuf);
332     }
333 };
334
335 struct kotlin9 {
336     static constexpr size_t LEN = 32 << 20;
337     char *buf, *buf_ptr, *buf_end, *tbuf;
338     char *int_buf, *int_buf_end;
339     kotlin9()
340         : buf(new char[LEN]), buf_ptr(buf),
341           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
342           int_buf_end(int_buf + CPI - 1) {}
343     ~kotlin9() {
344         flush();
345         delete[] buf, delete[] int_buf;
346     }
347     void flush() {
348         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
349         buf_ptr = buf;
350     }
351     void write_(const char &c) {
352         *buf_ptr = c;
353         if (++buf_ptr == buf_end) [[unlikely]]
354             flush();
355     }
356     void write_(const char *s) {
357         for (; *s != '\0'; ++s) write_(*s);
358     }
359     void write(int x) {
360         if (x < 0) write_('-'), x = -x;
361         if (x == 0) [[unlikely]]
362             return write_('0');
363         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
364             *tbuf = '0' + char(x % 10);
365         write_(++tbuf);
366     }
367 };
368
369 struct kotlin10 {
370     static constexpr size_t LEN = 32 << 20;
371     char *buf, *buf_ptr, *buf_end, *tbuf;
372     char *int_buf, *int_buf_end;
373     kotlin10()
374         : buf(new char[LEN]), buf_ptr(buf),
375           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
376           int_buf_end(int_buf + CPI - 1) {}
377     ~kotlin10() {
378         flush();
379         delete[] buf, delete[] int_buf;
380     }
381     void flush() {
382         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
383         buf_ptr = buf;
384     }
385     void write_(const char &c) {
386         *buf_ptr = c;
387         if (++buf_ptr == buf_end) [[unlikely]]
388             flush();
389     }
390     void write_(const char *s) {
391         for (; *s != '\0'; ++s) write_(*s);
392     }
393     void write(int x) {
394         if (x < 0) write_('-'), x = -x;
395         if (x == 0) [[unlikely]]
396             return write_('0');
397         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
398             *tbuf = '0' + char(x % 10);
399         write_(++tbuf);
400     }
401 };
402
403 struct kotlin11 {
404     static constexpr size_t LEN = 32 << 20;
405     char *buf, *buf_ptr, *buf_end, *tbuf;
406     char *int_buf, *int_buf_end;
407     kotlin11()
408         : buf(new char[LEN]), buf_ptr(buf),
409           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
410           int_buf_end(int_buf + CPI - 1) {}
411     ~kotlin11() {
412         flush();
413         delete[] buf, delete[] int_buf;
414     }
415     void flush() {
416         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
417         buf_ptr = buf;
418     }
419     void write_(const char &c) {
420         *buf_ptr = c;
421         if (++buf_ptr == buf_end) [[unlikely]]
422             flush();
423     }
424     void write_(const char *s) {
425         for (; *s != '\0'; ++s) write_(*s);
426     }
427     void write(int x) {
428         if (x < 0) write_('-'), x = -x;
429         if (x == 0) [[unlikely]]
430             return write_('0');
431         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
432             *tbuf = '0' + char(x % 10);
433         write_(++tbuf);
434     }
435 };
436
437 struct kotlin12 {
438     static constexpr size_t LEN = 32 << 20;
439     char *buf, *buf_ptr, *buf_end, *tbuf;
440     char *int_buf, *int_buf_end;
441     kotlin12()
442         : buf(new char[LEN]), buf_ptr(buf),
443           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
444           int_buf_end(int_buf + CPI - 1) {}
445     ~kotlin12() {
446         flush();
447         delete[] buf, delete[] int_buf;
448     }
449     void flush() {
450         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
451         buf_ptr = buf;
452     }
453     void write_(const char &c) {
454         *buf_ptr = c;
455         if (++buf_ptr == buf_end) [[unlikely]]
456             flush();
457     }
458     void write_(const char *s) {
459         for (; *s != '\0'; ++s) write_(*s);
460     }
461     void write(int x) {
462         if (x < 0) write_('-'), x = -x;
463         if (x == 0) [[unlikely]]
464             return write_('0');
465         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
466             *tbuf = '0' + char(x % 10);
467         write_(++tbuf);
468     }
469 };
470
471 struct kotlin13 {
472     static constexpr size_t LEN = 32 << 20;
473     char *buf, *buf_ptr, *buf_end, *tbuf;
474     char *int_buf, *int_buf_end;
475     kotlin13()
476         : buf(new char[LEN]), buf_ptr(buf),
477           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
478           int_buf_end(int_buf + CPI - 1) {}
479     ~kotlin13() {
480         flush();
481         delete[] buf, delete[] int_buf;
482     }
483     void flush() {
484         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
485         buf_ptr = buf;
486     }
487     void write_(const char &c) {
488         *buf_ptr = c;
489         if (++buf_ptr == buf_end) [[unlikely]]
490             flush();
491     }
492     void write_(const char *s) {
493         for (; *s != '\0'; ++s) write_(*s);
494     }
495     void write(int x) {
496         if (x < 0) write_('-'), x = -x;
497         if (x == 0) [[unlikely]]
498             return write_('0');
499         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
500             *tbuf = '0' + char(x % 10);
501         write_(++tbuf);
502     }
503 };
504
505 struct kotlin14 {
506     static constexpr size_t LEN = 32 << 20;
507     char *buf, *buf_ptr, *buf_end, *tbuf;
508     char *int_buf, *int_buf_end;
509     kotlin14()
510         : buf(new char[LEN]), buf_ptr(buf),
511           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
512           int_buf_end(int_buf + CPI - 1) {}
513     ~kotlin14() {
514         flush();
515         delete[] buf, delete[] int_buf;
516     }
517     void flush() {
518         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
519         buf_ptr = buf;
520     }
521     void write_(const char &c) {
522         *buf_ptr = c;
523         if (++buf_ptr == buf_end) [[unlikely]]
524             flush();
525     }
526     void write_(const char *s) {
527         for (; *s != '\0'; ++s) write_(*s);
528     }
529     void write(int x) {
530         if (x < 0) write_('-'), x = -x;
531         if (x == 0) [[unlikely]]
532             return write_('0');
533         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
534             *tbuf = '0' + char(x % 10);
535         write_(++tbuf);
536     }
537 };
538
539 struct kotlin15 {
540     static constexpr size_t LEN = 32 << 20;
541     char *buf, *buf_ptr, *buf_end, *tbuf;
542     char *int_buf, *int_buf_end;
543     kotlin15()
544         : buf(new char[LEN]), buf_ptr(buf),
545           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
546           int_buf_end(int_buf + CPI - 1) {}
547     ~kotlin15() {
548         flush();
549         delete[] buf, delete[] int_buf;
550     }
551     void flush() {
552         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
553         buf_ptr = buf;
554     }
555     void write_(const char &c) {
556         *buf_ptr = c;
557         if (++buf_ptr == buf_end) [[unlikely]]
558             flush();
559     }
560     void write_(const char *s) {
561         for (; *s != '\0'; ++s) write_(*s);
562     }
563     void write(int x) {
564         if (x < 0) write_('-'), x = -x;
565         if (x == 0) [[unlikely]]
566             return write_('0');
567         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
568             *tbuf = '0' + char(x % 10);
569         write_(++tbuf);
570     }
571 };
572
573 struct kotlin16 {
574     static constexpr size_t LEN = 32 << 20;
575     char *buf, *buf_ptr, *buf_end, *tbuf;
576     char *int_buf, *int_buf_end;
577     kotlin16()
578         : buf(new char[LEN]), buf_ptr(buf),
579           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
580           int_buf_end(int_buf + CPI - 1) {}
581     ~kotlin16() {
582         flush();
583         delete[] buf, delete[] int_buf;
584     }
585     void flush() {
586         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
587         buf_ptr = buf;
588     }
589     void write_(const char &c) {
590         *buf_ptr = c;
591         if (++buf_ptr == buf_end) [[unlikely]]
592             flush();
593     }
594     void write_(const char *s) {
595         for (; *s != '\0'; ++s) write_(*s);
596     }
597     void write(int x) {
598         if (x < 0) write_('-'), x = -x;
599         if (x == 0) [[unlikely]]
600             return write_('0');
601         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
602             *tbuf = '0' + char(x % 10);
603         write_(++tbuf);
604     }
605 };
606
607 struct kotlin17 {
608     static constexpr size_t LEN = 32 << 20;
609     char *buf, *buf_ptr, *buf_end, *tbuf;
610     char *int_buf, *int_buf_end;
611     kotlin17()
612         : buf(new char[LEN]), buf_ptr(buf),
613           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
614           int_buf_end(int_buf + CPI - 1) {}
615     ~kotlin17() {
616         flush();
617         delete[] buf, delete[] int_buf;
618     }
619     void flush() {
620         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
621         buf_ptr = buf;
622     }
623     void write_(const char &c) {
624         *buf_ptr = c;
625         if (++buf_ptr == buf_end) [[unlikely]]
626             flush();
627     }
628     void write_(const char *s) {
629         for (; *s != '\0'; ++s) write_(*s);
630     }
631     void write(int x) {
632         if (x < 0) write_('-'), x = -x;
633         if (x == 0) [[unlikely]]
634             return write_('0');
635         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
636             *tbuf = '0' + char(x % 10);
637         write_(++tbuf);
638     }
639 };
640
641 struct kotlin18 {
642     static constexpr size_t LEN = 32 << 20;
643     char *buf, *buf_ptr, *buf_end, *tbuf;
644     char *int_buf, *int_buf_end;
645     kotlin18()
646         : buf(new char[LEN]), buf_ptr(buf),
647           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
648           int_buf_end(int_buf + CPI - 1) {}
649     ~kotlin18() {
650         flush();
651         delete[] buf, delete[] int_buf;
652     }
653     void flush() {
654         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
655         buf_ptr = buf;
656     }
657     void write_(const char &c) {
658         *buf_ptr = c;
659         if (++buf_ptr == buf_end) [[unlikely]]
660             flush();
661     }
662     void write_(const char *s) {
663         for (; *s != '\0'; ++s) write_(*s);
664     }
665     void write(int x) {
666         if (x < 0) write_('-'), x = -x;
667         if (x == 0) [[unlikely]]
668             return write_('0');
669         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
670             *tbuf = '0' + char(x % 10);
671         write_(++tbuf);
672     }
673 };
674
675 struct kotlin19 {
676     static constexpr size_t LEN = 32 << 20;
677     char *buf, *buf_ptr, *buf_end, *tbuf;
678     char *int_buf, *int_buf_end;
679     kotlin19()
680         : buf(new char[LEN]), buf_ptr(buf),
681           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
682           int_buf_end(int_buf + CPI - 1) {}
683     ~kotlin19() {
684         flush();
685         delete[] buf, delete[] int_buf;
686     }
687     void flush() {
688         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
689         buf_ptr = buf;
690     }
691     void write_(const char &c) {
692         *buf_ptr = c;
693         if (++buf_ptr == buf_end) [[unlikely]]
694             flush();
695     }
696     void write_(const char *s) {
697         for (; *s != '\0'; ++s) write_(*s);
698     }
699     void write(int x) {
700         if (x < 0) write_('-'), x = -x;
701         if (x == 0) [[unlikely]]
702             return write_('0');
703         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
704             *tbuf = '0' + char(x % 10);
705         write_(++tbuf);
706     }
707 };
708
709 struct kotlin20 {
710     static constexpr size_t LEN = 32 << 20;
711     char *buf, *buf_ptr, *buf_end, *tbuf;
712     char *int_buf, *int_buf_end;
713     kotlin20()
714         : buf(new char[LEN]), buf_ptr(buf),
715           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
716           int_buf_end(int_buf + CPI - 1) {}
717     ~kotlin20() {
718         flush();
719         delete[] buf, delete[] int_buf;
720     }
721     void flush() {
722         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
723         buf_ptr = buf;
724     }
725     void write_(const char &c) {
726         *buf_ptr = c;
727         if (++buf_ptr == buf_end) [[unlikely]]
728             flush();
729     }
730     void write_(const char *s) {
731         for (; *s != '\0'; ++s) write_(*s);
732     }
733     void write(int x) {
734         if (x < 0) write_('-'), x = -x;
735         if (x == 0) [[unlikely]]
736             return write_('0');
737         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
738             *tbuf = '0' + char(x % 10);
739         write_(++tbuf);
740     }
741 };
742
743 struct kotlin21 {
744     static constexpr size_t LEN = 32 << 20;
745     char *buf, *buf_ptr, *buf_end, *tbuf;
746     char *int_buf, *int_buf_end;
747     kotlin21()
748         : buf(new char[LEN]), buf_ptr(buf),
749           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
750           int_buf_end(int_buf + CPI - 1) {}
751     ~kotlin21() {
752         flush();
753         delete[] buf, delete[] int_buf;
754     }
755     void flush() {
756         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
757         buf_ptr = buf;
758     }
759     void write_(const char &c) {
760         *buf_ptr = c;
761         if (++buf_ptr == buf_end) [[unlikely]]
762             flush();
763     }
764     void write_(const char *s) {
765         for (; *s != '\0'; ++s) write_(*s);
766     }
767     void write(int x) {
768         if (x < 0) write_('-'), x = -x;
769         if (x == 0) [[unlikely]]
770             return write_('0');
771         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
772             *tbuf = '0' + char(x % 10);
773         write_(++tbuf);
774     }
775 };
776
777 struct kotlin22 {
778     static constexpr size_t LEN = 32 << 20;
779     char *buf, *buf_ptr, *buf_end, *tbuf;
780     char *int_buf, *int_buf_end;
781     kotlin22()
782         : buf(new char[LEN]), buf_ptr(buf),
783           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
784           int_buf_end(int_buf + CPI - 1) {}
785     ~kotlin22() {
786         flush();
787         delete[] buf, delete[] int_buf;
788     }
789     void flush() {
790         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
791         buf_ptr = buf;
792     }
793     void write_(const char &c) {
794         *buf_ptr = c;
795         if (++buf_ptr == buf_end) [[unlikely]]
796             flush();
797     }
798     void write_(const char *s) {
799         for (; *s != '\0'; ++s) write_(*s);
800     }
801     void write(int x) {
802         if (x < 0) write_('-'), x = -x;
803         if (x == 0) [[unlikely]]
804             return write_('0');
805         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
806             *tbuf = '0' + char(x % 10);
807         write_(++tbuf);
808     }
809 };
810
811 struct kotlin23 {
812     static constexpr size_t LEN = 32 << 20;
813     char *buf, *buf_ptr, *buf_end, *tbuf;
814     char *int_buf, *int_buf_end;
815     kotlin23()
816         : buf(new char[LEN]), buf_ptr(buf),
817           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
818           int_buf_end(int_buf + CPI - 1) {}
819     ~kotlin23() {
820         flush();
821         delete[] buf, delete[] int_buf;
822     }
823     void flush() {
824         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
825         buf_ptr = buf;
826     }
827     void write_(const char &c) {
828         *buf_ptr = c;
829         if (++buf_ptr == buf_end) [[unlikely]]
830             flush();
831     }
832     void write_(const char *s) {
833         for (; *s != '\0'; ++s) write_(*s);
834     }
835     void write(int x) {
836         if (x < 0) write_('-'), x = -x;
837         if (x == 0) [[unlikely]]
838             return write_('0');
839         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
840             *tbuf = '0' + char(x % 10);
841         write_(++tbuf);
842     }
843 };
844
845 struct kotlin24 {
846     static constexpr size_t LEN = 32 << 20;
847     char *buf, *buf_ptr, *buf_end, *tbuf;
848     char *int_buf, *int_buf_end;
849     kotlin24()
850         : buf(new char[LEN]), buf_ptr(buf),
851           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
852           int_buf_end(int_buf + CPI - 1) {}
853     ~kotlin24() {
854         flush();
855         delete[] buf, delete[] int_buf;
856     }
857     void flush() {
858         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
859         buf_ptr = buf;
860     }
861     void write_(const char &c) {
862         *buf_ptr = c;
863         if (++buf_ptr == buf_end) [[unlikely]]
864             flush();
865     }
866     void write_(const char *s) {
867         for (; *s != '\0'; ++s) write_(*s);
868     }
869     void write(int x) {
870         if (x < 0) write_('-'), x = -x;
871         if (x == 0) [[unlikely]]
872             return write_('0');
873         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
874             *tbuf = '0' + char(x % 10);
875         write_(++tbuf);
876     }
877 };
878
879 struct kotlin25 {
880     static constexpr size_t LEN = 32 << 20;
881     char *buf, *buf_ptr, *buf_end, *tbuf;
882     char *int_buf, *int_buf_end;
883     kotlin25()
884         : buf(new char[LEN]), buf_ptr(buf),
885           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
886           int_buf_end(int_buf + CPI - 1) {}
887     ~kotlin25() {
888         flush();
889         delete[] buf, delete[] int_buf;
890     }
891     void flush() {
892         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
893         buf_ptr = buf;
894     }
895     void write_(const char &c) {
896         *buf_ptr = c;
897         if (++buf_ptr == buf_end) [[unlikely]]
898             flush();
899     }
900     void write_(const char *s) {
901         for (; *s != '\0'; ++s) write_(*s);
902     }
903     void write(int x) {
904         if (x < 0) write_('-'), x = -x;
905         if (x == 0) [[unlikely]]
906             return write_('0');
907         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
908             *tbuf = '0' + char(x % 10);
909         write_(++tbuf);
910     }
911 };
912
913 struct kotlin26 {
914     static constexpr size_t LEN = 32 << 20;
915     char *buf, *buf_ptr, *buf_end, *tbuf;
916     char *int_buf, *int_buf_end;
917     kotlin26()
918         : buf(new char[LEN]), buf_ptr(buf),
919           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
920           int_buf_end(int_buf + CPI - 1) {}
921     ~kotlin26() {
922         flush();
923         delete[] buf, delete[] int_buf;
924     }
92
```

```
delete[] buf;
9 }
```

1.3.5 ctypes

```
1 from ctypes import *
3 # computes 10**4300
4 gmp = CDLL('libgmp.so')
5 x = create_string_buffer(b'\x00'*16)
6 gmp.__gmpz_init_set_ui(byref(x), 10)
7 gmp.__gmpz_pow_ui(byref(x), byref(x), 4300)
8 gmp.__gmpz_printf(b'%Zd\n', byref(x))
9 gmp.__gmpz_clear(byref(x))
# objdump -T `whereis libgmp.so`
```

1.4 Algorithms

1.4.1 Bit Hacks

```
1 // next permutation of x as a bit sequence
2 ull next_bits_permutation(ull x) {
3     ull c = __builtin_ctzll(x), r = x + (1ULL << c);
4     return (r ^ x) >> (c + 2) | r;
5 }
// iterate over all (proper) subsets of bitset s
7 void subsets(ull s) {
8     for (ull x = s; x; x -= x & -x) { --x &= s; /* do stuff */ }
9 }
```

1.4.2 Aliens Trick

```
1 // min dp[i] value and its i (smallest one)
2 pll get_dp(int cost);
3 ll aliens(int k, int l, int r) {
4     while (l != r) {
5         int m = (l + r) / 2;
6         auto [f, s] = get_dp(m);
7         if (s == k) return f - m * k;
8         if (s < k) r = m;
9         else l = m + 1;
10    }
11    return get_dp(l).first - l * k;
12 }
```

1.4.3 Hilbert Curve

```
1 ll hilbert(ll n, int x, int y) {
2     ll res = 0;
3     for (ll s = n; s /= 2;) {
4         int rx = !(x & s), ry = !(y & s);
5         res += s * s * ((3 * rx) ^ ry);
6         if (ry == 0) {
7             if (rx == 1) x = s - 1 - x, y = s - 1 - y;
8             swap(x, y);
9         }
10    }
11    return res;
12 }
```

1.4.4 Longest Increasing Subsequence

```
1 template <class I> vi lis(const vector<I> &S) {
2     if (S.empty()) return {};
3     vi prev(sz(S));
4     typedef pair<I, int> p;
5     vector<p> res;
6     rep(i, 0, sz(S)) {
7         // change 0 -> i for longest non-decreasing subsequence
8         auto it = lower_bound(all(res), p{S[i], 0});
9         if (it == res.end())
10            res.emplace_back(), it = res.end() - 1;
11        *it = {S[i], i};
12        prev[i] = it == res.begin() ? 0 : (it - 1)->second;
13    }
14    int L = sz(res), cur = res.back().second;
15    vi ans(L);
16    while (L--) ans[L] = cur, cur = prev[cur];
17    return ans;
18 }
```

1.4.5 Mo's Algorithm on Tree

```
1 void MoAlgoOnTree() {
2     Dfs(0, -1);
3     vector<int> euler(tk);
4     for (int i = 0; i < n; ++i) {
5         euler[tin[i]] = i;
6         euler[tout[i]] = i;
7     }
8     vector<int> l(q), r(q), qr(q), sp(q, -1);
```

```
9     for (int i = 0; i < q; ++i) {
10        if (tin[u[i]] > tin[v[i]]) swap(u[i], v[i]);
11        int z = GetLCA(u[i], v[i]);
12        sp[i] = z[i];
13        if (z == u) l[i] = tin[u[i]], r[i] = tin[v[i]];
14        else l[i] = tout[u[i]], r[i] = tin[v[i]];
15        qr[i] = i;
16    }
17    sort(qr.begin(), qr.end(), [&](int i, int j) {
18        if (l[i] / kB == l[j] / kB) return r[i] < r[j];
19        return l[i] / kB < l[j] / kB;
20    });
21    vector<bool> used(n);
22    // Add(v): add/remove v to/from the path based on used[v]
23    for (int i = 0, tl = 0, tr = -1; i < q; ++i) {
24        while (tl < l[qr[i]]) Add(euler[tl++]);
25        while (tl > l[qr[i]]) Add(euler[--tl]);
26        while (tr > r[qr[i]]) Add(euler[tr--]);
27        while (tr < r[qr[i]]) Add(euler[++tr]);
28        // add/remove LCA(u, v) if necessary
29    }
30 }
```

2 Data Structures

2.1 GNU PBDS

```
1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/priority_queue.hpp>
3 #include <ext/pb_ds/tree_policy.hpp>
4 using namespace __gnu_pbds;
5
6 // most std::map + order_of_key, find_by_order, split, join
7 template <typename T, typename U = null_type>
8 using ordered_map = tree<T, U, std::less<, rb_tree_tag,
9     tree_order_statistics_node_update>;
10 // useful tags: rb_tree_tag, splay_tree_tag
11
12 template <typename T> struct myhash {
13     size_t operator()(T x) const; // splitmix, bswap(x*R), ...
14 };
15 // most of std::unordered_map, but faster (needs good hash)
16 template <typename T, typename U = null_type>
17 using hash_table = gp_hash_table<T, U, myhash<T>>;
18
19 // most std::priority_queue + modify, erase, split, join
20 using heap = priority_queue<int, std::less<>>;
21 // useful tags: pairing_heap_tag, binary_heap_tag,
22 // (rc_)?binomial_heap_tag, thin_heap_tag
```

2.2 Segment Tree (ZKW)

```
1 struct gextree {
2     using T = int;
3     T f(T a, T b) { return a + b; } // any monoid operation
4     static constexpr T ID = 0; // identity element
5     int n;
6     vector<T> v;
7     gextree(int n_) : n(n_), v(2 * n, ID) {}
8     gextree(vector<T> &a) : n(a.size()), v(2 * n, ID) {
9         copy_n(a.begin(), n, v.begin() + n);
10        for (int i = n - 1; i > 0; i--)
11            v[i] = f(v[i * 2], v[i * 2 + 1]);
12    }
13    void update(int i, T x) {
14        for (v[i += n] = x; i /= 2;)
15            v[i] = f(v[i * 2], v[i * 2 + 1]);
16    }
17    T query(int l, int r) {
18        T tl = ID, tr = ID;
19        for (l += n, r += n; l < r; l /= 2, r /= 2) {
20            if (l & 1) tl = f(tl, v[l++]);
21            if (r & 1) tr = f(v[--r], tr);
22        }
23        return f(tl, tr);
24    }
25};
```

2.3 Line Container

```
1 struct Line {
2     mutable ll k, m, p;
3     bool operator<(const Line &o) const { return k < o.k; }
4     bool operator<(ll x) const { return p < x; }
5 };
6 // add: line y=kx+m, query: maximum y of given x
7 struct LineContainer : multiset<Line, less<>> {
8     // (for doubles, use inf = 1/.0, div(a,b) = a/b)
9     static const ll inf = LLONG_MAX;
10    ll div(ll a, ll b) { // floored division
```

```

11     return a / b - ((a ^ b) < 0 && a % b);
12 }
13 bool isect(iterator x, iterator y) {
14     if (y == end()) return x->p = inf, 0;
15     if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
16     else x->p = div(y->m - x->m, x->k - y->k);
17     return x->p >= y->p;
18 }
19 void add(ll k, ll m) {
20     auto z = insert({k, m, 0}), y = z++, x = y;
21     while (isect(y, z)) z = erase(z);
22     if (x != begin() && isect(--x, y))
23         isect(x, y = erase(y));
24     while ((y = x) != begin() && (--x)->p >= y->p)
25         isect(x, erase(y));
26 }
27 ll query(ll x) {
28     assert(!empty());
29     auto l = *lower_bound(x);
30     return l.k * x + l.m;
31 }

```

2.4 Li-Chao Tree

```

1 constexpr ll MAXN = 2e5, INF = 2e18;
2 struct Line {
3     ll m, b;
4     Line() : m(0), b(-INF) {}
5     Line(ll _m, ll _b) : m(_m), b(_b) {}
6     ll operator()(ll x) const { return m * x + b; }
7 };
8 struct Li_Chao {
9     Line a[MAXN * 4];
10    void insert(Line seg, int l, int r, int v = 1) {
11        if (l == r) {
12            if (seg(l) > a[v](l)) a[v] = seg;
13            return;
14        }
15        int mid = (l + r) >> 1;
16        if (a[v].m > seg.m) swap(a[v], seg);
17        if (a[v](mid) < seg(mid)) {
18            swap(a[v], seg);
19            insert(seg, l, mid, v << 1);
20        } else insert(seg, mid + 1, r, v << 1 | 1);
21    }
22    ll query(int x, int l, int r, int v = 1) {
23        if (l == r) return a[v](x);
24        int mid = (l + r) >> 1;
25        if (x <= mid)
26            return max(a[v](x), query(x, l, mid, v << 1));
27        else
28            return max(a[v](x), query(x, mid + 1, r, v << 1 | 1));
29    }
30 }

```

2.5 adamant HLD

```

1 // subtree of v is [in[v], out[v]]
2 // top of heavy path of v is nxt[v]
3 void dfs1(int v) {
4     sz[v] = 1;
5     for (int u : child[v]) {
6         par[v] = u;
7         dfs1(u);
8         sz[v] += sz[u];
9         if (sz[u] > sz[child[v][0]]) { swap(u, child[v][0]); }
10    }
11 }
12 void dfs2(int v) {
13     in[v] = t++;
14     for (int u : child[v]) {
15         nxt[u] = (u == child[v][0] ? nxt[v] : u);
16         dfs2(u);
17     }
18     out[v] = t;
19 }
20 int lca(int a, int b) {
21     for (; b = par[nxt[b]]);
22     if (in[b] < in[a]) swap(a, b);
23     if (in[nxt[b]] <= in[a]) return a;
24 }
25 }

```

2.6 van Emde Boas Tree

```

1 // stores integers in  $[0, 2^B)$ 
2 // find(+) finds first  $\geq i$  (or  $-1/2^B$  if none)
3 // space:  $\sim 2^B$  bits, time:  $2^B$  init/clear, log B operation
4 template <int B, typename ENABLE = void> struct VEBTree {

```

```

5     const static int K = B / 2, R = (B + 1) / 2, M = (1 <<
6     B);
7     const static int S = 1 << K, MASK = (1 << R) - 1;
8     array<VEBTree<R>, S> ch;
9     VEBTree<K> act;
10    int mi, ma;
11    bool empty() const { return ma < mi; }
12    int findNext(int i) const {
13        if (i <= mi) return mi;
14        if (i > ma) return M;
15        int j = i >> R, x = i & MASK;
16        int res = ch[j].findNext(x);
17        if (res <= MASK) return (j << R) + res;
18        j = act.findNext(j + 1);
19        return (j >= S) ? ma : ((j << R) + ch[j].findNext(0));
20    }
21    int findPrev(int i) const {
22        if (i >= ma) return ma;
23        if (i < mi) return -1;
24        int j = i >> R, x = i & MASK;
25        int res = ch[j].findPrev(x);
26        if (res >= 0) return (j << R) + res;
27        j = act.findPrev(j - 1);
28        return (j < 0) ? mi : ((j << R) +
29        ch[j].findPrev(MASK));
30    }
31    void insert(int i) {
32        if (i <= mi) {
33            if (i == mi) return;
34            swap(mi, i);
35            if (i == M) ma = mi; // we were empty
36            if (i >= ma) return; // we had mi == ma
37        } else if (i >= ma) {
38            if (i == ma) return;
39            swap(ma, i);
40            if (i <= mi) return; // we had mi == ma
41        }
42        int j = i >> R;
43        if (ch[j].empty()) act.insert(j);
44        ch[j].insert(i & MASK);
45    }
46    void erase(int i) {
47        if (i <= mi) {
48            if (i < mi) return;
49            i = mi = findNext(mi + 1);
50            if (i >= ma) {
51                if (i > ma) ma = -1; // we had mi == ma
52                return; // after erase we have mi == ma
53            } else if (i >= ma) {
54                if (i > ma) return;
55                i = ma = findPrev(ma - 1);
56                if (i <= mi) return; // after erase we have mi == ma
57            }
58            int j = i >> R;
59            ch[j].erase(i & MASK);
60            if (ch[j].empty()) act.erase(j);
61        }
62    }
63    void clear() {
64        mi = M, ma = -1;
65        act.clear();
66        for (int i = 0; i < S; ++i) ch[i].clear();
67    }
68    template <class T>
69    void init(const T &bts, int shift = 0, int s0 = 0,
70             int s1 = 0) {
71        s0 =
72            -shift + bts.findNext(shift + s0, shift + M - 1 - s1);
73        s1 =
74            M - 1 -
75            (-shift + bts.findPrev(shift + M - 1 - s1, shift +
76            s0));
77        if (s0 + s1 >= M) clear();
78        else {
79            act.clear();
80            mi = s0, ma = M - 1 - s1;
81            ++s0;
82            ++s1;
83            for (int j = 0; j < S; ++j) {
84                ch[j].init(bts, shift + (j << R),
85                    max(0, s0 - (j << R)),
86                    max(0, s1 - ((S - 1 - j) << R)));
87                if (!ch[j].empty()) act.insert(j);
88            }
89        }
90    }
91    template <int B> struct VEBTree<B, enable_if_t<(B <= 6)>> {
92        const static int M = (1 << B);
93        ull act;
94        bool empty() const { return !act; }
95    };

```

```

93 void clear() { act = 0; }
94 int findNext(int i) const {
95     return ((i < M) && (act >> i))
96         ? i + __builtin_ctzll(act >> i)
97         : M;
98 }
99 int findPrev(int i) const {
100     return ((i != -1) && (act << (63 - i)))
101         ? i - __builtin_clzll(act << (63 - i))
102         : -1;
103 }
104 void insert(int i) { act |= 1ull << i; }
105 void erase(int i) { act &= ~(1ull << i); }
106 template <class T>
107 void init(const T &bts, int shift = 0, int s0 = 0,
108           int s1 = 0) {
109     if (s0 + s1 >= M) act = 0;
110     else
111         act = bts.getRange(shift + s0, shift + M - 1 - s1)
112             << s0;
113 }

```

2.7 Wavelet Matrix

```

1 #pragma GCC target("popcnt,bmi2")
2 #include <immintrin.h>
3 // T is unsigned. You might want to compress values first
4 template <typename T> struct wavelet_matrix {
5     static_assert(is_unsigned_v<T>, "only unsigned T");
6     struct bit_vector {
7         static constexpr uint W = 64;
8         uint n, cnt0;
9         vector<ull> bits;
10        vector<uint> sum;
11        bit_vector(uint n_)
12            : n(n_), bits(n / W + 1), sum(n / W + 1) {}
13        void build() {
14            for (uint j = 0; j != n / W; ++j)
15                sum[j + 1] = sum[j] + _mm_popcnt_u64(bits[j]);
16            cnt0 = rank0(n);
17        }
18        void set_bit(uint i) { bits[i / W] |= 1ULL << i % W; }
19        bool operator[](uint i) const {
20            return !!(bits[i / W] & 1ULL << i % W);
21        }
22        uint rank1(uint i) const {
23            return sum[i / W] +
24                _mm_popcnt_u64(_bzh_u64(bits[i / W], i % W));
25        }
26        uint rank0(uint i) const { return i - rank1(i); }
27    };
28    uint n, lg;
29    vector<bit_vector> b;
30    wavelet_matrix(const vector<T> &a) : n(a.size()) {
31        lg =
32            __lg(max(*max_element(a.begin(), a.end()), T(1))) + 1;
33        b.assign(lg, n);
34        vector<T> cur = a, nxt(n);
35        for (int h = lg; h--;) {
36            for (uint i = 0; i < n; ++i)
37                if (cur[i] & (T(1) << h)) b[h].set_bit(i);
38            b[h].build();
39            int il = 0, ir = b[h].cnt0;
40            for (uint i = 0; i < n; ++i)
41                nxt[(b[h][i] ? ir : il)++] = cur[i];
42            swap(cur, nxt);
43        }
44    }
45    T operator[](uint i) const {
46        T res = 0;
47        for (int h = lg; h--;)
48            if (b[h][i])
49                i += b[h].cnt0 - b[h].rank0(i), res |= T(1) << h;
50        else i = b[h].rank0(i);
51        return res;
52    }
53    // query k-th smallest (0-based) in a[l, r)
54    T kth(uint l, uint r, uint k) const {
55        T res = 0;
56        for (int h = lg; h--;) {
57            uint tl = b[h].rank0(l), tr = b[h].rank0(r);
58            if (k >= tr - tl) {
59                k -= tr - tl;
60                l += b[h].cnt0 - tl;
61                r += b[h].cnt0 - tr;
62                res |= T(1) << h;
63            } else l = tl, r = tr;
64        }
65        return res;

```

```

66    }
67    // count of i in [l, r) with a[i] < u
68    uint count(uint l, uint r, T u) const {
69        if (u >= T(1) << lg) return r - l;
70        uint res = 0;
71        for (int h = lg; h--;) {
72            uint tl = b[h].rank0(l), tr = b[h].rank0(r);
73            if (u & (T(1) << h)) {
74                l += b[h].cnt0 - tl;
75                r += b[h].cnt0 - tr;
76                res += tr - tl;
77            } else l = tl, r = tr;
78        }
79        return res;
80    }
81 }

```

2.8 Link-Cut Tree

```

1 const int MXN = 100005;
2 const int MEM = 100005;
3
4 struct Splay {
5     static Splay nil, mem[MEM], *pmem;
6     Splay *ch[2], *f;
7     int val, rev, size;
8     Splay() : val(-1), rev(0), size(0) {
9         f = ch[0] = ch[1] = &nil;
10    }
11    Splay(int val) : val(val), rev(0), size(1) {
12        f = ch[0] = ch[1] = &nil;
13    }
14    bool isr() {
15        return f->ch[0] != this && f->ch[1] != this;
16    }
17    int dir() { return f->ch[0] == this ? 0 : 1; }
18    void setCh(Splay *c, int d) {
19        ch[d] = c;
20        if (c != &nil) c->f = this;
21        pull();
22    }
23    void push() {
24        if (rev) {
25            swap(ch[0], ch[1]);
26            if (ch[0] != &nil) ch[0]->rev ^= 1;
27            if (ch[1] != &nil) ch[1]->rev ^= 1;
28            rev = 0;
29        }
30    }
31    void pull() {
32        size = ch[0]->size + ch[1]->size + 1;
33        if (ch[0] != &nil) ch[0]->f = this;
34        if (ch[1] != &nil) ch[1]->f = this;
35    }
36    } Splay::nil, Splay::mem[MEM], *Splay::pmem = Splay::mem;
37    Splay *nil = &Splay::nil;
38
39    void rotate(Splay *x) {
40        Splay *p = x->f;
41        int d = x->dir();
42        if (!p->isr()) p->f->setCh(x, p->dir());
43        else x->f = p->f;
44        p->setCh(x->ch[!d], d);
45        x->setCh(p, !d);
46        p->pull();
47        x->pull();
48    }
49
50    vector<Splay*> splayVec;
51    void splay(Splay *x) {
52        splayVec.clear();
53        for (Splay *q = x; q; q = q->f) {
54            splayVec.push_back(q);
55            if (q->isr()) break;
56        }
57        reverse(begin(splayVec), end(splayVec));
58        for (auto it : splayVec) it->push();
59        while (!x->isr()) {
60            if (x->f->isr()) rotate(x);
61            else if (x->dir() == x->f->dir())
62                rotate(x->f), rotate(x);
63            else rotate(x), rotate(x);
64        }
65    }
66
67    Splay *access(Splay *x) {
68        Splay *q = nil;
69        for (; x != nil; x = x->f) {
70            splay(x);
71            x->setCh(q, 1);
72            q = x;

```



```

73 }
74 return q;
75 }
76 void evert(Splay *x) {
77     access(x);
78     splay(x);
79     x->rev ^= 1;
80     x->push();
81     x->pull();
82 }
83 void link(Splay *x, Splay *y) {
84     // evert(x);
85     access(x);
86     splay(x);
87     evert(y);
88     x->setCh(y, 1);
89 }
90 void cut(Splay *x, Splay *y) {
91     // evert(x);
92     access(y);
93     splay(y);
94     y->push();
95     y->ch[0] = y->ch[0]->f = nil;
96 }
97
98 int N, Q;
99 Splay *vt[MXN];
100
101 int ask(Splay *x, Splay *y) {
102     access(x);
103     access(y);
104     splay(x);
105     int res = x->f->val;
106     if (res == -1) res = x->val;
107     return res;
108 }
109
110 int main(int argc, char **argv) {
111     scanf("%d%d", &N, &Q);
112     for (int i = 1; i <= N; i++)
113         vt[i] = new (Splay::pmem++) Splay(i);
114     while (Q--) {
115         char cmd[105];
116         int u, v;
117         scanf("%s", cmd);
118         if (cmd[1] == 'i') {
119             scanf("%d%d", &u, &v);
120             link(vt[u], vt[v]);
121         } else if (cmd[0] == 'c') {
122             scanf("%d", &v);
123             cut(vt[1], vt[v]);
124         } else {
125             scanf("%d%d", &u, &v);
126             int res = ask(vt[u], vt[v]);
127             printf("%d\n", res);
128         }
129     }
130 }

```

3 Graph

3.1 Modeling

- Maximum/Minimum flow with lower bound / Circulation problem
 - Construct super source S and sink T .
 - For each edge (x, y, l, u) , connect $x \rightarrow y$ with capacity $u - l$.
 - For each vertex v , denote by $\text{in}(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
 - If $\text{in}(v) > 0$, connect $S \rightarrow v$ with capacity $\text{in}(v)$, otherwise, connect $v \rightarrow T$ with capacity $-\text{in}(v)$.
 - To maximize, connect $t \rightarrow s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T . If $f \neq \sum_{v \in V, \text{in}(v) > 0} \text{in}(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
 - To minimize, let f be the maximum flow from S to T . Connect $t \rightarrow s$ with capacity ∞ and let the flow from S to T be f' . If $f + f' \neq \sum_{v \in V, \text{in}(v) > 0} \text{in}(v)$, there's no solution. Otherwise, f' is the answer.

- The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.
- Construct minimum vertex cover from maximum matching M on bipartite graph (X, Y)
 - Redirect every edge: $y \rightarrow x$ if $(x, y) \in M$, $x \rightarrow y$ otherwise.
 - DFS from unmatched vertices in X .
 - $x \in X$ is chosen iff x is unvisited.
 - $y \in Y$ is chosen iff y is visited.
- Minimum cost cyclic flow
 - Construct super source S and sink T
 - For each edge (x, y, c) , connect $x \rightarrow y$ with $(\text{cost}, \text{cap}) = (c, 1)$ if $c > 0$, otherwise connect $y \rightarrow x$ with $(\text{cost}, \text{cap}) = (-c, 1)$
 - For each edge with $c < 0$, sum these cost as K , then increase $d(y)$ by 1, decrease $d(x)$ by 1
 - For each vertex v with $d(v) > 0$, connect $S \rightarrow v$ with $(\text{cost}, \text{cap}) = (0, d(v))$
 - For each vertex v with $d(v) < 0$, connect $v \rightarrow T$ with $(\text{cost}, \text{cap}) = (0, -d(v))$
 - Flow from S to T , the answer is the cost of the flow $C + K$
- Maximum density induced subgraph
 - Binary search on answer, suppose we're checking answer T
 - Construct a max flow model, let K be the sum of all weights
 - Connect source $s \rightarrow v, v \in G$ with capacity K
 - For each edge (u, v, w) in G , connect $u \rightarrow v$ and $v \rightarrow u$ with capacity w
 - For $v \in G$, connect it with sink $v \rightarrow t$ with capacity $K + 2T - \left(\sum_{e \in E(v)} w(e) \right) - 2w(v)$
 - T is a valid answer if the maximum flow $f < K|V|$
- Minimum weight edge cover
 - For each $v \in V$ create a copy v' , and connect $u' \rightarrow v'$ with weight $w(u, v)$.
 - Connect $v \rightarrow v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v .
 - Find the minimum weight perfect matching on G' .
- Project selection problem
 - If $p_v > 0$, create edge (s, v) with capacity p_v ; otherwise, create edge (v, t) with capacity $-p_v$.
 - Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v .
 - The mincut is equivalent to the maximum profit of a subset of projects.
- 0/1 quadratic programming

$$\sum_x c_x x + \sum_y c_y \bar{y} + \sum_{xy} c_{xy} x \bar{y} + \sum_{xyx'y'} c_{xyx'y'} (x \bar{y} + x' \bar{y}')$$

can be minimized by the mincut of the following graph:

- Create edge (x, t) with capacity c_x and create edge (s, y) with capacity c_y .
- Create edge (x, y) with capacity c_{xy} .
- Create edge (x, y) and edge (x', y') with capacity $c_{xyx'y'}$.

3.2 Matching/Flows

3.2.1 Dinic's Algorithm

```

1 struct Dinic {
2     struct edge {
3         int to, cap, flow, rev;
4     };
5     static constexpr int MAXN = 1000, MAXF = 1e9;
6     vector<edge> v[MAXN];
7     int top[MAXN], deep[MAXN], side[MAXN], s, t;
8     void make_edge(int s, int t, int cap) {
9         v[s].push_back({t, cap, 0, (int)v[t].size()});
10        v[t].push_back({s, 0, 0, (int)v[s].size() - 1});
11    }
12    int dfs(int a, int flow) {
13        if (a == t || !flow) return flow;
14        for (int &i = top[a]; i < v[a].size(); i++) {
15            edge &e = v[a][i];
16            if (deep[a] + 1 == deep[e.to] && e.cap - e.flow) {

```

```

17     int x = dfs(e.to, min(e.cap - e.flow, flow));
18     if (x) {
19         e.flow += x, v[e.to][e.rev].flow -= x;
20         return x;
21     }
22 }
23 deep[a] = -1;
24 return 0;
25 }
26 bool bfs() {
27     queue<int> q;
28     fill_n(deep, MAXN, 0);
29     q.push(s), deep[s] = 1;
30     int tmp;
31     while (!q.empty()) {
32         tmp = q.front(), q.pop();
33         for (edge e : v[tmp])
34             if (!deep[e.to] && e.cap != e.flow)
35                 deep[e.to] = deep[tmp] + 1, q.push(e.to);
36     }
37     return deep[t];
38 }
39 int max_flow(int _s, int _t) {
40     s = _s, t = _t;
41     int flow = 0, tflow;
42     while (bfs()) {
43         fill_n(top, MAXN, 0);
44         while ((tflow = dfs(s, MAXF))) flow += tflow;
45     }
46     return flow;
47 }
48 void reset() {
49     fill_n(side, MAXN, 0);
50     for (auto &i : v) i.clear();
51 }
52 }
53 };

```

3.2.2 Minimum Cost Flow

```

1 struct MCF {
2     struct edge {
3         ll to, from, cap, flow, cost, rev;
4     } *fromE[MAXN];
5     vector<edge> v[MAXN];
6     ll n, s, t, flows[MAXN], dis[MAXN], pi[MAXN], flowlim;
7     void make_edge(int s, int t, ll cap, ll cost) {
8         if (!cap) return;
9         v[s].pb(edge{t, s, cap, 0LL, cost, v[t].size()});
10        v[t].pb(edge{s, t, 0LL, 0LL, -cost, v[s].size() - 1});
11    }
12    bitset<MAXN> vis;
13    void dijkstra() {
14        vis.reset();
15        __gnu_pbds::priority_queue<pair<ll, int>> q;
16        vector<decltype(q)::point_iterator> its(n);
17        q.push({0LL, s});
18        while (!q.empty()) {
19            int now = q.top().second;
20            q.pop();
21            if (vis[now]) continue;
22            vis[now] = 1;
23            ll ndis = dis[now] + pi[now];
24            for (edge &e : v[now]) {
25                if (e.flow == e.cap || vis[e.to]) continue;
26                if (dis[e.to] > ndis + e.cost - pi[e.to]) {
27                    dis[e.to] = ndis + e.cost - pi[e.to];
28                    flows[e.to] = min(flows[now], e.cap - e.flow);
29                    fromE[e.to] = &e;
30                    if (its[e.to] == q.end())
31                        its[e.to] = q.push({-dis[e.to], e.to});
32                    else q.modify(its[e.to], {-dis[e.to], e.to});
33                }
34            }
35        }
36    }
37    bool AP(ll &flow) {
38        fill_n(dis, n, INF);
39        fromE[s] = 0;
40        dis[s] = 0;
41        flows[s] = flowlim - flow;
42        dijkstra();
43        if (dis[t] == INF) return false;
44        flow += flows[t];
45        for (edge *e = fromE[t]; e; e = fromE[e->from]) {
46            e->flow += flows[t];
47            v[e->to][e->rev].flow -= flows[t];
48        }
49        for (int i = 0; i < n; i++)
50            pi[i] = min(pi[i] + dis[i], INF);
51        return true;
52    }
53 };

```

```

53 }
54 pll solve(int _s, int _t, ll_flowlim = INF) {
55     s = _s, t = _t, flowlim = _flowlim;
56     pll re;
57     while (re.F != flowlim && AP(re.F))
58         ;
59     for (int i = 0; i < n; i++)
60         for (edge &e : v[i])
61             if (e.flow != 0) re.S += e.flow * e.cost;
62     re.S /= 2;
63     return re;
64 }
65 void init(int _n) {
66     n = _n;
67     fill_n(pi, n, 0);
68     for (int i = 0; i < n; i++) v[i].clear();
69 }
70 void setpi(int s) {
71     fill_n(pi, n, INF);
72     pi[s] = 0;
73     for (ll it = 0, flag = 1, tdis; flag && it < n; it++) {
74         flag = 0;
75         for (int i = 0; i < n; i++)
76             if (pi[i] != INF)
77                 for (edge &e : v[i])
78                     if (e.cap && (tdis = pi[i] + e.cost) <
79                         pi[e.to])
80                         pi[e.to] = tdis, flag = 1;
81     }
82 }

```

3.2.3 Gomory-Hu Tree

```

1 int e[MAXN][MAXN];
2 int p[MAXN];
3 Dinic D; // original graph
4 void gomory_hu() {
5     fill(p, p + n, 0);
6     fill(e[0], e[n], INF);
7     for (int s = 1; s < n; s++) {
8         int t = p[s];
9         Dinic F = D;
10        int tmp = F.max_flow(s, t);
11        for (int i = 1; i < s; i++)
12            e[s][i] = e[i][s] = min(tmp, e[t][i]);
13        for (int i = s + 1; i <= n; i++)
14            if (p[i] == t && F.side[i]) p[i] = s;
15    }
16 }

```

3.2.4 Global Minimum Cut

```

1 // weights is an adjacency matrix, undirected
2 pair<int, vi> getMinCut(vector<vi> &weights) {
3     int N = sz(weights);
4     vi used(N), cut, best_cut;
5     int best_weight = -1;
6
7     for (int phase = N - 1; phase >= 0; phase--) {
8         vi w = weights[0], added = used;
9         int prev, k = 0;
10        rep(i, 0, phase) {
11            prev = k;
12            k = -1;
13            rep(j, 1, N) if (!added[j] &&
14                (k == -1 || w[j] > w[k])) k = j;
15            if (i == phase - 1) {
16                rep(j, 0, N) weights[prev][j] += weights[k][j];
17                rep(j, 0, N) weights[j][prev] = weights[prev][j];
18                used[k] = true;
19                cut.push_back(k);
20                if (best_weight == -1 || w[k] < best_weight) {
21                    best_cut = cut;
22                    best_weight = w[k];
23                }
24            } else {
25                rep(j, 0, N) w[j] += weights[k][j];
26                added[k] = true;
27            }
28        }
29    }
30    return {best_weight, best_cut};
31 }

```

3.2.5 Bipartite Minimum Cover

```

1 // maximum independent set = all vertices not covered
2 // x : [0, n), y : [0, m]
3 struct Bipartite_vertex_cover {
4     Dinic D;
5 };

```

```

5  int n, m, s, t, x[maxn], y[maxn];
   void make_edge(int x, int y) { D.make_edge(x, y + n,
1); }
7  int matching() {
   int re = D.max_flow(s, t);
9   for (int i = 0; i < n; i++)
       for (Dinic::edge &e : D.v[i])
11      if (e.to != s && e.flow == 1) {
          x[i] = e.to - n, y[e.to - n] = i;
13      break;
       }
15   return re;
   }
17 // init() and matching() before use
   void solve(vector<int> &vx, vector<int> &vy) {
19     bitset<maxn * 2 + 10> vis;
       queue<int> q;
21     for (int i = 0; i < n; i++)
         if (x[i] == -1) q.push(i), vis[i] = 1;
23     while (!q.empty()) {
         int now = q.front();
25         q.pop();
         if (now < n) {
27             for (Dinic::edge &e : D.v[now])
                 if (e.to != s && e.to - n != x[now] && !
vis[e.to])
29                 vis[e.to] = 1, q.push(e.to);
             } else {
31                 if (!vis[y[now - n]])
                     vis[y[now - n]] = 1, q.push(y[now - n]);
33             }
35         for (int i = 0; i < n; i++)
             if (!vis[i]) vx.pb(i);
37         for (int i = 0; i < m; i++)
             if (vis[i + n]) vy.pb(i);
39     }
   void init(int _n, int _m) {
41     n = _n, m = _m, s = n + m, t = s + 1;
       for (int i = 0; i < n; i++)
43         x[i] = -1, D.make_edge(s, i, 1);
       for (int i = 0; i < m; i++)
45         y[i] = -1, D.make_edge(i + n, t, 1);
   }
47 };

```

3.2.6 Edmonds' Algorithm

```

1 struct Edmonds {
   int n, T;
3   vector<vector<int>> g;
   vector<int> pa, p, used, base;
5   Edmonds(int n)
       : n(n), T(0), g(n), pa(n, -1), p(n), used(n),
         base(n) {}
7   void add(int a, int b) {
       g[a].push_back(b);
       g[b].push_back(a);
9   }
11  int getBase(int i) {
       while (i != base[i])
13     base[i] = base[base[i]], i = base[i];
       return i;
15  }
17  vector<int> toJoin;
   void mark_path(int v, int x, int b, vector<int> &path) {
19     for (; getBase(v) != b; v = p[x]) {
         p[v] = x, x = pa[v];
21     toJoin.push_back(v);
         toJoin.push_back(x);
23     if (!used[x]) used[x] = ++T, path.push_back(x);
   }
25  }
   bool go(int v) {
27     for (int x : g[v]) {
         int b, bv = getBase(v), bx = getBase(x);
29     if (bv == bx) {
         continue;
31     } else if (used[x]) {
         vector<int> path;
33     toJoin.clear();
         if (used[bx] < used[bv])
35     mark_path(v, x, b = bx, path);
         else mark_path(x, v, b = bv, path);
37     for (int z : toJoin) base[getBase(z)] = b;
         for (int z : path)
39     if (go(z)) return 1;
     } else if (p[x] == -1) {
41     p[x] = v;
         if (pa[x] == -1) {
43     for (int y; x != -1; x = v)

```

```

       y = p[x], v = pa[y], pa[x] = y, pa[y] = x;
45     return 1;
       }
47     if (!used[pa[x]]) {
         used[pa[x]] = ++T;
49     if (go(pa[x])) return 1;
       }
51   }
53   return 0;
   }
55  void init_dfs() {
       for (int i = 0; i < n; i++)
57     used[i] = 0, p[i] = -1, base[i] = i;
   }
59  bool dfs(int root) {
       used[root] = ++T;
61     return go(root);
   }
63  void match() {
       int ans = 0;
65     for (int v = 0; v < n; v++)
         for (int x : g[v])
67         if (pa[v] == -1 && pa[x] == -1) {
             pa[v] = x, pa[x] = v, ans++;
69         break;
       }
71     init_dfs();
       for (int i = 0; i < n; i++)
73     if (pa[i] == -1 && dfs(i)) ans++, init_dfs();
       cout << ans * 2 << "\n";
75     for (int i = 0; i < n; i++)
         if (pa[i] > i)
77     cout << i + 1 << " " << pa[i] + 1 << "\n";
79 };

```

3.2.7 Minimum Weight Matching

```

1 struct Graph {
   static const int MAXN = 105;
3   int n, e[MAXN][MAXN];
   int match[MAXN], d[MAXN], onstk[MAXN];
5   vector<int> stk;
   void init(int _n) {
7     n = _n;
       for (int i = 0; i < n; i++)
9     for (int j = 0; j < n; j++)
         // change to appropriate infinity
         // if not complete graph
11     e[i][j] = 0;
13   }
15  void add_edge(int u, int v, int w) {
       e[u][v] = e[v][u] = w;
17  }
19  bool SPFA(int u) {
       if (onstk[u]) return true;
       stk.push_back(u);
       onstk[u] = 1;
21     for (int v = 0; v < n; v++) {
         if (u != v && match[u] != v && !onstk[v]) {
23         int m = match[v];
             if (d[m] > d[u] - e[v][m] + e[u][v]) {
25         d[m] = d[u] - e[v][m] + e[u][v];
                 onstk[v] = 1;
                 stk.push_back(v);
                 if (SPFA(m)) return true;
29         stk.pop_back();
                 onstk[v] = 0;
31     }
       }
33   }
35   onstk[u] = 0;
       stk.pop_back();
       return false;
37  }
39  int solve() {
       for (int i = 0; i < n; i += 2) {
41     match[i] = i + 1;
         match[i + 1] = i;
43   }
       while (true) {
45     int found = 0;
         for (int i = 0; i < n; i++) onstk[i] = d[i] = 0;
         for (int i = 0; i < n; i++) {
47     stk.clear();
             if (!onstk[i] && SPFA(i)) {
49     found = 1;
                 while (stk.size() >= 2) {
51     int u = stk.back();
                     stk.pop_back();

```



```

53         int v = stk.back();
54         stk.pop_back();
55         match[u] = v;
56         match[v] = u;
57     }
58 }
59 if (!found) break;
60 }
61 int ret = 0;
62 for (int i = 0; i < n; i++) ret += e[i][match[i]];
63 ret /= 2;
64 return ret;
65 }
66 } graph;

```

3.2.8 Stable Marriage

```

1 // normal stable marriage problem
2 /* input:
3 3
4 Albert Laura Nancy Marcy
5 Brad Marcy Nancy Laura
6 Chuck Laura Marcy Nancy
7 Laura Chuck Albert Brad
8 Marcy Albert Chuck Brad
9 Nancy Brad Albert Chuck
10 */
11
12 using namespace std;
13 const int MAXN = 505;
14
15 int n;
16 int favor[MAXN][MAXN]; // favor[boy_id][rank] = girl_id;
17 int order[MAXN][MAXN]; // order[girl_id][boy_id] = rank;
18 int current[MAXN]; // current[boy_id] = rank;
19 // boy_id will pursue current[boy_id] girl.
20 int girl_current[MAXN]; // girl[girl_id] = boy_id;
21
22 void initialize() {
23     for (int i = 0; i < n; i++) {
24         current[i] = 0;
25         girl_current[i] = n;
26         order[i][n] = n;
27     }
28 }
29
30 map<string, int> male, female;
31 string bname[MAXN], gname[MAXN];
32 int fit = 0;
33
34 void stable_marriage() {
35     queue<int> que;
36     for (int i = 0; i < n; i++) que.push(i);
37     while (!que.empty()) {
38         int boy_id = que.front();
39         que.pop();
40
41         int girl_id = favor[boy_id][current[boy_id]];
42         current[boy_id]++;
43
44         if (order[girl_id][boy_id] <
45             order[girl_id][girl_current[girl_id]]) {
46             if (girl_current[girl_id] < n)
47                 que.push(girl_current[girl_id]);
48             girl_current[girl_id] = boy_id;
49         } else {
50             que.push(boy_id);
51         }
52     }
53 }
54
55 int main() {
56     cin >> n;
57
58     for (int i = 0; i < n; i++) {
59         string p, t;
60         cin >> p;
61         male[p] = i;
62         bname[i] = p;
63         for (int j = 0; j < n; j++) {
64             cin >> t;
65             if (!female.count(t)) {
66                 gname[fit] = t;
67                 female[t] = fit++;
68             }
69             favor[i][j] = female[t];
70         }
71     }
72 }

```

```

75 for (int i = 0; i < n; i++) {
76     string p, t;
77     cin >> p;
78     for (int j = 0; j < n; j++) {
79         cin >> t;
80         order[female[p]][male[t]] = j;
81     }
82 }
83 initialize();
84 stable_marriage();
85
86 for (int i = 0; i < n; i++) {
87     cout << bname[i] << " "
88         << gname[favor[i][current[i] - 1]] << endl;
89 }
90 }
91 }

```

3.2.9 Kuhn-Munkres algorithm

```

1 // Maximum Weight Perfect Bipartite Matching
2 // Detect non-perfect-matching:
3 // 1. set all edge[i][j] as INF
4 // 2. if solve() >= INF, it is not perfect matching.
5
6 typedef long long ll;
7 struct KM {
8     static const int MAXN = 1050;
9     static const ll INF = 1LL << 60;
10     int n, match[MAXN], vx[MAXN], vy[MAXN];
11     ll edge[MAXN][MAXN], lx[MAXN], ly[MAXN], slack[MAXN];
12     void init(int _n) {
13         n = _n;
14         for (int i = 0; i < n; i++)
15             for (int j = 0; j < n; j++) edge[i][j] = 0;
16     }
17     void add_edge(int x, int y, ll w) { edge[x][y] = w; }
18     bool DFS(int x) {
19         vx[x] = 1;
20         for (int y = 0; y < n; y++) {
21             if (vy[y] continue;
22             if (lx[x] + ly[y] > edge[x][y]) {
23                 slack[y] =
24                     min(slack[y], lx[x] + ly[y] - edge[x][y]);
25             } else {
26                 vy[y] = 1;
27                 if (match[y] == -1 || DFS(match[y])) {
28                     match[y] = x;
29                     return true;
30                 }
31             }
32         }
33         return false;
34     }
35     ll solve() {
36         fill(match, match + n, -1);
37         fill(lx, lx + n, -INF);
38         fill(ly, ly + n, 0);
39         for (int i = 0; i < n; i++)
40             for (int j = 0; j < n; j++)
41                 lx[i] = max(lx[i], edge[i][j]);
42         for (int i = 0; i < n; i++) {
43             fill(slack, slack + n, INF);
44             while (true) {
45                 fill(vx, vx + n, 0);
46                 fill(vy, vy + n, 0);
47                 if (DFS(i)) break;
48                 ll d = INF;
49                 for (int j = 0; j < n; j++)
50                     if (!vy[j]) d = min(d, slack[j]);
51                 for (int j = 0; j < n; j++) {
52                     if (vx[j]) lx[j] -= d;
53                     if (vy[j]) ly[j] += d;
54                     else slack[j] -= d;
55                 }
56             }
57         }
58         ll res = 0;
59         for (int i = 0; i < n; i++) {
60             res += edge[match[i]][i];
61         }
62         return res;
63     }
64 } graph;

```

3.3 Shortest Path Faster Algorithm

```

1 struct SPFA {
2     static const int maxn = 1010, INF = 1e9;
3     int dis[maxn];

```

```

bitset<maxn> inq, inneg;
5 queue<int> q, tq;
vector<pii> v[maxn];
7 void make_edge(int s, int t, int w) {
    v[s].emplace_back(t, w);
9 }
void dfs(int a) {
    inneg[a] = 1;
    for (pii i : v[a])
        if (!inneg[i.F]) dfs(i.F);
13 }
bool solve(int n, int s) { // true if have neg-cycle
    for (int i = 0; i <= n; i++) dis[i] = INF;
    dis[s] = 0, q.push(s);
    for (int i = 0; i < n; i++) {
        inq.reset();
        int now;
        while (!q.empty()) {
            now = q.front(), q.pop();
            for (pii &i : v[now]) {
                if (dis[i.F] > dis[now] + i.S) {
                    dis[i.F] = dis[now] + i.S;
                    if (!inq[i.F]) tq.push(i.F), inq[i.F] = 1;
                }
            }
            q.swap(tq);
        }
        bool re = !q.empty();
        inneg.reset();
        while (!q.empty()) {
            if (!inneg[q.front()]) dfs(q.front());
            q.pop();
        }
        return re;
    }
    void reset(int n) {
        for (int i = 0; i <= n; i++) v[i].clear();
    }
43 };

```

3.4 Strongly Connected Components

```

1 struct TarjanScc {
    int n, step;
    vector<int> time, low, instk, stk;
    vector<vector<int>> e, scc;
5 TarjanScc(int n_)
    : n(n_), step(0), time(n), low(n), instk(n), e(n) {}
7 void add_edge(int u, int v) { e[u].push_back(v); }
void dfs(int x) {
    time[x] = low[x] = ++step;
    stk.push_back(x);
    instk[x] = 1;
    for (int y : e[x])
        if (!time[y]) {
            dfs(y);
            low[x] = min(low[x], low[y]);
        } else if (instk[y]) {
            low[x] = min(low[x], time[y]);
        }
19 if (time[x] == low[x]) {
    scc.emplace_back();
    for (int y = -1; y != x; ) {
        y = stk.back();
        stk.pop_back();
        instk[y] = 0;
        scc.back().push_back(y);
    }
27 }
29 void solve() {
    for (int i = 0; i < n; i++)
        if (!time[i]) dfs(i);
    reverse(scc.begin(), scc.end());
33 // scc in topological order
}
35 };

```

3.4.1 2-Satisfiability

```

1 // 1 based, vertex in SCC = MAXN * 2
// (not i) is i + n
3 struct two_SAT {
    int n, ans[MAXN];
5 SCC S;
void imply(int a, int b) { S.make_edge(a, b); }
7 bool solve(int _n) {
    n = _n;
    S.solve(n * 2);
    for (int i = 1; i <= n; i++) {

```

```

11 if (S.scc[i] == S.scc[i + n]) return false;
    ans[i] = (S.scc[i] < S.scc[i + n]);
13 }
    return true;
15 }
void init(int _n) {
    n = _n;
    fill_n(ans, n + 1, 0);
    S.init(n * 2);
    }
21 } SAT;

```

3.5 Biconnected Components

3.5.1 Articulation Points

```

1 void dfs(int x, int p) {
    tin[x] = low[x] = ++t;
    int ch = 0;
    for (auto u : g[x])
        if (u.first != p) {
            if (!ins[u.second])
                st.push(u.second), ins[u.second] = true;
            if (tin[u.first]) {
                low[x] = min(low[x], tin[u.first]);
                continue;
            }
            ++ch;
            dfs(u.first, x);
            low[x] = min(low[x], low[u.first]);
            if (low[u.first] >= tin[x]) {
                cut[x] = true;
                ++sz;
                while (true) {
                    int e = st.top();
                    st.pop();
                    bcc[e] = sz;
                    if (e == u.second) break;
                }
            }
        }
    if (ch == 1 && p == -1) cut[x] = false;
27 }

```

3.5.2 Bridges

```

1 // if there are multi-edges, then they are not bridges
void dfs(int x, int p) {
    tin[x] = low[x] = ++t;
    st.push(x);
    for (auto u : g[x])
        if (u.first != p) {
            if (tin[u.first]) {
                low[x] = min(low[x], tin[u.first]);
                continue;
            }
            dfs(u.first, x);
            low[x] = min(low[x], low[u.first]);
            if (low[u.first] == tin[u.first]) br[u.second] =
true;
15 if (tin[x] == low[x]) {
    ++sz;
    while (st.size()) {
        int u = st.top();
        st.pop();
        bcc[u] = sz;
        if (u == x) break;
    }
23 }
}

```

3.6 Triconnected Components

```

1 // requires a union-find data structure
struct ThreeEdgeCC {
    int V, ind;
    vector<int> id, pre, post, low, deg, path;
    vector<vector<int>> components;
    UnionFind uf;
7 template <class Graph>
void dfs(const Graph &G, int v, int prev) {
    pre[v] = ++ind;
    for (int w : G[v])
        if (w != v) {
            if (w == prev) {
                prev = -1;
                continue;
            }
            if (pre[w] != -1) {
                if (pre[w] < pre[v]) {

```

```

19     deg[v]++;
    low[v] = min(low[v], pre[w]);
    } else {
21     deg[v]--;
    int &u = path[v];
23     for (; u != -1 && pre[u] <= pre[w] &&
        pre[w] <= post[u];) {
25         uf.join(v, u);
        deg[v] += deg[u];
27         u = path[u];
    }
29     }
    continue;
31 }
dfs(G, w, v);
33 if (path[w] == -1 && deg[w] <= 1) {
    deg[v] += deg[w];
35     low[v] = min(low[v], low[w]);
    continue;
37 }
if (deg[w] == 0) w = path[w];
39 if (low[v] > low[w]) {
    low[v] = min(low[v], low[w]);
41     swap(w, path[v]);
    }
43 for (; w != -1; w = path[w]) {
    uf.join(v, w);
45     deg[v] += deg[w];
    }
47 }
post[v] = ind;
49 }
template <class Graph>
51 ThreeEdgeCC(const Graph &G)
    : V(G.size()), ind(-1), id(V, -1), pre(V, -1),
53     post(V), low(V, INT_MAX), deg(V, 0), path(V, -1),
    uf(V) {
55     for (int v = 0; v < V; v++)
        if (pre[v] == -1) dfs(G, v, -1);
57     components.reserve(uf.cnt);
    for (int v = 0; v < V; v++)
59         if (uf.find(v) == v) {
            id[v] = components.size();
61             components.emplace_back(1, v);
            components.back().reserve(uf.getSize(v));
63         }
    for (int v = 0; v < V; v++)
65         if (id[v] == -1)
            components[id[v] = id[uf.find(v)]].push_back(v);
67 }
};

```

3.7 Centroid Decomposition

```

1 void get_center(int now) {
    v[now] = true;
3     vtx.push_back(now);
    sz[now] = 1;
5     mx[now] = 0;
    for (int u : G[now])
7         if (!v[u]) {
            get_center(u);
9             mx[now] = max(mx[now], sz[u]);
            sz[now] += sz[u];
11        }
    }
13 void get_dis(int now, int d, int len) {
    dis[d][now] = cnt;
15     v[now] = true;
    for (auto u : G[now])
17         if (!v[u.first]) { get_dis(u, d, len + u.second); }
    }
19 void dfs(int now, int fa, int d) {
    get_center(now);
21     int c = -1;
    for (int i : vtx) {
23         if (max(mx[i], (int)vtx.size() - sz[i]) <=
            (int)vtx.size() / 2)
25             c = i;
        v[i] = false;
27    }
    get_dis(c, d, 0);
29    for (int i : vtx) v[i] = false;
    v[c] = true;
31    vtx.clear();
    dep[c] = d;
33    p[c] = fa;
    for (auto u : G[c])
35        if (u.first != fa && !v[u.first]) {
            dfs(u.first, c, d + 1);
        }
    }

```

```

37     }
    }
}

3.8 Minimum Mean Cycle
1 // d[i][j] == 0 if {i,j} !in E
long long d[1003][1003], dp[1003][1003];
3
pair<long long, long long> MMWC() {
5     memset(dp, 0x3f, sizeof(dp));
    for (int i = 1; i <= n; ++i) dp[0][i] = 0;
7     for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n; ++j) {
9             for (int k = 1; k <= n; ++k) {
                dp[i][k] = min(dp[i - 1][j] + d[j][k], dp[i][k]);
11            }
        }
13    }
    long long au = 1ll << 31, ad = 1;
15    for (int i = 1; i <= n; ++i) {
        if (dp[n][i] == 0x3f3f3f3f3f3f3f3f) continue;
17        long long u = 0, d = 1;
        for (int j = n - 1; j >= 0; --j) {
19            if ((dp[n][i] - dp[j][i]) * d > u * (n - j)) {
                u = dp[n][i] - dp[j][i];
21                d = n - j;
            }
23        }
        if (u * ad < au * d) au = u, ad = d;
25    }
    long long g = __gcd(au, ad);
27    return make_pair(au / g, ad / g);
}

```

3.9 Directed MST

```

1 template <typename T> struct DMST {
    T g[maxn][maxn], fw[maxn];
3     int n, fr[maxn];
    bool vis[maxn], inc[maxn];
5     void clear() {
        for (int i = 0; i < maxn; ++i) {
7            for (int j = 0; j < maxn; ++j) g[i][j] = inf;
            vis[i] = inc[i] = false;
9        }
    }
11    void addedge(int u, int v, T w) {
        g[u][v] = min(g[u][v], w);
13    }
    T operator()(int root, int _n) {
15        n = _n;
        if (dfs(root) != n) return -1;
17        T ans = 0;
        while (true) {
19            for (int i = 1; i <= n; ++i) fw[i] = inf, fr[i] = i;
            for (int i = 1; i <= n; ++i)
21                if (!inc[i]) {
                    for (int j = 1; j <= n; ++j) {
23                        if (!inc[j] && i != j && g[j][i] < fw[i]) {
                            fw[i] = g[j][i];
25                            fr[i] = j;
                        }
                    }
27                }
            int x = -1;
29            for (int i = 1; i <= n; ++i)
                if (i != root && !inc[i]) {
31                    int j = i, c = 0;
                    while (j != root && fr[j] != i && c <= n)
33                        ++c, j = fr[j];
                    if (j == root || c > n) continue;
35                    else {
                        x = i;
                        break;
37                    }
                }
            if (!x) {
41                for (int i = 1; i <= n; ++i)
                    if (i != root && !inc[i]) ans += fw[i];
                    return ans;
43            }
            int y = x;
45            for (int i = 1; i <= n; ++i) vis[i] = false;
            do {
47                ans += fw[y];
                y = fr[y];
                vis[y] = inc[y] = true;
51            } while (y != x);
            inc[x] = false;
53            for (int k = 1; k <= n; ++k)
                if (vis[k]) {
55                    if (vis[k]) {

```

```

57     for (int j = 1; j <= n; ++j)
58         if (!vis[j]) {
59             if (g[x][j] > g[k][j]) g[x][j] = g[k][j];
60             if (g[j][k] < inf &&
61                 g[j][k] - fw[k] < g[j][x])
62                 g[j][x] = g[j][k] - fw[k];
63         }
64     }
65     return ans;
66 }
67 int dfs(int now) {
68     int r = 1;
69     vis[now] = true;
70     for (int i = 1; i <= n; ++i)
71         if (g[now][i] < inf && !vis[i]) r += dfs(i);
72     return r;
73 }

```

3.10 Maximum Clique

```

1 // source: KACTL
3 typedef vector<bitset<200>> vb;
4 struct Maxclique {
5     double limit = 0.025, pk = 0;
6     struct Vertex {
7         int i, d = 0;
8     };
9     typedef vector<Vertex> vv;
10    vb e;
11    vv V;
12    vector<vi> C;
13    vi qmax, q, S, old;
14    void init(vv &r) {
15        for (auto &v : r) v.d = 0;
16        for (auto &v : r)
17            for (auto j : r) v.d += e[v.i][j.i];
18        sort(all(r), [](auto a, auto b) { return a.d > b.d; });
19        int mxD = r[0].d;
20        rep(i, 0, sz(r)) r[i].d = min(i, mxD) + 1;
21    }
22    void expand(vv &R, int lev = 1) {
23        S[lev] += S[lev - 1] - old[lev];
24        old[lev] = S[lev - 1];
25        while (sz(R)) {
26            if (sz(q) + R.back().d <= sz(qmax)) return;
27            q.push_back(R.back().i);
28            vv T;
29            for (auto v : R)
30                if (e[R.back().i][v.i]) T.push_back({v.i});
31            if (sz(T)) {
32                if (S[lev]++ / ++pk < limit) init(T);
33                int j = 0, mxk = 1,
34                    mnk = max(sz(qmax) - sz(q) + 1, 1);
35                C[1].clear(), C[2].clear();
36                for (auto v : T) {
37                    int k = 1;
38                    auto f = [&](int i) { return e[v.i][i]; };
39                    while (any_of(all(C[k]), f)) k++;
40                    if (k > mxk) mxk = k, C[mxk + 1].clear();
41                    if (k < mnk) T[j++].i = v.i;
42                    C[k].push_back(v.i);
43                }
44                if (j > 0) T[j - 1].d = 0;
45                rep(k, mnk, mxk + 1) for (int i : C[k]) T[j].i = i,
46                    T[j++].d = k;
47                expand(T, lev + 1);
48            } else if (sz(q) > sz(qmax)) qmax = q;
49            q.pop_back(), R.pop_back();
50        }
51    }
52    vi maxClique() {
53        init(V), expand(V);
54        return qmax;
55    }
56    Maxclique(vb conn)
57        : e(conn), C(sz(e) + 1), S(sz(C)), old(S) {
58        rep(i, 0, sz(e)) V.push_back({i});
59    }
60 }

```

3.11 Dominator Tree

```

1 // idom[n] is the unique node that strictly dominates n but
2 // does not strictly dominate any other node that strictly
3 // dominates n. idom[n] = 0 if n is entry or the entry
4 // cannot reach n.
5 struct DominatorTree {

```

```

6     static const int MAXN = 200010;
7     int n, s;
8     vector<int> g[MAXN], pred[MAXN];
9     vector<int> cov[MAXN];
10    int dfn[MAXN], nfd[MAXN], ts;
11    int par[MAXN];
12    int sdom[MAXN], idom[MAXN];
13    int mom[MAXN], mn[MAXN];
14
15    inline bool cmp(int u, int v) { return dfn[u] < dfn[v]; }
16
17    int eval(int u) {
18        if (mom[u] == u) return u;
19        int res = eval(mom[u]);
20        if (cmp(sdom[mn[mom[u]]], sdom[mn[u]]))
21            mn[u] = mn[mom[u]];
22        return mom[u] = res;
23    }
24
25    void init(int _n, int _s) {
26        n = _n;
27        s = _s;
28        REP1(i, 1, n) {
29            g[i].clear();
30            pred[i].clear();
31            idom[i] = 0;
32        }
33    }
34    void add_edge(int u, int v) {
35        g[u].push_back(v);
36        pred[v].push_back(u);
37    }
38    void DFS(int u) {
39        ts++;
40        dfn[u] = ts;
41        nfd[ts] = u;
42        for (int v : g[u])
43            if (dfn[v] == 0) {
44                par[v] = u;
45                DFS(v);
46            }
47    }
48    void build() {
49        ts = 0;
50        REP1(i, 1, n) {
51            dfn[i] = nfd[i] = 0;
52            cov[i].clear();
53            mom[i] = mn[i] = sdom[i] = i;
54        }
55        DFS(s);
56        for (int i = ts; i >= 2; i--) {
57            int u = nfd[i];
58            if (u == 0) continue;
59            for (int v : pred[u])
60                if (dfn[v]) {
61                    eval(v);
62                    if (cmp(sdom[mn[v]], sdom[u]))
63                        sdom[u] = sdom[mn[v]];
64                }
65            cov[sdom[u]].push_back(u);
66            mom[u] = par[u];
67            for (int w : cov[par[u]]) {
68                eval(w);
69                if (cmp(sdom[mn[w]], par[u])) idom[w] = mn[w];
70                else idom[w] = par[u];
71            }
72            cov[par[u]].clear();
73        }
74        REP1(i, 2, ts) {
75            int u = nfd[i];
76            if (u == 0) continue;
77            if (idom[u] != sdom[u]) idom[u] = idom[idom[u]];
78        }
79    } dom;

```

3.12 Manhattan Distance MST

```

1 // returns [(dist, from, to), ...]
2 // then do normal mst afterwards
3 typedef Point<int> P;
4 vector<array<int, 3>> manhattanMST(vector<P> ps) {
5     vi id(sz(ps));
6     iota(all(id), 0);
7     vector<array<int, 3>> edges;
8     rep(k, 0, 4) {
9         sort(all(id), [&](int i, int j) {
10             return (ps[i] - ps[j]).x < (ps[j] - ps[i]).y;
11         });
12         map<int, int> sweep;
13         for (int i : id) {

```

```

    for (auto it = sweep.lower_bound(-ps[i].y);
         it != sweep.end(); sweep.erase(it++)) {
        int j = it->second;
        P d = ps[i] - ps[j];
        if (d.y > d.x) break;
        edges.push_back({d.y + d.x, i, j});
    }
    sweep[-ps[i].y] = i;
}
for (P &p : ps)
    if (k & 1) p.x = -p.x;
    else swap(p.x, p.y);
}
return edges;
}

```

3.13 Virtual Tree

```

1 // id[u] is the index of u in pre-order traversal
vector<pii> build(vector<int> h) {
3   sort(h.begin(), h.end(),
        [&](int u, int v) { return id[u] < id[v]; });
5   int root = h[0], top = 0;
   for (int i : h) root = lca(i, root);
7   vector<int> stk(h.size(), root);
   vector<pii> e;
9   for (int u : h) {
       if (u == root) continue;
       int l = lca(u, stk[top]);
       if (l != stk[top]) {
13          while (id[l] < id[stk[top - 1]])
              e.emplace_back(stk[top - 1], stk[top]), top--;
15          e.emplace_back(stk[top], l), top--;
              if (l != stk[top]) stk[++top] = l;
17          }
       stk[++top] = u;
19   }
   while (top) e.emplace_back(stk[top - 1], stk[top]),
top--;
21   return e;
}

```

4 Math

4.1 Number Theory

4.1.1 Mod Struct

A list of safe primes:

- 26003, 27767, 28319, 28979, 29243, 29759, 30467
- 910927547, 919012223, 947326223, 990669467, 1007939579, 1019126699
- 929760389146037459, 975500632317046523, 989312547895528379

NTT prime p	$p - 1$	primitive root
65537	$1 \ll 16$	3
469762049	$7 \ll 26$	3
998244353	$119 \ll 23$	3
2748779069441	$5 \ll 39$	3
1945555039024054273	$27 \ll 56$	5

```

1 template <typename T> struct M {
    static T MOD; // change to constexpr if already known
3   T v;
   M(T x = 0) {
5       v = (-MOD <= x && x < MOD) ? x : x % MOD;
       if (v < 0) v += MOD;
7   }
   explicit operator T() const { return v; }
9   bool operator==(const M &b) const { return v == b.v; }
   bool operator!=(const M &b) const { return v != b.v; }
11  M operator-() { return M(-v); }
   M operator+(M b) { return M(v + b.v); }
13  M operator-(M b) { return M(v - b.v); }
   M operator*(M b) { return M((__int128)v * b.v % MOD); }
15  M operator/(M b) { return *this * (b ^ (MOD - 2)); }
   // change above implementation to this if MOD is not prime
17  M inv() {
       auto [p, _, g] = extgcd(v, MOD);
19       return assert(g == 1), p;
   }
21  friend M operator^(M a, ll b) {
       M ans(1);
23       for (; b >= 1, a *= a)
           if (b & 1) ans *= a;

```

```

25     return ans;
   }
27   friend M &operator+=(M &a, M b) { return a = a + b; }
28   friend M &operator-=(M &a, M b) { return a = a - b; }
29   friend M &operator*=(M &a, M b) { return a = a * b; }
30   friend M &operator/=(M &a, M b) { return a = a / b; }
31 };
using Mod = M<int>;
33 template <> int Mod::MOD = 1'000'000'007;
int &MOD = Mod::MOD;

```

4.1.2 Miller-Rabin

```

1 // checks if Mod::MOD is prime
bool is_prime() {
3   if (MOD < 2 || MOD % 2 == 0) return MOD == 2;
   Mod A[] = {2, 7, 61}; // for int values (< 2^31)
5   // ll: 2, 325, 9375, 28178, 450775, 9780504, 1795265022
   int s = __builtin_ctzll(MOD - 1), i;
7   for (Mod a : A) {
       Mod x = a ^ (MOD >> s);
9       for (i = 0; i < s && (x + 1).v > 2; i++) x *= x;
       if (i && x != -1) return 0;
11  }
   return 1;
13 }

```

4.1.3 Linear Sieve

```

1 constexpr ll MAXN = 1000000;
bitset<MAXN> is_prime;
3 vector<ll> primes;
ll mpf[MAXN], phi[MAXN], mu[MAXN];
5
void sieve() {
7   is_prime.set();
   is_prime[1] = 0;
9   mu[1] = phi[1] = 1;
   for (ll i = 2; i < MAXN; i++) {
11      if (is_prime[i]) {
          mpf[i] = i;
13          primes.push_back(i);
          phi[i] = i - 1;
15          mu[i] = -1;
       }
       for (ll p : primes) {
17          if (p > mpf[i] || i * p >= MAXN) break;
19          is_prime[i * p] = 0;
          mpf[i * p] = p;
21          mu[i * p] = -mu[i];
          if (i % p == 0)
23             phi[i * p] = phi[i] * p, mu[i * p] = 0;
          else phi[i * p] = phi[i] * (p - 1);
25      }
27 }

```

4.1.4 Get Factors

```

1 vector<ll> all_factors(ll n) {
   vector<ll> fac = {1};
3   while (n > 1) {
       const ll p = mpf[n];
       vector<ll> cur = {1};
5       while (n % p == 0) {
           n /= p;
7           cur.push_back(cur.back() * p);
           }
9       vector<ll> tmp;
       for (auto x : fac)
11          for (auto y : cur) tmp.push_back(x * y);
       tmp.swap(fac);
13   }
15   return fac;
}

```

4.1.5 Binary GCD

```

1 // returns the gcd of non-negative a, b
ull bin_gcd(ull a, ull b) {
3   if (!a || !b) return a + b;
   int s = __builtin_ctzll(a | b);
5   a >>= __builtin_ctzll(a);
   while (b) {
7       if ((b >>= __builtin_ctzll(b)) < a) swap(a, b);
       b -= a;
9   }
   return a << s;
11 }

```

4.1.6 Extended GCD


```

1 // returns (p, q, g): p * a + q * b == g == gcd(a, b)
  // g is not guaranteed to be positive when a < 0 or b < 0
3 tuple<ll, ll, ll> extgcd(ll a, ll b) {
  ll s = 1, t = 0, u = 0, v = 1;
  5 while (b) {
    ll q = a / b;
    7 swap(a -= q * b, b);
    swap(s -= q * t, t);
    9 swap(u -= q * v, v);
  }
  11 return {s, u, a};
}

```

4.1.7 Chinese Remainder Theorem

```

1 // for 0 <= a < m, 0 <= b < n, returns the smallest x >= 0
  // such that x % m == a and x % n == b
3 ll crt(ll a, ll m, ll b, ll n) {
  if (n > m) swap(a, b), swap(m, n);
  5 auto [x, y, g] = extgcd(m, n);
  assert((a - b) % g == 0); // no solution
  7 x = ((b - a) / g * x) % (n / g) * m + a;
  return x < 0 ? x + m / g * n : x;
  9 }

```

4.1.8 Baby-Step Giant-Step

```

1 // returns x such that a ^ x = b where x \in [l, r)
  ll bsgs(Mod a, Mod b, ll l = 0, ll r = MOD - 1) {
  3 int m = sqrt(r - l) + 1, i;
  unordered_map<ll, ll> tb;
  5 Mod d = (a ^ l) / b;
  for (i = 0, d = (a ^ l) / b; i < m; i++, d *= a)
    7 if (d == 1) return l + i;
    else tb[(ll)d] = l + i;
  9 Mod c = Mod(1) / (a ^ m);
  for (i = 0, d = 1; i < m; i++, d *= c)
    11 if (auto j = tb.find((ll)d); j != tb.end())
      return j->second + i * m;
  13 return assert(0), -1; // no solution
}

```

4.1.9 Pohlig-Hellman Algorithm

Goal: Find an integer x such that $g^x = h$ in an order p^e group.

1. Let $x = 0$ and $\gamma = g^{p^{e-1}}$.
2. For $k = 0, 1, \dots, e - 1$:
Let $c = (g^{-x}h)^{p^{e-1-k}}$, and compute d such that $\gamma^d = c$.
Set $x = x + p^k d$.

4.1.10 Pollard's Rho

```

1 ll f(ll x, ll mod) { return (x * x + 1) % mod; }
  // n should be composite
3 ll pollard_rho(ll n) {
  if (!(n & 1)) return 2;
  5 while (1) {
    ll y = 2, x = RNG() % (n - 1) + 1, res = 1;
    7 for (int sz = 2; res == 1; sz *= 2) {
      for (int i = 0; i < sz && res <= 1; i++) {
        x = f(x, n);
        res = __gcd(abs(x - y), n);
      }
      y = x;
    }
    13 if (res != 0 && res != n) return res;
  }
  15 }
}

```

4.1.11 Tonelli-Shanks Algorithm

```

1 int legendre(Mod a) {
  if (a == 0) return 0;
  3 return (a ^ ((MOD - 1) / 2)) == 1 ? 1 : -1;
}
5 Mod sqrt(Mod a) {
  assert(legendre(a) != -1); // no solution
  7 ll p = MOD, s = p - 1;
  if (a == 0) return 0;
  if (p == 2) return 1;
  if (p % 4 == 3) return a ^ ((p + 1) / 4);
  11 int r, m;
  for (r = 0; !(s & 1); r++) s >>= 1;
  13 Mod n = 2;
  while (legendre(n) != -1) n += 1;
  15 Mod x = a ^ ((s + 1) / 2), b = a ^ s, g = n ^ s;
  while (b != 1) {
    17 Mod t = b;
    for (m = 0; t != 1; m++) t *= t;
    19 Mod gs = g ^ (1LL << (r - m - 1));

```

```

    g = gs * gs, x *= gs, b *= g, r = m;
  21 }
  return x;
  23 }
  // to get sqrt(X) modulo p^k, where p is an odd prime:
  25 // c = x^2 (mod p), c = X^2 (mod p^k), q = p^(k-1)
  // X = x^q * c^((p^k-2q+1)/2) (mod p^k)

```

4.1.12 Chinese Sieve

```

1 const ll N = 1000000;
  // f, g, h multiplicative, h = f (dirichlet convolution) g
3 ll pre_g(ll n);
  ll pre_h(ll n);
  5 // preprocessed prefix sum of f
  ll pre_f[N];
  7 // prefix sum of multiplicative function f
  ll solve_f(ll n) {
    9 static unordered_map<ll, ll> m;
    if (n < N) return pre_f[n];
    11 if (m.count(n)) return m[n];
    ll ans = pre_h(n);
    13 for (ll l = 2, r; l <= n; l = r + 1) {
      r = n / (n / l);
      15 ans -= (pre_g(r) - pre_g(l - 1)) * djs_f(n / l);
    }
    17 return m[n] = ans;
  }

```

4.1.13 Rational Number Binary Search

```

1 struct QQ {
  ll p, q;
  3 QQ go(QQ b, ll d) { return {p + b.p * d, q + b.q * d}; }
};
5 bool pred(QQ);
  // returns smallest p/q in [lo, hi] such that
  7 // pred(p/q) is true, and 0 <= p, q <= N
  QQ frac_bs(ll N) {
    9 QQ lo{0, 1}, hi{1, 0};
    if (pred(lo)) return lo;
    11 assert(pred(hi));
    bool dir = 1, L = 1, H = 1;
    13 for (; L || H; dir = !dir) {
      ll len = 0, step = 1;
      15 for (int t = 0; t < 2 && (t ? step /= 2 : step *= 2);)
        if (QQ mid = hi.go(lo, len + step);
          17 mid.p > N || mid.q > N || dir ^ pred(mid))
          t++;
      19 else len += step;
      swap(lo, hi = hi.go(lo, len));
      21 (dir ? L : H) = !len;
    }
    23 return dir ? hi : lo;
  }

```

4.1.14 Farey Sequence

```

1 // returns (e/f), where (a/b, c/d, e/f) are
  // three consecutive terms in the order n farey sequence
3 // to start, call next_farey(n, 0, 1, 1, n)
  pll next_farey(ll n, ll a, ll b, ll c, ll d) {
    5 ll p = (n + b) / d;
    return pll(p * c - a, p * d - b);
    7 }

```

4.2 Combinatorics

4.2.1 Matroid Intersection

This template assumes 2 weighted matroids of the same type, and that removing an element is much more expensive than checking if one can be added. **Remember to change the implementation details.**

The ground set is $0, 1, \dots, n - 1$, where element i has weight $w[i]$. For the unweighted version, remove weights and change BF/SPFA to BFS.

```

1 constexpr int N = 100;
  constexpr int INF = 1e9;
  3
  struct Matroid {
    5 Matroid(bitset<N>); // initialize from an independent set
    bool can_add(int); // if adding will break independence
    7 Matroid remove(int); // removing from the set
  };
  9
  auto matroid_intersection(int n, const vector<int> &w) {

```

```

11  bitset<N> S;
    for (int sz = 1; sz <= n; sz++) {
13      Matroid M1(S), M2(S);

15      vector<vector<pii>> e(n + 2);
      for (int j = 0; j < n; j++)
17        if (!S[j]) {
            if (M1.can_add(j)) e[n].emplace_back(j, -w[j]);
            if (M2.can_add(j)) e[j].emplace_back(n + 1, 0);
19        }
      for (int i = 0; i < n; i++)
21        if (S[i]) {
            Matroid T1 = M1.remove(i), T2 = M2.remove(i);
            for (int j = 0; j < n; j++)
23              if (!S[j]) {
                  if (T1.can_add(j)) e[i].emplace_back(j, -w[j]);
                  if (T2.can_add(j)) e[j].emplace_back(i, w[i]);
25              }
27        }
29    }

31    vector<pii> dis(n + 2, {INF, 0});
    vector<int> prev(n + 2, -1);
33    dis[n] = {0, 0};
    // change to SPFA for more speed, if necessary
35    bool upd = 1;
    while (upd) {
37      upd = 0;
      for (int u = 0; u < n + 2; u++)
39        for (auto [v, c] : e[u]) {
            pii x(dis[u].first + c, dis[u].second + 1);
            if (x < dis[v]) dis[v] = x, prev[v] = u, upd = 1;
41        }
43    }

45    if (dis[n + 1].first < INF)
        for (int x = prev[n + 1]; x != n; x = prev[x])
47      S.flip(x);
    else break;
49    // S is the max-weighted independent set with size sz
51  }
  return S;
53 }

```

4.2.2 De Bruijn Sequence

```

1  int res[kN], aux[kN], a[kN], sz;
  void Rec(int t, int p, int n, int k) {
3    if (t > n) {
        if (n % p == 0)
5      for (int i = 1; i <= p; ++i) res[sz++] = aux[i];
    } else {
        aux[t] = aux[t - p];
        Rec(t + 1, p, n, k);
9      for (aux[t] = aux[t - p] + 1; aux[t] < k; ++aux[t])
            Rec(t + 1, t, n, k);
11    }
  }

13  int DeBruijn(int k, int n) {
    // return cyclic string of length k^n such that every
15    // string of length n using k character appears as a
    // substring.
17    if (k == 1) return res[0] = 0, 1;
    fill(aux, aux + k * n, 0);
19    return sz = 0, Rec(1, 1, n, k), sz;
  }

```

4.2.3 Multinomial

```

1  // ways to permute v[i]
  ll multinomial(vi &v) {
3    ll c = 1, m = v.empty() ? 1 : v[0];
    for (int i = 1; i < v.size(); i++)
5      for (int j = 0; i < v[i]; j++) c = c * ++m / (j + 1);
    return c;
7  }

```

4.3 Theorems

Kirchhoff's Theorem

Denote L be a $n \times n$ matrix as the Laplacian matrix of graph G , where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i, j) in G .

- The number of undirected spanning in G is $|\det(\tilde{L}_{11})|$.
- The number of directed spanning tree rooted at r in G is $|\det(\tilde{L}_{rr})|$.

Tutte's Matrix

Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniformly at random) if $i < j$ and $(i, j) \in E$, otherwise $d_{ij} = -d_{ji} \cdot \frac{\text{rank}(D)}{2}$ is the maximum matching on G .

Cayley's Formula

- Given a degree sequence d_1, d_2, \dots, d_n for each labeled vertices, there are

$$\frac{(n-2)!}{(d_1-1)!(d_2-1)!\dots(d_n-1)!}$$

spanning trees.

- Let $T_{n,k}$ be the number of labeled forests on n vertices with k components, such that vertex $1, 2, \dots, k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.

Erdős-Gallai Theorem

A sequence of non-negative integers $d_1 \geq d_2 \geq \dots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + d_2 + \dots + d_n$ is even and

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$$

holds for all $1 \leq k \leq n$.

Gale-Ryser Theorem

Two sequences of non-negative integers $a_1 \geq a_2 \geq \dots \geq a_n$ and b_1, b_2, \dots, b_n can be represented as the degree sequence of two partitions of a simple bipartite graph on $2n$ vertices if and only if $a_1 + a_2 + \dots + a_n = b_1 + b_2 + \dots + b_n$ and

$$\sum_{i=1}^k a_i \leq \sum_{i=1}^n \min(b_i, k)$$

holds for all $1 \leq k \leq n$.

Burnside's Lemma

Let X be a set and G be a group that acts on X . For $g \in G$, denote by X^g the elements fixed by g :

$$X^g = \{x \in X \mid gx = x\}$$

Then

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|.$$

Gram-Schmidt Process

Let $\mathbf{v}_1, \mathbf{v}_2, \dots$ be linearly independent vectors, then the orthogonalized vectors are

$$\mathbf{u}_i = \mathbf{v}_i - \sum_{j=1}^{i-1} \frac{\langle \mathbf{u}_j, \mathbf{v}_i \rangle}{\langle \mathbf{u}_j, \mathbf{u}_j \rangle} \mathbf{u}_j$$

5 Numeric

5.1 Barrett Reduction

```

1  using ull = unsigned long long;
  using ul = __uint128_t;
3  // very fast calculation of a % m
  struct reduction {
5    const ull m, d;
    explicit reduction(ull m) : m(m), d(((ul)1 << 64) / m) {}
7    inline ull operator()(ull a) const {
        ull q = (ull)((ul)d * a) >> 64;
9    return (a - q * m) >= m ? a - m : a;
  }

```

```

    }
11 };

```

5.2 Long Long Multiplication

```

1 using ull = unsigned long long;
  using ll = long long;
  using ld = long double;
  // returns a * b % M where a, b < M < 2**63
5 ull mult(ull a, ull b, ull M) {
  ll ret = a * b - M * ull(ld(a) * ld(b) / ld(M));
7  return ret + M * (ret < 0) - M * (ret >= (ll)M);
}

```

5.3 Fast Fourier Transform

```

1 template <typename T>
  void fft_(int n, vector<T> &a, vector<T> &rt, bool inv) {
3   vector<int> br(n);
   for (int i = 1; i < n; i++) {
5     br[i] = (i & 1) ? br[i - 1] + n / 2 : br[i / 2] / 2;
     if (br[i] > i) swap(a[i], a[br[i]]);
7   }
   for (int len = 2; len <= n; len *= 2)
9     for (int i = 0; i < n; i += len)
       for (int j = 0; j < len / 2; j++) {
11        int pos = n / len * (inv ? len - j : j);
          T u = a[i + j], v = a[i + j + len / 2] * rt[pos];
13        a[i + j] = u + v, a[i + j + len / 2] = u - v;
       }
15   if (T minv = T(1) / T(n); inv)
     for (T &x : a) x *= minv;
17 }

1 void ntt(vector<Mod> &a, bool inv, Mod primitive_root) {
   int n = a.size();
3   Mod root = primitive_root ^ (MOD - 1) / n;
   vector<Mod> rt(n + 1, 1);
5   for (int i = 0; i < n; i++) rt[i + 1] = rt[i] * root;
   fft_(n, a, rt, inv);
7 }

  void fft(vector<complex<double>> &a, bool inv) {
9   int n = a.size();
   vector<complex<double>> rt(n + 1);
11  double arg = acos(-1) * 2 / n;
   for (int i = 0; i <= n; i++)
13    rt[i] = {cos(arg * i), sin(arg * i)};
   fft_(n, a, rt, inv);
15 }

```

5.4 Fast Walsh-Hadamard Transform

```

1 void fwht(vector<Mod> &a, bool inv) {
   int n = a.size();
3   for (int d = 1; d < n; d <= 1)
     for (int m = 0; m < n; m++)
5     if (!(m & d)) {
       inv ? a[m] -= a[m | d] : a[m] += a[m | d]; // AND
7     inv ? a[m | d] -= a[m] : a[m | d] += a[m]; // OR
       Mod x = a[m], y = a[m | d]; // XOR
9     a[m] = x + y, a[m | d] = x - y; // XOR
     }
11  if (Mod iv = Mod(1) / n; inv) // XOR
     for (Mod &i : a) i *= iv; // XOR
13 }

```

5.5 Subset Convolution

```

1 #pragma GCC target("popcnt")
  #include <immintrin.h>
3
  void fwht(int n, vector<vector<Mod>> &a, bool inv) {
5   for (int h = 0; h < n; h++)
     for (int i = 0; i < (1 << n); i++)
7     if (!(i & (1 << h)))
       for (int k = 0; k <= n; k++)
9       inv ? a[i | (1 << h)][k] -= a[i][k]
          : a[i | (1 << h)][k] += a[i][k];
11  }
  // c[k] = sum(popcnt(i & j) == sz && i | j == k) a[i] * b[j]
13 vector<Mod> subset_convolution(int n, int sz,
                                const vector<Mod> &a_,
                                const vector<Mod> &b_) {
15   int len = n + sz + 1, N = 1 << n;
   vector<vector<Mod>> a(1 << n, vector<Mod>(len, 0)), b =
a;
17   for (int i = 0; i < N; i++)
     a[i][_mm_popcnt_u64(i)] = a_[i],
     b[i][_mm_popcnt_u64(i)] = b_[i];
21  fwht(n, a, 0), fwht(n, b, 0);
   for (int i = 0; i < N; i++) {

```

```

23   vector<Mod> tmp(len);
   for (int j = 0; j < len; j++)
25     for (int k = 0; k <= j; k++)
       tmp[j] += a[i][k] * b[i][j - k];
27   a[i] = tmp;
   }
29  fwht(n, a, 1);
   vector<Mod> c(N);
31  for (int i = 0; i < N; i++)
     c[i] = a[i][_mm_popcnt_u64(i) + sz];
33  return c;
}

```

5.6 Linear Recurrences

5.6.1 Berlekamp-Massey Algorithm

```

1 template <typename T>
  vector<T> berlekamp_massey(const vector<T> &s) {
3   int n = s.size(), l = 0, m = 1;
   vector<T> r(n), p(n);
5   r[0] = p[0] = 1;
   T b = 1, d = 0;
7   for (int i = 0; i < n; i++, m++, d = 0) {
     for (int j = 0; j <= l; j++) d += r[j] * s[i - j];
9     if ((d /= b) == 0) continue; // change if T is float
     auto t = r;
11    for (int j = m; j < n; j++) r[j] -= d * p[j - m];
     if (l * 2 <= i) l = i + 1 - l, b *= d, m = 0, p = t;
13  }
  return r.resize(l + 1), reverse(r.begin(), r.end()), r;
15 }

```

5.6.2 Linear Recurrence Calculation

```

1 template <typename T> struct lin_rec {
  using poly = vector<T>;
3   poly mul(poly a, poly b, poly m) {
     int n = m.size();
5     poly r(n);
     for (int i = n - 1; i >= 0; i--) {
7       r.insert(r.begin(), 0), r.pop_back();
       T c = r[n - 1] + a[n - 1] * b[i];
9       // c /= m[n - 1]; if m is not monic
       for (int j = 0; j < n; j++)
11        r[j] += a[j] * b[i] - c * m[j];
     }
13   return r;
   }

15   poly pow(poly p, ll k, poly m) {
     poly r(m.size());
17     r[0] = 1;
     for (; k >= 1; p = mul(p, p, m))
19       if (k & 1) r = mul(r, p, m);
     return r;
21   }

  T calc(poly t, poly r, ll k) {
23     int n = r.size();
     poly p(n);
25     p[1] = 1;
     poly q = pow(p, k, r);
27     T ans = 0;
     for (int i = 0; i < n; i++) ans += t[i] * q[i];
29     return ans;
   }
31 };

```

5.7 Matrices

5.7.1 Determinant

```

1 Mod det(vector<vector<Mod>> a) {
   int n = a.size();
3   Mod ans = 1;
   for (int i = 0; i < n; i++) {
5     int b = i;
     for (int j = i + 1; j < n; j++)
7       if (a[j][i] != 0) {
         b = j;
         break;
9     }
     if (i != b) swap(a[i], a[b]), ans = -ans;
     ans *= a[i][i];
11    if (ans == 0) return 0;
     for (int j = i + 1; j < n; j++) {
13       Mod v = a[j][i] / a[i][i];
       if (v != 0)
15         for (int k = i + 1; k < n; k++)
           a[j][k] -= v * a[i][k];
17     }
19 }

```

```

    }
21 return ans;
}

1 double det(vector<vector<double>> a) {
    int n = a.size();
    double ans = 1;
    for (int i = 0; i < n; i++) {
        int b = i;
        for (int j = i + 1; j < n; j++)
            if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
        if (i != b) swap(a[i], a[b]), ans = -ans;
        ans *= a[i][i];
        if (ans == 0) return 0;
        for (int j = i + 1; j < n; j++) {
            double v = a[j][i] / a[i][i];
            if (v != 0)
                for (int k = i + 1; k < n; k++)
                    a[j][k] -= v * a[i][k];
        }
    }
    return ans;
}
19 }

```

5.7.2 Inverse

```

1 // Returns rank.
// Result is stored in A unless singular (rank < n).
3 // For prime powers, repeatedly set
//  $A^{-1} = A^{-1} (2I - A * A^{-1}) \pmod{p^k}$ 
5 // where  $A^{-1}$  starts as the inverse of  $A \pmod{p}$ ,
// and k is doubled in each step.
7
9 int matInv(vector<vector<double>> &A) {
    int n = sz(A);
    vi col(n);
    vector<vector<double>> tmp(n, vector<double>(n));
    rep(i, 0, n) tmp[i][i] = 1, col[i] = i;

    rep(i, 0, n) {
        int r = i, c = i;
        rep(j, i, n)
            rep(k, i, n) if (fabs(A[j][k]) > fabs(A[r][c])) r = j, c = k;

        if (fabs(A[r][c]) < 1e-12) return i;
        A[i].swap(A[r]);
        tmp[i].swap(tmp[r]);
        rep(j, 0, n) swap(A[j][i], A[j][c]),
            swap(tmp[j][i], tmp[j][c]);
        swap(col[i], col[c]);
        double v = A[i][i];
        rep(j, i + 1, n) {
            double f = A[j][i] / v;
            A[j][i] = 0;
            rep(k, i + 1, n) A[j][k] -= f * A[i][k];
            rep(k, 0, n) tmp[j][k] -= f * tmp[i][k];
        }
        rep(j, i + 1, n) A[i][j] /= v;
        rep(j, 0, n) tmp[i][j] /= v;
        A[i][i] = 1;
    }

    /// forget A at this point, just eliminate tmp backward
    for (int i = n - 1; i > 0; --i) rep(j, 0, i) {
        double v = A[j][i];
        rep(k, 0, n) tmp[j][k] -= v * tmp[i][k];
    }

    rep(i, 0, n) rep(j, 0, n) A[col[i]][col[j]] = tmp[i][j];
    return n;
}

47 int matInv_mod(vector<vector<ll>> &A) {
    int n = sz(A);
    vi col(n);
    vector<vector<ll>> tmp(n, vector<ll>(n));
    rep(i, 0, n) tmp[i][i] = 1, col[i] = i;

    rep(i, 0, n) {
        int r = i, c = i;
        rep(j, i, n) rep(k, i, n) if (A[j][k]) {
            r = j;
            c = k;
            goto found;
        }
        return i;
    found:
        A[i].swap(A[r]);
        tmp[i].swap(tmp[r]);
        rep(j, 0, n) swap(A[j][i], A[j][c]),
            swap(tmp[j][i], tmp[j][c]);
    }
}
65

```

```

        swap(col[i], col[c]);
        ll v = modpow(A[i][i], mod - 2);
        rep(j, i + 1, n) {
            ll f = A[j][i] * v % mod;
            A[j][i] = 0;
            rep(k, i + 1, n) A[j][k] =
                (A[j][k] - f * A[i][k]) % mod;
            rep(k, 0, n) tmp[j][k] =
                (tmp[j][k] - f * tmp[i][k]) % mod;
        }
        rep(j, i + 1, n) A[i][j] = A[i][j] * v % mod;
        rep(j, 0, n) tmp[i][j] = tmp[i][j] * v % mod;
        A[i][i] = 1;
    }

    for (int i = n - 1; i > 0; --i) rep(j, 0, i) {
        ll v = A[j][i];
        rep(k, 0, n) tmp[j][k] =
            (tmp[j][k] - v * tmp[i][k]) % mod;
    }

    rep(i, 0, n) rep(j, 0, n) A[col[i]][col[j]] =
        tmp[i][j] % mod + (tmp[i][j] < 0 ? mod : 0);
    return n;
}

```

5.7.3 Characteristic Polynomial

```

1 // calculate det(a - xI)
template <typename T>
3 vector<T> CharacteristicPolynomial(vector<vector<T>> a) {
    int N = a.size();

    for (int j = 0; j < N - 2; j++) {
        for (int i = j + 1; i < N; i++) {
            if (a[i][j] != 0) {
                swap(a[j + 1], a[i]);
                for (int k = 0; k < N; k++)
                    swap(a[k][j + 1], a[k][i]);
                break;
            }
        }
        if (a[j + 1][j] != 0) {
            T inv = T(1) / a[j + 1][j];
            for (int i = j + 2; i < N; i++) {
                if (a[i][j] == 0) continue;
                T coe = inv * a[i][j];
                for (int l = j; l < N; l++)
                    a[i][l] -= coe * a[j + 1][l];
                for (int k = 0; k < N; k++)
                    a[k][j + 1] += coe * a[k][i];
            }
        }
    }

    vector<vector<T>> p(N + 1);
    p[0] = {T(1)};
    for (int i = 1; i <= N; i++) {
        p[i].resize(i + 1);
        for (int j = 0; j < i; j++) {
            p[i][j + 1] -= p[i - 1][j];
            p[i][j] += p[i - 1][j] * a[i - 1][i - 1];
        }
        T x = 1;
        for (int m = 1; m < i; m++) {
            x *= -a[i - m][i - m - 1];
            T coe = x * a[i - m - 1][i - 1];
            for (int j = 0; j < i - m; j++)
                p[i][j] += coe * p[i - m - 1][j];
        }
    }
    return p[N];
}
45 }

```

5.7.4 Solve Linear Equation

```

1 typedef vector<double> vd;
const double eps = 1e-12;
3
// solves for x: A * x = b
5 int solveLinear(vector<vd> &A, vd &b, vd &x) {
    int n = sz(A), m = sz(x), rank = 0, br, bc;
    if (n) assert(sz(A[0]) == m);
    vi col(m);
    iota(all(col), 0);

    rep(i, 0, n) {
        double v, bv = 0;
        rep(r, i, n) rep(c, i, m) if ((v = fabs(A[r][c])) > bv)
            br = r, bc = c, bv = v;
    }
}
15

```

```

    if (bv <= eps) {
17         rep(j, i, n) if (fabs(b[j]) > eps) return -1;
        break;
19     }
    swap(A[i], A[br]);
21     swap(b[i], b[br]);
    swap(col[i], col[bc]);
23     rep(j, 0, n) swap(A[j][i], A[j][bc]);
    bv = 1 / A[i][i];
25     rep(j, i + 1, n) {
        double fac = A[j][i] * bv;
27         b[j] -= fac * b[i];
        rep(k, i + 1, m) A[j][k] -= fac * A[i][k];
29     }
    rank++;
31 }

33 x.assign(m, 0);
    for (int i = rank; i--;) {
35         b[i] /= A[i][i];
        x[col[i]] = b[i];
37         rep(j, 0, i) b[j] -= A[j][i] * b[i];
    }
39 return rank; // (multiple solutions if rank < m)
}

```

5.8 Polynomial Interpolation

```

1 // returns a, such that a[0]x^0 + a[1]x^1 + a[2]x^2 + ...
  // passes through the given points
3 typedef vector<double> vd;
  vd interpolate(vd x, vd y, int n) {
5     vd res(n), temp(n);
    rep(k, 0, n - 1) rep(i, k + 1, n) y[i] =
7     (y[i] - y[k]) / (x[i] - x[k]);
    double last = 0;
9     temp[0] = 1;
    rep(k, 0, n) rep(i, 0, n) {
11         res[i] += y[k] * temp[i];
        swap(last, temp[i]);
13         temp[i] -= last * x[k];
    }
15 return res;
}

```

5.9 Simplex Algorithm

```

1 // Two-phase simplex algorithm for solving linear programs
  // of the form
3 //
  //      maximize      c^T x
5 //      subject to    Ax <= b
  //                   x >= 0
7 //
  // INPUT: A -- an m x n matrix
9 //         b -- an m-dimensional vector
  //         c -- an n-dimensional vector
11 //         x -- a vector where the optimal solution will be
  //             stored
13 //
  // OUTPUT: value of the optimal solution (infinity if
15 // unbounded
  //         above, nan if infeasible)
17 //
  // To use this code, create an LPSolver object with A, b,
19 // and c as arguments. Then, call Solve(x).

21 typedef long double ld;
  typedef vector<ld> vd;
23 typedef vector<vd> vvd;
  typedef vector<int> vi;

25 const ld EPS = 1e-9;

27 struct LPSolver {
29     int m, n;
    vi B, N;
31     vvd D;

33     LPSolver(const vvd &A, const vd &b, const vd &c)
        : m(b.size()), n(c.size()), N(n + 1), B(m),
35         D(m + 2, vd(n + 2)) {
        for (int i = 0; i < m; i++)
37             for (int j = 0; j < n; j++) D[i][j] = A[i][j];
        for (int i = 0; i < m; i++) {
39             B[i] = n + i;
            D[i][n] = -1;
41             D[i][n + 1] = b[i];
        }
43         for (int j = 0; j < n; j++) {
            N[j] = j;

```

```

45         D[m][j] = -c[j];
        }
47         N[n] = -1;
        D[m + 1][n] = 1;
49     }

51     void Pivot(int r, int s) {
        double inv = 1.0 / D[r][s];
53         for (int i = 0; i < m + 2; i++)
            if (i != r)
55                 for (int j = 0; j < n + 2; j++)
                    if (j != s) D[i][j] -= D[r][j] * D[i][s] * inv;
57         for (int j = 0; j < n + 2; j++)
            if (j != s) D[r][j] *= inv;
59         for (int i = 0; i < m + 2; i++)
            if (i != r) D[i][s] *= -inv;
61         D[r][s] = inv;
        swap(B[r], N[s]);
63     }

65     bool Simplex(int phase) {
        int x = phase == 1 ? m + 1 : m;
67         while (true) {
            int s = -1;
69             for (int j = 0; j <= n; j++) {
                if (phase == 2 && N[j] == -1) continue;
71                 if (s == -1 || D[x][j] < D[x][s] ||
                    D[x][j] == D[x][s] && N[j] < N[s])
                    s = j;
            }
75             if (D[x][s] > -EPS) return true;
            int r = -1;
77             for (int i = 0; i < m; i++) {
                if (D[i][s] < EPS) continue;
79                 if (r == -1 ||
                    D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s]
81                 ||
                    (D[i][n + 1] / D[i][s]) ==
83                     (D[r][n + 1] / D[r][s]) &&
                    B[i] < B[r])
                    r = i;
85             }
            if (r == -1) return false;
87             Pivot(r, s);
89         }
    }

91     ld Solve(vd &x) {
        int r = 0;
93         for (int i = 1; i < m; i++)
            if (D[i][n + 1] < D[r][n + 1]) r = i;
95         if (D[r][n + 1] < -EPS) {
            Pivot(r, n);
97             if (!Simplex(1) || D[m + 1][n + 1] < -EPS)
                return numeric_limits<ld>::infinity();
99             for (int i = 0; i < m; i++)
                if (B[i] == -1) {
101                 int s = -1;
                for (int j = 0; j <= n; j++)
103                     if (s == -1 || D[i][j] < D[i][s] ||
                        D[i][j] == D[i][s] && N[j] < N[s])
                            s = j;
105                 Pivot(i, s);
107             }
        }
109         if (!Simplex(2)) return numeric_limits<ld>::infinity();
        x = vd(n);
111         for (int i = 0; i < m; i++)
            if (B[i] < n) x[B[i]] = D[i][n + 1];
113         return D[m][n + 1];
115     };

117     int main() {
119         const int m = 4;
        const int n = 3;
121         ld _A[m][n] = {
            {6, -1, 0}, {-1, -5, 0}, {1, 5, 1}, {-1, -5, -1}};
123         ld _b[m] = {10, -4, 5, -5};
        ld _c[n] = {1, -1, 0};

125         vvd A(m);
        vd b(_b, _b + m);
127         vd c(_c, _c + n);
        for (int i = 0; i < m; i++) A[i] = vd(_A[i], _A[i] + n);
129         LPSolver solver(A, b, c);
        vd x;
131         ld value = solver.Solve(x);
133     }
}

```



```

135 cerr << "VALUE: " << value << endl; // VALUE: 1.29032
    cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
137 for (size_t i = 0; i < x.size(); i++) cerr << " " <<
    x[i];
    cerr << endl;
139 return 0;
}

```

6 Geometry

6.1 Point

```

1 template <typename T> struct P {
    T x, y;
    P(T x = 0, T y = 0) : x(x), y(y) {}
    bool operator<(const P &p) const {
        return tie(x, y) < tie(p.x, p.y);
    }
    bool operator==(const P &p) const {
        return tie(x, y) == tie(p.x, p.y);
    }
    P operator-() const { return {-x, -y}; }
    P operator+(P p) const { return {x + p.x, y + p.y}; }
    P operator-(P p) const { return {x - p.x, y - p.y}; }
    P operator*(T d) const { return {x * d, y * d}; }
    P operator/(T d) const { return {x / d, y / d}; }
    T dist2() const { return x * x + y * y; }
    double len() const { return sqrt(dist2()); }
    P unit() const { return *this / len(); }
    friend T dot(P a, P b) { return a.x * b.x + a.y * b.y; }
    friend T cross(P a, P b) { return a.x * b.y - a.y *
        b.x; }
    friend T cross(P a, P b, P o) {
        return cross(a - o, b - o);
    }
23 };
using pt = P<ll>;

```

6.1.1 Quaternion

```

1 constexpr double PI = 3.141592653589793;
constexpr double EPS = 1e-7;
2 struct Q {
    using T = double;
    T x, y, z, r;
    Q(T r = 0) : x(0), y(0), z(0), r(r) {}
    Q(T x, T y, T z, T r = 0) : x(x), y(y), z(z), r(r) {}
    friend bool operator==(const Q &a, const Q &b) {
        return (a - b).abs2() <= EPS;
    }
    friend bool operator!=(const Q &a, const Q &b) {
        return !(a == b);
    }
    Q operator-() { return Q(-x, -y, -z, -r); }
    Q operator+(const Q &b) const {
        return Q(x + b.x, y + b.y, z + b.z, r + b.r);
    }
    Q operator-(const Q &b) const {
        return Q(x - b.x, y - b.y, z - b.z, r - b.r);
    }
    Q operator*(const T &t) const {
        return Q(x * t, y * t, z * t, r * t);
    }
    Q operator*(const Q &b) const {
        return Q(r * b.x + x * b.r + y * b.z - z * b.y,
            r * b.y - x * b.z + y * b.r + z * b.x,
            r * b.z + x * b.y - y * b.x + z * b.r,
            r * b.r - x * b.x - y * b.y - z * b.z);
    }
    Q operator/(const Q &b) const { return *this * b.inv(); }
    T abs2() const { return r * r + x * x + y * y + z * z; }
    T len() const { return sqrt(abs2()); }
    Q conj() const { return Q(-x, -y, -z, r); }
    Q unit() const { return *this * (1.0 / len()); }
    Q inv() const { return conj() * (1.0 / abs2()); }
    friend T dot(Q a, Q b) {
        return a.x * b.x + a.y * b.y + a.z * b.z;
    }
    friend Q cross(Q a, Q b) {
        return Q(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z,
            a.x * b.y - a.y * b.x);
    }
    friend Q rotation_around(Q axis, T angle) {
        return axis.unit() * sin(angle / 2) + cos(angle / 2);
    }
    Q rotated_around(Q axis, T angle) {
        Q u = rotation_around(axis, angle);
        return u * *this / u;
    }
    friend Q rotation_between(Q a, Q b) {

```

```

51 a = a.unit(), b = b.unit();
    if (a == -b) {
53         // degenerate case
        Q ortho = abs(a.y) > EPS ? cross(a, Q(1, 0, 0))
            : cross(a, Q(0, 1, 0));
55         return rotation_around(ortho, PI);
    }
57     return (a * (a + b)).conj();
59 }
};

```

6.1.2 Spherical Coordinates

```

1 struct car_p {
    double x, y, z;
    };
    struct sph_p {
    double r, theta, phi;
    };
    sph_p conv(car_p p) {
    double r = sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
    double theta = asin(p.y / r);
    double phi = atan2(p.y, p.x);
    return {r, theta, phi};
    }
    car_p conv(sph_p p) {
    double x = p.r * cos(p.theta) * sin(p.phi);
    double y = p.r * cos(p.theta) * cos(p.phi);
    double z = p.r * sin(p.theta);
    return {x, y, z};
    }
19 }

```

6.2 Segments

```

1 // for non-collinear ABCD, if segments AB and CD intersect
bool intersects(pt a, pt b, pt c, pt d) {
3     if (cross(b, c, a) * cross(b, d, a) > 0) return false;
        if (cross(d, a, c) * cross(d, b, c) > 0) return false;
5     return true;
}
// the intersection point of lines AB and CD
pt intersect(pt a, pt b, pt c, pt d) {
9     auto x = cross(b, c, a), y = cross(b, d, a);
        if (x == y) {
11         // if(abs(x, y) < 1e-8) {
            // is parallel
13         } else {
            return d * (x / (x - y)) - c * (y / (x - y));
15         }
    }
}

```

6.3 Convex Hull

```

1 // returns a convex hull in counterclockwise order
// for a non-strict one, change cross >= to >
3 vector<pt> convex_hull(vector<pt> p) {
    sort(ALL(p));
5     if (p[0] == p.back()) return {p[0]};
        int n = p.size(), t = 0;
        vector<pt> h(n + 1);
        for (int _ = 2, s = 0; _--; s = --t, reverse(ALL(p)))
9             for (pt i : p) {
                while (t > s + 1 && cross(i, h[t - 1], h[t - 2]) >=
0)
11                 t--;
                    h[t++] = i;
13             }
                return h.resize(t), h;
15 }

```

6.3.1 3D Hull

```

1 typedef Point3D<double> P3;
2
3 struct PR {
    void ins(int x) { (a == -1 ? a : b) = x; }
    void rem(int x) { (a == x ? a : b) = -1; }
    int cnt() { return (a != -1) + (b != -1); }
    int a, b;
    };
    struct F {
    int a, b, c;
    };
    // collinear points will kill it, please remove before use
    // skip between -snip- comments if no 4 coplanar points
    vector<F> hull3d(vector<P3> A) {
17     int n = A.size(), t2 = 2, t3 = 3;
        vector<vector<PR>> E(n, vector<PR>(n, {-1, -1}));

```

```

19 vector<F> FS;
21 for (int i = 2; i < n; i++) // -snip-
    for (int j = i + 1; j < n; j++) {
23         ll v = cross(A[0], A[1], A[i]).dot(A[j] - A[0]);
25         if (v != 0) {
27             if (v < 0) swap(i, j);
29             swap(A[2], A[t2 = i]), swap(A[3], A[t3 = j]);
31             goto ok;
33         }
35         assert(!"all coplanar");
37     }
38 ok; // -snip-
39
40 #define E(x, y) E[min(f.x, f.y)][max(f.x, f.y)]
41 #define C(a, b)
42 if (E(a, b).cnt() != 2) mf(f.a, f.b, i);
43
44 auto mf = [&](int i, int j, int k) {
45     F f = {i, j, k};
46     E(a, b).ins(k);
47     E(a, c).ins(j);
48     E(b, c).ins(i);
49     FS.push_back(f);
50 };
51
52 auto in = [&](int i, int j, int k, int l) {
53     P3 a = cross(A[i], A[j], A[l]),
54         b = cross(A[j], A[k], A[l]),
55         c = cross(A[k], A[i], A[l]);
56     return a.dot(b) > 0 && b.dot(c) > 0;
57 };
58 mf(0, 2, 1), mf(0, 1, 3), mf(1, 2, 3), mf(0, 3, 2);
59
60 for (int i = 4; i < n; i++) {
61     for (int j = 0; j < FS.size(); j++) {
62         F f = FS[j];
63         ll d =
64             cross(A[f.a], A[f.b], A[f.c]).dot(A[i] - A[f.a]);
65         if (d > 0 || (d == 0 && in(f.a, f.b, f.c, i))) {
66             E(a, b).rem(f.c);
67             E(a, c).rem(f.b);
68             E(b, c).rem(f.a);
69             swap(FS[j--], FS.back());
70             FS.pop_back();
71         }
72     }
73     for (int j = 0, s = FS.size(); j < s; j++) {
74         F f = FS[j];
75         C(c, b);
76         C(b, a);
77         C(a, c);
78     }
79     vector<int> idx(n), ri(n); // -snip-
80     iota(idx.begin(), idx.end(), 0);
81     swap(idx[t3], idx[3]), swap(idx[t2], idx[2]);
82     for (int i = 0; i < n; i++) ri[idx[i]] = i;
83     for (auto &a, b, c : FS)
84         a = ri[a], b = ri[b], c = ri[c]; // -snip-
85     return FS;
86 };
87 #undef E
88 #undef C

```

6.4 Angular Sort

```

1 auto angle_cmp = [](const pt &a, const pt &b) {
2     auto btm = [](const pt &a) {
3         return a.y < 0 || (a.y == 0 && a.x < 0);
4     };
5     return make_tuple(btm(a), a.y * b.x, abs2(a)) <
6         make_tuple(btm(b), a.x * b.y, abs2(b));
7 };
8 void angular_sort(vector<pt> &p) {
9     sort(p.begin(), p.end(), angle_cmp);
10 }

```

6.5 Convex Hull Tangent

```

1 // before calling, do
2 // int top = max_element(c.begin(), c.end()) -
3 // c.begin();
4 // c.push_back(c[0]), c.push_back(c[1]);
5 pt left_tangent(const vector<pt> &c, int top, pt p) {
6     int n = c.size() - 2;
7     int ans = -1;
8     do {
9         if (cross(p, c[n], c[n + 1]) >= 0 &&
10             (cross(p, c[top + 1], c[n]) > 0 ||

```

```

11             cross(p, c[top], c[top + 1]) < 0))
12         break;
13     int l = top + 1, r = n + 1;
14     while (l < r - 1) {
15         int m = (l + r) / 2;
16         if (cross(p, c[m - 1], c[m]) > 0 &&
17             cross(p, c[top + 1], c[m]) > 0)
18             l = m;
19         else r = m;
20     }
21     ans = l;
22 } while (false);
23 do {
24     if (cross(p, c[top], c[top + 1]) >= 0 &&
25         (cross(p, c[1], c[top]) > 0 ||
26         cross(p, c[0], c[1]) < 0))
27         break;
28     int l = 1, r = top + 1;
29     while (l < r - 1) {
30         int m = (l + r) / 2;
31         if (cross(p, c[m - 1], c[m]) > 0 &&
32             cross(p, c[1], c[m]) > 0)
33             l = m;
34         else r = m;
35     }
36     ans = l;
37 } while (false);
38 return c[ans] - p;
39 }

```

6.6 Convex Polygon Minkowski Sum

```

1 // O(n) convex polygon minkowski sum
2 // must be sorted and counterclockwise
3 vector<pt> minkowski_sum(vector<pt> p, vector<pt> q) {
4     auto diff = [](vector<pt> &c) {
5         auto rcmp = [](pt a, pt b) {
6             return pt{a.y, a.x} < pt{b.y, b.x};
7         };
8         rotate(c.begin(), min_element(ALL(c), rcmp), c.end());
9         c.push_back(c[0]);
10        vector<pt> ret;
11        for (int i = 1; i < c.size(); i++)
12            ret.push_back(c[i] - c[i - 1]);
13        return ret;
14    };
15    auto dp = diff(p), dq = diff(q);
16    pt cur = p[0] + q[0];
17    vector<pt> d(dp.size() + dq.size(), ret = {cur});
18    // include angle_cmp from angular-sort.cpp
19    merge(ALL(dp), ALL(dq), d.begin(), angle_cmp);
20    // optional: make ret strictly convex (UB if degenerate)
21    int now = 0;
22    for (int i = 1; i < d.size(); i++) {
23        if (cross(d[i], d[now]) == 0) d[now] = d[now] + d[i];
24        else d[++now] = d[i];
25    }
26    d.resize(now + 1);
27    // end optional part
28    for (pt v : d) ret.push_back(cur = cur + v);
29    return ret.pop_back(), ret;
30 }

```

6.7 Point In Polygon

```

1 bool on_segment(pt a, pt b, pt p) {
2     return cross(a, b, p) == 0 && dot((p - a), (p - b)) <= 0;
3 }
4 // p can be any polygon, but this is O(n)
5 bool inside(const vector<pt> &p, pt a) {
6     int cnt = 0, n = p.size();
7     for (int i = 0; i < n; i++) {
8         pt l = p[i], r = p[(i + 1) % n];
9         // change to return 0; for strict version
10        if (on_segment(l, r, a)) return 1;
11        cnt ^= ((a.y < l.y) - (a.y < r.y)) * cross(l, r, a) >
12            0;
13    }
14    return cnt;
15 }

```

6.7.1 Convex Version

```

1 // no preprocessing version
2 // p must be a strict convex hull, counterclockwise
3 // if point is inside or on border
4 bool is_inside(const vector<pt> &c, pt p) {
5     int n = c.size(), l = 1, r = n - 1;
6     if (cross(c[0], c[1], p) < 0) return false;
7     if (cross(c[n - 1], c[0], p) < 0) return false;
8     while (l < r - 1) {

```

```

9   int m = (l + r) / 2;
   T a = cross(c[0], c[m], p);
11  if (a > 0) l = m;
   else if (a < 0) r = m;
13  else return dot(c[0] - p, c[m] - p) <= 0;
   }
15  if (l == r) return dot(c[0] - p, c[l] - p) <= 0;
   else return cross(c[l], c[r], p) >= 0;
17 }

19 // with preprocessing version
   vector<pt> vecs;
21 pt center;
   // p must be a strict convex hull, counterclockwise
23 // BEWARE OF OVERFLOWS!!
   void preprocess(vector<pt> p) {
25     for (auto &v : p) v = v * 3;
       center = p[0] + p[1] + p[2];
27     center.x /= 3, center.y /= 3;
       for (auto &v : p) v = v - center;
29     vecs = (angular_sort(p), p);
   }
31 bool intersect_strict(pt a, pt b, pt c, pt d) {
   if (cross(b, c, a) * cross(b, d, a) > 0) return false;
33  if (cross(d, a, c) * cross(d, b, c) >= 0) return false;
   return true;
35 }
   // if point is inside or on border
37 bool query(pt p) {
   p = p * 3 - center;
39   auto pr = upper_bound(ALL(vecs), p, angle_cmp);
   if (pr == vecs.end()) pr = vecs.begin();
41   auto pl = (pr == vecs.begin()) ? vecs.back() : *(pr - 1);
   return !intersect_strict({0, 0}, p, pl, *pr);
43 }

```

6.7.2 Offline Multiple Points Version

```

1 using Double = __float128;
   using Point = pt<Double, Double>;
3
   int n, m;
5 vector<Point> poly;
   vector<Point> query;
7 vector<int> ans;

9 struct Segment {
   Point a, b;
11  int id;
   };
13 vector<Segment> segs;

15 Double Xnow;
   inline Double get_y(const Segment &u, Double xnow = Xnow) {
17     const Point &a = u.a;
       const Point &b = u.b;
19     return (a.y * (b.x - xnow) + b.y * (xnow - a.x)) /
       (b.x - a.x);
21 }
   bool operator<(Segment u, Segment v) {
23     Double yu = get_y(u);
       Double yv = get_y(v);
25     if (yu != yv) return yu < yv;
       return u.id < v.id;
27 }
   ordered_map<Segment> st;

29
   struct Event {
31     int type; // +1 insert seg, -1 remove seg, 0 query
       Double x, y;
33     int id;
   };
35 bool operator<(Event a, Event b) {
   if (a.x != b.x) return a.x < b.x;
37  if (a.type != b.type) return a.type < b.type;
   return a.y < b.y;
39 }
   vector<Event> events;

41
   void solve() {
43     set<Double> xs;
       set<Point> ps;
45     for (int i = 0; i < n; i++) {
       xs.insert(poly[i].x);
47     ps.insert(poly[i]);
   }
49     for (int i = 0; i < n; i++) {
       Segment s{poly[i], poly[(i + 1) % n], i};
51     if (s.a.x > s.b.x ||
       (s.a.x == s.b.x && s.a.y > s.b.y)) {
53         swap(s.a, s.b);

```

```

   }
55     segs.push_back(s);

57     if (s.a.x != s.b.x) {
       events.push_back({+1, s.a.x + 0.2, s.a.y, i});
59     events.push_back({-1, s.b.x - 0.2, s.b.y, i});
   }
61 }
   for (int i = 0; i < m; i++) {
63     events.push_back({0, query[i].x, query[i].y, i});
   }
65 sort(events.begin(), events.end());
   int cnt = 0;
67 for (Event e : events) {
   int i = e.id;
69   Xnow = e.x;
   if (e.type == 0) {
71     Double x = e.x;
       Double y = e.y;
73     Segment tmp = {{x - 1, y}, {x + 1, y}, -1};
       auto it = st.lower_bound(tmp);
75
       if (ps.count(query[i]) > 0) {
77         ans[i] = 0;
       } else if (xs.count(x) > 0) {
79         ans[i] = -2;
       } else if (it != st.end() &&
81             get_y(*it) == get_y(tmp)) {
           ans[i] = 0;
83       } else if (it != st.begin() &&
           get_y(*prev(it)) == get_y(tmp)) {
85         ans[i] = 0;
       } else {
87         int rk = st.order_of_key(tmp);
           if (rk % 2 == 1) {
89             ans[i] = 1;
           } else {
91             ans[i] = -1;
           }
       }
93     }
   } else if (e.type == 1) {
95     st.insert(segs[i]);
       assert((int)st.size() == ++cnt);
97   } else if (e.type == -1) {
       st.erase(segs[i]);
99     assert((int)st.size() == --cnt);
   }
101 }

```

6.8 Closest Pair

```

1 vector<pll> p; // sort by x first!
   bool cmpy(const pll &a, const pll &b) const {
3     return a.y < b.y;
   }
5 ll sq(ll x) { return x * x; }
   // returns (minimum dist)^2 in [l, r)
7 ll solve(int l, int r) {
   if (r - l <= 1) return 1e18;
9   int m = (l + r) / 2;
   ll mid = p[m].x, d = min(solve(l, m), solve(m, r));
11  auto pb = p.begin();
   inplace_merge(pb + l, pb + m, pb + r, cmpy);
13  vector<pll> s;
   for (int i = l; i < r; i++)
15     if (sq(p[i].x - mid) < d) s.push_back(p[i]);
   for (int i = 0; i < s.size(); i++)
17     for (int j = i + 1;
       j < s.size() && sq(s[j].y - s[i].y) < d; j++)
19         d = min(d, dis(s[i], s[j]));
   return d;
21 }

```

6.9 Minimum Enclosing Circle

```

1 typedef Point<double> P;
   double ccRadius(const P &A, const P &B, const P &C) {
3     return (B - A).dist() * (C - B).dist() * (A - C).dist() /
       abs((B - A).cross(C - A)) / 2;
5 }
   P ccCenter(const P &A, const P &B, const P &C) {
7     P b = C - A, c = B - A;
       return A + (b * c.dist2() - c * b.dist2()).perp() /
9     b.cross(c) / 2;
   }
11 pair<P, double> mec(vector<P> ps) {
   shuffle(all(ps), mt19937(time(0)));
13   P o = ps[0];
   double r = 0, EPS = 1 + 1e-8;
15   rep(i, 0, sz(ps)) if ((o - ps[i]).dist() > r * EPS) {

```

```

    o = ps[i], r = 0;
17 rep(j, 0, i) if ((o - ps[j]).dist() > r * EPS) {
    o = (ps[i] + ps[j]) / 2;
19 r = (o - ps[i]).dist();
    rep(k, 0, j) if ((o - ps[k]).dist() > r * EPS) {
21 o = ccCenter(ps[i], ps[j], ps[k]);
    r = (o - ps[i]).dist();
23 }
25 }
    return {o, r};
27 }

```

6.10 Delaunay Triangulation

```

1 // O(n * log(n)), T_large must be able to hold O(T^4) (can
  // be long long if coord <= 2e4)
3 struct quad_edge {
    int o = -1; // origin of the arc
5 quad_edge *onext, *rot;
    bool mark = false;
7 quad_edge() {}
    quad_edge(int o) : o(o) {}
9 int d() { return sym()->o; } // destination of the arc
    quad_edge *sym() { return rot->rot; }
11 quad_edge *oprev() { return rot->onext->rot; }
    quad_edge *lnext() { return sym()->oprev(); }
13 static quad_edge *make_sphere(int a, int b) {
    array<quad_edge *, 4> q{
15 {new quad_edge{a}, new quad_edge{b}, new quad_edge{b},
    new quad_edge{a}};
17 for (auto i = 0; i < 4; ++i)
    q[i]->onext = q[-i & 3], q[i]->rot = q[i + 1 & 3];
19 return q[0];
    }
21 static void splice(quad_edge *a, quad_edge *b) {
    swap(a->onext->rot->onext, b->onext->rot->onext);
23 swap(a->onext, b->onext);
    }
25 static quad_edge *connect(quad_edge *a, quad_edge *b) {
    quad_edge *q = make_sphere(a->d(), b->o);
27 splice(q, a->lnext()), splice(q->sym(), b);
    return q;
29 }
};
31 template <class T, class T_large, class F1, class F2>
bool delaunay_triangulation(const vector<point<T>> &a,
33 F1 process_outer_face,
    F2 process_triangles) {
35 vector<int> ind(a.size());
    iota(ind.begin(), ind.end(), 0);
37 sort(ind.begin(), ind.end(),
    [&](int i, int j) { return a[i] < a[j]; });
39 ind.erase(
    unique(ind.begin(), ind.end(),
41 [&](int i, int j) { return a[i] == a[j]; }),
    ind.end());
43 int n = (int)ind.size();
    if (n < 2) return {};
45 auto circular = [&](point<T> p, point<T> a, point<T> b,
    point<T> c) {
47 a -= p, b -= p, c -= p;
    return ((T_large)a.squared_norm() * (b ^ c) +
49 (T_large)b.squared_norm() * (c ^ a) +
    (T_large)c.squared_norm() * (a ^ b)) *
51 (doubled_signed_area(a, b, c) > 0 ? 1 : -1) >
    0;
53 };
    auto recurse = [&](auto self, int l,
55 int r) -> array<quad_edge *, 2> {
    if (r - l <= 3) {
57 quad_edge *p =
    quad_edge::make_sphere(ind[l], ind[l + 1]);
59 if (r - l == 2) return {p, p->sym()};
    quad_edge *q =
61 quad_edge::make_sphere(ind[l + 1], ind[l + 2]);
    quad_edge::splice(p->sym(), q);
63 auto side = doubled_signed_area(
    a[ind[l]], a[ind[l + 1]], a[ind[l + 2]]);
65 quad_edge *c = side ? quad_edge::connect(q, p) : NULL;
    return {side < 0 ? c->sym() : p,
67 side < 0 ? c : q->sym()};
    }
69 int m = l + (r - l >> 1);
    auto [ra, A] = self(self, l, m);
71 auto [B, rb] = self(self, m, r);
    while (
73 doubled_signed_area(a[B->o], a[A->d()], a[A->o]) < 0 &&
    (A = A->lnext()) ||
75 doubled_signed_area(a[A->o], a[B->d()], a[B->o]) > 0 &&
    (B = B->sym()->onext))

```

```

77 ;
    quad_edge *base = quad_edge::connect(B->sym(), A);
79 if (A->o == ra->o) ra = base->sym();
    if (B->o == rb->o) rb = base;
81 #define valid(e) \
    (doubled_signed_area(a[e->d()], a[base->d()], \
    \
    a[base->o]) > 0)
83 #define DEL(e, init, dir) \
85 quad_edge *e = init->dir; \
    if (valid(e)) \
87 while (circular(a[e->dir->d()], a[base->d()], \
    \
    a[base->o], a[e->d()])) { \
89 quad_edge *t = e->dir; \
    quad_edge::splice(e, e->oprev()); \
91 quad_edge::splice(e->sym(), e->sym()->oprev()); \
    \
    delete e->rot->rot->rot; \
93 delete e->rot->rot; \
    delete e->rot; \
95 delete e; \
    e = t; \
    }
97 while (true) {
99 DEL(LC, base->sym(), onext);
    DEL(RC, base, oprev());
101 if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) ||
103 valid(RC) && circular(a[RC->d()], a[RC->o],
    a[LC->d()], a[LC->o]))
105 base = quad_edge::connect(RC, base->sym());
    else
107 base = quad_edge::connect(base->sym(), LC->sym());
    }
109 return {ra, rb};
};
111 auto e = recurse(recurse, 0, n)[0];
    vector<quad_edge *> q = {e, rem};
113 while (doubled_signed_area(a[e->onext->d()], a[e->d()],
    a[e->o]) < 0)
115 e = e->onext;
    vector<int> face;
117 face.reserve(n);
    bool colinear = false;
119 #define ADD \
    { \
121 quad_edge *c = e; \
    face.clear(); \
123 do { \
    c->mark = true; \
125 face.push_back(c->o); \
    q.push_back(c->sym()); \
127 rem.push_back(c); \
    c = c->lnext(); \
129 } while (c != e); \
    }
131 ADD;
    process_outer_face(face);
133 for (auto qi = 0; qi < (int)q.size(); ++qi) {
    if (!(e = q[qi])->mark) {
135 ADD;
    colinear = false;
137 process_triangles(face[0], face[1], face[2]);
    }
139 for (auto e : rem) delete e->rot, delete e;
141 return !colinear;
    }

```

6.10.1 Quadratic Time Version

```

1 template <class P, class F>
void delaunay(vector<P> &ps, F trifun) {
3 if (sz(ps) == 3) {
    int d = (ps[0].cross(ps[1], ps[2]) < 0);
5 trifun(0, 1 + d, 2 - d);
    }
7 vector<P3> p3;
    for (P p : ps) p3.emplace_back(p.x, p.y, p.dist2());
9 if (sz(ps) > 3)
    for (auto t : hull3d(p3))
11 if ((p3[t.b] - p3[t.a])
    .cross(p3[t.c] - p3[t.a])
13 .dot(P3(0, 0, 1)) < 0)
    trifun(t.a, t.c, t.b);
15 }

```

6.11 Half Plane Intersection


```

1 struct Line {
2     Point P;
3     Vector v;
4     bool operator<(const Line &b) const {
5         return atan2(v.y, v.x) < atan2(b.v.y, b.v.x);
6     }
7 };
8 bool OnLeft(const Line &L, const Point &p) {
9     return Cross(L.v, p - L.P) > 0;
10 }
11 Point GetIntersection(Line a, Line b) {
12     Vector u = a.P - b.P;
13     Double t = Cross(b.v, u) / Cross(a.v, b.v);
14     return a.P + a.v * t;
15 }
16 int HalfplaneIntersection(Line *L, int n, Point *poly) {
17     sort(L, L + n);
18
19     int first, last;
20     Point *p = new Point[n];
21     Line *q = new Line[n];
22     q[first = last = 0] = L[0];
23     for (int i = 1; i < n; i++) {
24         while (first < last && !OnLeft(L[i], p[last - 1]))
25             last--;
26         while (first < last && !OnLeft(L[i], p[first])) first++;
27         q[++last] = L[i];
28         if (fabs(Cross(q[last].v, q[last - 1].v)) < EPS) {
29             last--;
30             if (OnLeft(q[last], L[i].P)) q[last] = L[i];
31             if (first < last)
32                 p[last - 1] = GetIntersection(q[last - 1], q[last]);
33         }
34         while (first < last && !OnLeft(q[first], p[last - 1]))
35             last--;
36         if (last - first <= 1) return 0;
37         p[last] = GetIntersection(q[last], q[first]);
38
39         int m = 0;
40         for (int i = first; i <= last; i++) poly[m++] = p[i];
41         return m;
42     }
43 }
44
45 void build_fail(int ptr) {
46     int tmp;
47     for (int i = 0; i < maxc; i++)
48         if (T[ptr].Next[i]) {
49             tmp = T[ptr].fail;
50             while (tmp != 1 && !T[tmp].Next[i])
51                 tmp = T[tmp].fail;
52             if (T[tmp].Next[i] != T[ptr].Next[i])
53                 if (T[tmp].Next[i]) tmp = T[tmp].Next[i];
54             T[T[ptr].Next[i]].fail = tmp;
55             q[qtot++] = T[ptr].Next[i];
56         }
57 }
58 void AC_auto(const string &s) {
59     int ptr = 1;
60     for (char c : s) {
61         while (ptr != 1 && !T[ptr].Next[c]) ptr =
62             T[ptr].fail;
63         if (T[ptr].Next[c]) {
64             ptr = T[ptr].Next[c];
65             T[ptr].ans++;
66         }
67     }
68 }
69 void Solve(string &s) {
70     for (char &c : s) // change char id
71         c -= 'a';
72     for (int i = 0; i < qtop; i++) build_fail(q[i]);
73     AC_auto(s);
74     for (int i = qtop - 1; i > -1; i--)
75         T[T[q[i]].fail].ans += T[q[i]].ans;
76 }
77 void reset() {
78     qtop = top = q[0] = 1;
79     get_node(1);
80 }
81 } AC;
82 // usage example
83 string s, S;
84 int n, t, ans_place[50000];
85 int main() {
86     Tie cin >> t;
87     while (t--) {
88         AC.reset();
89         cin >> S >> n;
90         for (int i = 0; i < n; i++) {
91             cin >> s;
92             ans_place[i] = AC.insert(s);
93         }
94         AC.Solve(S);
95         for (int i = 0; i < n; i++)
96             cout << AC.T[ans_place[i]].ans << '\n';
97     }
98 }

```

7 Strings

7.1 Knuth-Morris-Pratt Algorithm

```

1 vector<int> pi(const string &s) {
2     vector<int> p(s.size());
3     for (int i = 1; i < s.size(); i++) {
4         int g = p[i - 1];
5         while (g && s[i] != s[g]) g = p[g - 1];
6         p[i] = g + (s[i] == s[g]);
7     }
8     return p;
9 }
10 vector<int> match(const string &s, const string &pat) {
11     vector<int> p = pi(pat + '\0' + s), res;
12     for (int i = p.size() - s.size(); i < p.size(); i++)
13         if (p[i] == pat.size())
14             res.push_back(i - 2 * pat.size());
15     return res;
16 }

```

7.2 Aho-Corasick Automaton

```

1 struct Aho_Corasick {
2     static const int maxc = 26, maxn = 4e5;
3     struct NODES {
4         int Next[maxc], fail, ans;
5     };
6     NODES T[maxn];
7     int top, qtop, q[maxn];
8     int get_node(const int &fail) {
9         fill_n(T[top].Next, maxc, 0);
10         T[top].fail = fail;
11         T[top].ans = 0;
12         return top++;
13     }
14     int insert(const string &s) {
15         int ptr = 1;
16         for (char c : s) { // change char id
17             c -= 'a';
18             if (!T[ptr].Next[c]) T[ptr].Next[c] = get_node(ptr);
19             ptr = T[ptr].Next[c];
20         }
21         return ptr;
22     } // return ans_last_place
23 }

```

7.3 Suffix Array

```

1 // sa[i]: starting index of suffix at rank i
2 // 0-indexed, sa[0] = n (empty string)
3 // lcp[i]: lcp of sa[i] and sa[i - 1], lcp[0] = 0
4 struct SuffixArray {
5     vector<int> sa, lcp;
6     SuffixArray(string &s,
7         int lim = 256) { // or basic_string<int>
8         int n = sz(s) + 1, k = 0, a, b;
9         vector<int> x(all(s) + 1), y(n), ws(max(n, lim)),
10             rank(n);
11         sa = lcp = y, iota(all(sa), 0);
12         for (int j = 0, p = 0; p < n;
13             j = max(1, j * 2), lim = p) {
14             p = j, iota(all(y), n - j);
15             for (int i = 0; i < n; i++)
16                 if (sa[i] >= j) y[p++] = sa[i] - j;
17             fill(all(ws), 0);
18             for (int i = 0; i < n; i++) ws[x[i]]++;
19             for (int i = 1; i < lim; i++) ws[i] += ws[i - 1];
20             for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
21             swap(x, y), p = 1, x[sa[0]] = 0;
22             for (int i = 1; i < n; i++)
23                 a = sa[i - 1], b = sa[i],
24                 // clang-format off
25                 x[b] = (y[a] == y[b] && y[a + j] == y[b + j])
26                     ? p - 1 : p++;
27             // clang-format on
28         }
29         for (int i = 1; i < n; i++) rank[sa[i]] = i;
30         for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
31             for (k && k--, j = sa[rank[i] - 1];
32                 s[i + k] == s[j + k]; k++)
33                 ;
34     }
35 }

```



```

}
35 };

7.4 Suffix Tree

1 struct SAM {
    static const int maxc = 26; // char range
    static const int maxn = 10010; // string len
    struct Node {
        Node *green, *edge[maxc];
        int max_len, in, times;
    } *root, *last, reg[maxn * 2];
    int top;
    Node *get_node(int _max) {
        Node *re = &reg[top++];
        re->in = 0, re->times = 1;
        re->max_len = _max, re->green = 0;
        for (int i = 0; i < maxc; i++) re->edge[i] = 0;
        return re;
    }
    void insert(const char c) { // c in range [0, maxc)
        Node *p = last;
        last = get_node(p->max_len + 1);
        while (p && !p->edge[c])
            p->edge[c] = last, p = p->green;
        if (!p) last->green = root;
        else {
            Node *pot_green = p->edge[c];
            if ((pot_green->max_len) == (p->max_len + 1))
                last->green = pot_green;
            else {
                Node *wish = get_node(p->max_len + 1);
                wish->times = 0;
                while (p && p->edge[c] == pot_green)
                    p->edge[c] = wish, p = p->green;
                for (int i = 0; i < maxc; i++)
                    wish->edge[i] = pot_green->edge[i];
                wish->green = pot_green->green;
                pot_green->green = wish;
                last->green = wish;
            }
        }
    }
    Node *q[maxn * 2];
    int ql, qr;
    void get_times(Node *p) {
        ql = 0, qr = -1, reg[0].in = 1;
        for (int i = 1; i < top; i++) reg[i].green->in++;
        for (int i = 0; i < top; i++)
            if (!reg[i].in) q[++qr] = &reg[i];
        while (ql <= qr) {
            q[ql]->green->times += q[ql]->times;
            if (!(--q[ql]->green->in)) q[++qr] = q[ql]->green;
            ql++;
        }
    }
    void build(const string &s) {
        top = 0;
        root = last = get_node(0);
        for (char c : s) insert(c - 'a'); // change char id
        get_times(root);
        // call build before solve
        int solve(const string &s) {
            Node *p = root;
            for (char c : s)
                if (!(p = p->edge[c - 'a'])) // change char id
                    return 0;
            return p->times;
        }
    }
};

```

7.5 Cocke-Younger-Kasami Algorithm

```

1 struct rule {
    // s -> xy
    // if y == -1, then s -> x (unit rule)
    int s, x, y, cost;
};
int state;
// state (id) for each letter (variable)
// lowercase letters are terminal symbols
9 map<char, int> rules;
vector<rule> cnf;
11 void init() {
    state = 0;
    rules.clear();
    cnf.clear();
15 }
// convert a cfg rule to cnf (but with unit rules) and add
17 // it

```

```

void add_to_cnf(char s, const string &p, int cost) {
19 if (!rules.count(s)) rules[s] = state++;
    for (char c : p)
        if (!rules.count(c)) rules[c] = state++;
    if (p.size() == 1) {
        cnf.push_back({rules[s], rules[p[0]], -1, cost});
    } else {
        // length >= 3 -> split
        int left = rules[s];
        int sz = p.size();
        for (int i = 0; i < sz - 2; i++) {
            cnf.push_back({left, rules[p[i]], state, 0});
            left = state++;
        }
        cnf.push_back(
            {left, rules[p[sz - 2]], rules[p[sz - 1]], cost});
    }
35 }

37 constexpr int MAXN = 55;
    vector<long long> dp[MAXN][MAXN];
    // unit rules with negative costs can cause negative cycles
    vector<bool> neg_INF[MAXN][MAXN];
41 void relax(int l, int r, rule c, long long cost,
    bool neg_c = 0) {
    if (!neg_INF[l][r][c.s] &&
        (neg_INF[l][r][c.x] || cost < dp[l][r][c.s])) {
        if (neg_c || neg_INF[l][r][c.x]) {
            dp[l][r][c.s] = 0;
            neg_INF[l][r][c.s] = true;
        } else {
            dp[l][r][c.s] = cost;
        }
    }
51 }

53 void bellman(int l, int r, int n) {
    for (int k = 1; k <= state; k++)
        for (rule c : cnf)
            if (c.y == -1)
                relax(l, r, c, dp[l][r][c.x] + c.cost, k == n);
59 }

61 void cyk(const string &s) {
    vector<int> tok;
    for (char c : s) tok.push_back(rules[c]);
    for (int i = 0; i < tok.size(); i++) {
        for (int j = 0; j < tok.size(); j++) {
            dp[i][j] = vector<long long>(state + 1, INT_MAX);
            neg_INF[i][j] = vector<bool>(state + 1, false);
        }
        dp[i][i][tok[i]] = 0;
        bellman(i, i, tok.size());
    }
    for (int r = 1; r < tok.size(); r++) {
        for (int l = r - 1; l >= 0; l--) {
            for (int k = l; k < r; k++)
                for (rule c : cnf)
                    if (c.y != -1)
                        relax(l, r, c,
                            dp[l][k][c.x] + dp[k + 1][r][c.y] +
                            c.cost);
        }
        bellman(l, r, tok.size());
    }
81 }

83 // usage example
85 int main() {
    init();
    add_to_cnf('S', "aSc", 1);
    add_to_cnf('S', "BBB", 1);
    add_to_cnf('S', "SB", 1);
    add_to_cnf('B', "b", 1);
    cyk("abbbbc");
    // dp[0][s.size() - 1][rules[start]] = min cost to
    // generate s
    cout << dp[0][5][rules['S']] << '\n'; // 7
    cyk("acbc");
    cout << dp[0][3][rules['S']] << '\n'; // INT_MAX
    add_to_cnf('S', "S", -1);
    cyk("abbbbc");
    cout << neg_INF[0][5][rules['S']] << '\n'; // 1
}

```

7.6 Z Value

```

1 int z[n];
    void zval(string s) {
        // z[i] => longest common prefix of s and s[i:], i > 0
        int n = s.size();
        z[0] = 0;

```

```

    for (int b = 0, i = 1; i < n; i++) {
        if (z[b] + b <= i) z[i] = 0;
        else z[i] = min(z[i - b], z[b] + b - i);
        while (s[i + z[i]] == s[z[i]]) z[i]++;
        if (i + z[i] > b + z[b]) b = i;
    }
}

St[i->fail].cnt += i->cnt;
}
}
inline int size() { // The number of diff. pal.
    return SZ(St) - 2;
}
};

```

7.7 Manacher's Algorithm

```

1 int z[n];
void manacher(string s) {
3 // z[i] => longest odd palindrome centered at s[i] is
  // s[(i-z[i])..(i+z[i])]
5 // to get all palindromes (including even length),
  // insert a '#' between each s[i] and s[i+1]
7 // after that s[i..=j] is palindrome iff z[i+j] >= j-i
  int n = s.size();
9 z[0] = 0;
  for (int b = 0, i = 1; i < n; i++) {
11 if (z[b] + b >= i)
    z[i] = min(z[2 * b - i], b + z[b] - i);
13 else z[i] = 0;
    while (i + z[i] + 1 < n && i - z[i] - 1 >= 0 &&
15 s[i + z[i] + 1] == s[i - z[i] - 1])
      z[i]++;
17 if (z[i] + i > z[b] + b) b = i;
  }
19 }

```

7.8 Lyndon Factorization

```

1 vector<string> duval(string s) {
  // s += s for min rotation
3 int n = s.size(), i = 0, ans;
  vector<string> res;
5 while (i < n) { // change to i < n / 2 for min rotation
  ans = i;
7 int j = i + 1, k = i;
  for (; j < n && s[k] <= s[j]; j++)
9 k = s[k] < s[j] ? i : k + 1;
  while (i <= k) {
11 res.push_back(s.substr(i, j - k));
    i += j - k;
13 }
  }
15 // min rotation is s.substr(ans, n / 2)
  return res;
17 }

```

7.9 Palindromic Tree

```

1 struct palindromic_tree {
  struct node {
3 int next[26], fail, len;
    int cnt,
5 num; // cnt: appear times, num: number of pal. suf.
    node(int l = 0) : fail(0), len(l), cnt(0), num(0) {
7 for (int i = 0; i < 26; ++i) next[i] = 0;
  }
9 };
  vector<node> St;
11 vector<char> s;
  int last, n;
13 palindromic_tree() : St(2), last(1), n(0) {
    St[0].fail = 1, St[1].len = -1, s.pb(-1);
15 }
  inline void clear() {
17 St.clear(), s.clear(), last = 1, n = 0;
    St.pb(0), St.pb(-1);
19 St[0].fail = 1, s.pb(-1);
  }
21 inline int get_fail(int x) {
    while (s[n - St[x].len - 1] != s[n]) x = St[x].fail;
23 return x;
  }
25 inline void add(int c) {
    s.push_back(c - 'a'), ++n;
27 int cur = get_fail(last);
    if (!St[cur].next[c]) {
29 int now = SZ(St);
    St.pb(St[cur].len + 2);
31 St[now].fail = St[get_fail(St[cur].fail)].next[c];
    St[cur].next[c] = now;
33 St[now].num = St[St[now].fail].num + 1;
  }
35 last = St[cur].next[c], ++St[last].cnt;
}
37 inline void count() { // counting cnt
  auto i = St.rbegin();
39 for (; i != St.rend(); ++i) {

```