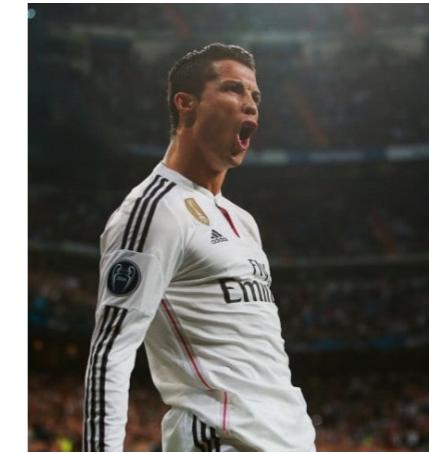


# El Bicho

DondeEstasCR7



19/02/2026

## Índice

<b>1. Algos</b>	<b>3</b>
1.1. Binary Search . . . . .	3
1.2. Fast Io . . . . .	3
1.3. Sliding Window . . . . .	4
1.4. Tablas Y Cotas . . . . .	5
1.5. Two Pointers . . . . .	7
<b>2. Bit Manipulation</b>	<b>8</b>
2.1. Bits . . . . .	8
<b>3. Combinatory</b>	<b>9</b>
3.1. Combi Brute Sin Mod . . . . .	9
3.2. Combinatory . . . . .	9
<b>4. Data Structures</b>	<b>10</b>
4.1. Fenwick Tree . . . . .	10
4.2. Find Two Numbers . . . . .	11
4.3. Segment Tree . . . . .	11
4.4. Sparse Table . . . . .	12
<b>5. Dp</b>	<b>12</b>
5.1. Digit Dp Pattern . . . . .	12
5.2. Digit Dp . . . . .	13
5.3. Edit Distance . . . . .	14
5.4. Knapsack . . . . .	14
5.5. Lcs . . . . .	14
<b>6. Geometry</b>	<b>15</b>
6.1. 2D Geometry . . . . .	15
6.2. Convex Hull . . . . .	22
<b>7. Graph</b>	<b>23</b>
7.1. Bellman Ford . . . . .	23
7.2. Bfs . . . . .	23
7.3. Bipartite . . . . .	24

7.4. Dfs . . . . .	24
7.5. Dfs 2D . . . . .	25
7.6. Dijkstra . . . . .	25
7.7. Disjoint Set Union Dsu . . . . .	26
7.8. Floyd Warshall . . . . .	26
7.9. Kruskal . . . . .	26
7.10. Lowest Common Ancestor Lca . . . . .	27
7.11. Prim . . . . .	28
7.12. Scc . . . . .	29
7.13. Topo Sort Dfs . . . . .	30
7.14. Topo Sort Kahns Bfs . . . . .	30
<b>8. Manhattan Distance</b>	<b>31</b>
8.1. Farthest Pair Of Points . . . . .	31
8.2. Nearest Neighbor In Each Octant . . . . .	31
<b>9. Number Theory</b>	<b>32</b>
9.1. Euler Toliente . . . . .	32
9.2. Gcd Lcm . . . . .	32
9.3. Number Theory . . . . .	33
9.4. Phi Euler . . . . .	33
9.5. Potenciacion Binaria . . . . .	34
9.6. Sieve . . . . .	34
9.7. Sieve Bitset . . . . .	34
9.8. Sum Of Divisors . . . . .	35
<b>10. Numerical Methods</b>	<b>35</b>
10.1. Sumas Notables . . . . .	35
10.2. Determinant Of A Matrix . . . . .	36
10.3. Linear Equations . . . . .	37
10.4. Modular Slae . . . . .	38
10.5. Simpson'S Integration . . . . .	38
<b>11. String</b>	<b>38</b>
11.1. Kmp . . . . .	38
11.2. Trie . . . . .	39
11.3. Z Algorithm . . . . .	40

## 1. Algos

### 1.1. Binary Search

```
1 // binary search en array ordenado
2 int n, target; cin >> n >> target;
3 vector<int> a(n);
4 for (int i = 0; i < n; i++) cin >> a[i];
5
6 // encontrar primera posición >= target
7 int l = 0, r = n - 1, first_pos = n;
8 while (l <= r) {
9     int mid = l + (r - 1) / 2;
10    if (a[mid] >= target) {
11        first_pos = mid;
12        r = mid - 1;
13    } else {
14        l = mid + 1;
15    }
16}
17
18 // encontrar última posición <= target
19 l = 0, r = n - 1;
20 int last_pos = -1;
21 while (l <= r) {
22     int mid = l + (r - 1) / 2;
23     if (a[mid] <= target) {
24         last_pos = mid;
25         l = mid + 1;
26     } else {
27         r = mid - 1;
28     }
29}
30
31 // binary search en función monótona
32 function<bool(int)> check = [&](int x) {
33     return true; // condición
34};
35 l = 0, r = 1e9;
36 int ans = -1;
37 while (l <= r) {
38     int mid = l + (r - 1) / 2;
39     if (check(mid)) {
40         ans = mid;
41         l = mid + 1; // o r = mid - 1 dependiendo del problema
42     } else {
43         r = mid - 1; // o l = mid + 1
44     }
45}
```

### 1.2. Fast Io

```
1 #include <bits/stdc++.h>
2 #include <ext/pb_ds/assoc_container.hpp>
3 #include <ext/pb_ds/tree_policy.hpp>
4
5 #define cpu() ios::sync_with_stdio(false);cin.tie(nullptr);
6
7 using namespace std;
```

```

8 using namespace __gnu_pbds;
9 template <class T>
10 using ordered_set = tree<T, null_type, less_equal<T>, rb_tree_tag,
11     tree_order_statistics_node_update>;
12
13 #define pb push_back
14 #define sz(a) ((int)(a).size())
15 #define ff first
16 #define ss second
17 #define all(a) (a).begin(), (a).end()
18 #define allr(a) (a).rbegin(), (a).rend()
19 #define approx(a) fixed << setprecision(a)
20
21 template <class T> void read(vector<T> &v);
22 template <class F, class S> void read(pair<F, S> &p);
23 template <class T, size_t Z> void read(array<T, Z> &a);
24 template <class T> void read(T &x) {cin >> x;}
25 template <class R, class... T> void read(R& r, T&... t){read(r); read(t...);}
26 template <class T> void read(vector<T> &v) {for(auto& x : v) read(x);}
27 template <class F, class S> void read(pair<F, S> &p) {read(p.ff, p.ss);}
28 template <class T, size_t Z> void read(array<T, Z> &a) { for(auto &x : a) read(x); }
29
30 template <class F, class S> void pr(const pair<F, S> &x);
31 template <class T> void pr(const T &x) {cout << x;}
32 template <class R, class... T> void pr(const R& r, const T&... t) {pr(r); pr(t...);}
33 template <class F, class S> void pr(const pair<F, S> &x) {pr("{", x.ff, ",", x.ss, "}")\n";}
34 void ps() {pr("\n");}
35 template <class T> void ps(const T &x) {pr(x); ps();}
36 template <class T> void ps(vector<T> &v) {for(auto& x : v) pr(x, ' '); ps();}
37 template <class T, size_t Z> void ps(const array<T, Z> &a) { for(auto &x : a) pr(x, ' '); ps()
38     (); }
39
40 using ll = long long;
41 const double PI = 3.141592653589793;
42 const ll MX = 1e9 + 1;
43
44 void solve() {
45
46 }
47
48 int main() {
49     cpu();
50
51     int t = 1;
52     //cin >> t;
53     while (t--) {
54         solve();
55     }
56
57     return 0;
58 }

```

### 1.3. Sliding Window

```

1 int n, k; cin >> n >> k;
2 vector<int> a(n);
3 for (int i = 0; i < n; i++) cin >> a[i];

```

```

33     }
34 }
35 vector<int> occurrences;
36 for (int i = pattern.size() + 1; i < combined.size(); i++) {
37     if (z_combined[i] == pattern.size()) {
38         occurrences.push_back(i - pattern.size() - 1);
39     }
40 }
41 // z[i] = longitud del substring mas largo que empieza en i y es prefijo de s
42 // occurrences contiene las posiciones donde pattern aparece en text

```

```

40     cur = nodes[cur].next[idx];
41 }
42 return nodes[cur].count;
43 }
44 }
45 }
46 // uso:
47 // Trie trie;
48 // int n; cin >> n;
49 // for (int i = 0; i < n; i++) {
50 //     string s; cin >> s;
51 //     trie.insert(s);
52 }
53 // string query; cin >> query;
54 // bool exists = trie.search(query); // true si query existe en el trie
55 // int count = trie.count_prefix(query); // cantidad de strings que tienen query como
56 //     prefijo
57
58 Trie trie;
59 trie.insert("hello");
60 trie.insert("hell");
61 bool found = trie.search("hello"); // true
62 int count = trie.count_prefix("hel"); // 2 (hello y hell)

```

### 11.3. Z Algorithm

```

1 string s; cin >> s;
2 int n = s.size();
3 vector<int> z(n);
4 int l = 0, r = 0;
5 for (int i = 1; i < n; i++) {
6     if (i <= r) {
7         z[i] = min(r - i + 1, z[i - 1]);
8     }
9     while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
10        z[i]++;
11    }
12    if (i + z[i] - 1 > r) {
13        l = i;
14        r = i + z[i] - 1;
15    }
16 }
17
18 string pattern, text;
19 cin >> pattern >> text;
20 string combined = pattern + "#" + text;
21 vector<int> z_combined(combined.size());
22 int l_combined = 0, r_combined = 0;
23 for (int i = 1; i < combined.size(); i++) {
24     if (i <= r_combined) {
25         z_combined[i] = min(r_combined - i + 1, z_combined[i - l_combined]);
26     }
27     while (i + z_combined[i] < combined.size() && combined[z_combined[i]] == combined[i +
28         z_combined[i]]) {
29         z_combined[i]++;
30     }
31     if (i + z_combined[i] - 1 > r_combined) {
32         l_combined = i;
33         r_combined = i + z_combined[i] - 1;
34     }
35 }
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
229
230
231
232
233
234
235
236
237
238
239
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
277
278
279
279
280
281
281
282
283
283
284
285
285
286
287
287
288
289
289
290
291
291
292
293
293
294
295
295
296
297
297
298
299
299
300
301
301
302
302
303
303
304
304
305
305
306
306
307
307
307
308
308
309
309
310
310
311
311
311
312
312
312
313
313
313
314
314
314
315
315
315
316
316
316
317
317
317
318
318
318
319
319
319
320
320
320
321
321
321
322
322
322
323
323
323
324
324
324
325
325
325
326
326
326
327
327
327
328
328
328
329
329
329
330
330
330
331
331
331
332
332
332
333
333
333
334
334
334
335
335
335
336
336
336
337
337
337
338
338
338
339
339
339
340
340
340
341
341
341
342
342
342
343
343
343
344
344
344
345
345
345
346
346
346
347
347
347
348
348
348
349
349
349
350
350
350
351
351
351
352
352
352
353
353
353
354
354
354
355
355
355
356
356
356
357
357
357
358
358
358
359
359
359
360
360
360
361
361
361
362
362
362
363
363
363
364
364
364
365
365
365
366
366
366
367
367
367
368
368
368
369
369
369
370
370
370
371
371
371
372
372
372
373
373
373
374
374
374
375
375
375
376
376
376
377
377
377
378
378
378
379
379
379
380
380
380
381
381
381
382
382
382
383
383
383
384
384
384
385
385
385
386
386
386
387
387
387
388
388
388
389
389
389
390
390
390
391
391
391
392
392
392
393
393
393
394
394
394
395
395
395
396
396
396
397
397
397
398
398
398
399
399
399
400
400
400
401
401
401
402
402
402
403
403
403
404
404
404
405
405
405
406
406
406
407
407
407
408
408
408
409
409
409
410
410
410
411
411
411
412
412
412
413
413
413
414
414
414
415
415
415
416
416
416
417
417
417
418
418
418
419
419
419
420
420
420
421
421
421
422
422
422
423
423
423
424
424
424
425
425
425
426
426
426
427
427
427
428
428
428
429
429
429
430
430
430
431
431
431
432
432
432
433
433
433
434
434
434
435
435
435
436
436
436
437
437
437
438
438
438
439
439
439
440
440
440
441
441
441
442
442
442
443
443
443
444
444
444
445
445
445
446
446
446
447
447
447
448
448
448
449
449
449
450
450
450
451
451
451
452
452
452
453
453
453
454
454
454
455
455
455
456
456
456
457
457
457
458
458
458
459
459
459
460
460
460
461
461
461
462
462
462
463
463
463
464
464
464
465
465
465
466
466
466
467
467
467
468
468
468
469
469
469
470
470
470
471
471
471
472
472
472
473
473
473
474
474
474
475
475
475
476
476
476
477
477
477
478
478
478
479
479
479
480
480
480
481
481
481
482
482
482
483
483
483
484
484
484
485
485
485
486
486
486
487
487
487
488
488
488
489
489
489
490
490
490
491
491
491
492
492
492
493
493
493
494
494
494
495
495
495
496
496
496
497
497
497
498
498
498
499
499
499
500
500
500
501
501
501
502
502
502
503
503
503
504
504
504
505
505
505
506
506
506
507
507
507
508
508
508
509
509
509
510
510
510
511
511
511
512
512
512
513
513
513
514
514
514
515
515
515
516
516
516
517
517
517
518
518
518
519
519
519
520
520
520
521
521
521
522
522
522
523
523
523
524
524
524
525
525
525
526
526
526
527
527
527
528
528
528
529
529
529
530
530
530
531
531
531
532
532
532
533
533
533
534
534
534
535
535
535
536
536
536
537
537
537
538
538
538
539
539
539
540
540
540
541
541
541
542
542
542
543
543
543
544
544
544
545
545
545
546
546
546
547
547
547
548
548
548
549
549
549
550
550
550
551
551
551
552
552
552
553
553
553
554
554
554
555
555
555
556
556
556
557
557
557
558
558
558
559
559
559
560
560
560
561
561
561
562
562
562
563
563
563
564
564
564
565
565
565
566
566
566
567
567
567
568
568
568
569
569
569
570
570
570
571
571
571
572
572
572
573
573
573
574
574
574
575
575
575
576
576
576
577
577
577
578
578
578
579
579
579
580
580
580
581
581
581
582
582
582
583
583
583
584
584
584
585
585
585
586
586
586
587
587
587
588
588
588
589
589
589
590
590
590
591
591
591
592
592
592
593
593
593
594
594
594
595
595
595
596
596
596
597
597
597
598
598
598
599
599
599
600
600
600
601
601
601
602
602
602
603
603
603
604
604
604
605
605
605
606
606
606
607
607
607
608
608
608
609
609
609
610
610
610
611
611
611
612
612
612
613
613
613
614
614
614
615
615
615
616
616
616
617
617
617
618
618
618
619
619
619
620
620
620
621
621
621
622
622
622
623
623
623
624
624
624
625
625
625
626
626
626
627
627
627
628
628
628
629
629
629
630
630
630
631
631
631
632
632
632
633
633
633
634
634
634
635
635
635
636
636
636
637
637
637
638
638
638
639
639
639
640
640
640
641
641
641
642
642
642
643
643
643
644
644
644
645
645
645
646
646
646
647
647
647
648
648
648
649
649
649
650
650
650
651
651
651
652
652
652
653
653
653
654
654
654
655
655
655
656
656
656
657
657
657
658
658
658
659
659
659
660
660
660
661
661
661
662
662
662
663
663
663
664
664
664
665
665
665
666
666
666
667
667
667
668
668
668
669
669
669
670
670
670
671
671
671
672
672
672
673
673
673
674
674
674
675
675
675
676
676
676
677
677
677
678
678
678
679
679
679
680
680
680
681
681
681
682
682
682
683
683
683
684
684
684
685
685
685
686
686
686
687
687
687
688
688
688
689
689
689
690
690
690
691
691
691
692
692
692
693
693
693
694
694
694
695
695
695
696
696
696
697
697
697
698
698
698
699
699
699
700
700
700
701
701
701
702
702
702
703
703
703
704
704
704
705
705
705
706
706
706
707
707
707
708
708
708
709
709
709
710
710
710
711
711
711
712
712
712
713
713
713
714
714
714
715
715
715
716
716
716
717
717
717
718
718
718
719
719
719
720
720
720
721
721
721
722
722
722
723
723
723
724
724
724
725
725
725
726
726
726
727
727
727
728
728
728
729
729
729
730
730
730
731
731
731
732
732
732
733
733
733
734
734
734
735
735
735
736
736
736
737
737
737
738
738
738
739
739
739
740
740
740
741
741
741
742
742
742
743
743
743
744
744
744
745
745
745
746
746
746
747
747
747
748
748
748
749
749
749
750
750
750
751
751
751
752
752
752
753
753
753
754
754
754
755
755
755
756
756
756
757
757
757
758
758
758
759
759
759
760
760
760
761
761
761
762
762
762
763
763
763
764
764
764
765
765
765
766
766
766
767
767
767
768
768
768
769
769
769
770
770
770
771
771
771
772
772
772
773
773
773
774
774
774
775
775
775
776
776
776
777
777
777
778
778
778
779
779
779
780
780
780
781
781
781
782
782
782
783
783
783
784
784
784
785
785
785
786
786
786
787
787
787
788
788
788
789
789
789
790
790
790
791
791
791
792
792
792
793
793
793
794
794
794
79
```

```

34 pi(10^11) = 4.118.054.813
35 pi(10^12) = 37.607.912.018
36
37 // Cantidad de divisores
38 sigma0(60) = 12 -> [1, 2, 3, 4, 6, 10, 12, 15, 20, 30, 60]
39 sigma0(120) = 16 -> [1, 2, 3, 4, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, 120]
40 sigma0(180) = 18 -> [1, 2, 3, 4, 5, 6, 9, 10, 12, 15, 18, 20, 30, 36, 60, 90, 180]
41 sigma0(240) = 20 -> [1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 30, 40, 60, 80, 120, 240]
42 sigma0(360) = 24
43 sigma0(720) = 30
44 sigma0(840) = 32
45 sigma0(1.260) = 36
46 sigma0(1.680) = 40
47 sigma0(10.080) = 72
48 sigma0(15.120) = 80
49 sigma0(50.400) = 108
50 sigma0(83.160) = 128
51 sigma0(110.880) = 144
52 sigma0(498.960) = 200
53 sigma0(554.400) = 216
54 sigma0(1.081.080) = 256
55 sigma0(1.441.440) = 288
56 sigma0(4.324.320) = 384
57 sigma0(8.648.640) = 448
58
59 // Suma de divisores
60 sigma1(96) = 252 -> [1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 96]
61 sigma1(108) = 280 -> [1, 2, 3, 4, 6, 9, 12, 18, 27, 36, 54, 108]
62 sigma1(120) = 360 -> [1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, 120]
63 sigma1(144) = 403 -> [1, 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 36, 48, 72, 144]
64 sigma1(168) = 480
65 sigma1(960) = 3.048
66 sigma1(1.008) = 3.224
67 sigma1(1.080) = 3.600
68 sigma1(1.200) = 3.844
69 sigma1(4.620) = 16.128
70 sigma1(4.680) = 16.380
71 sigma1(5.040) = 19.344
72 sigma1(5.760) = 19.890
73 sigma1(8.820) = 31.122
74 sigma1(9.240) = 34.560
75 sigma1(10.080) = 39.312
76 sigma1(10.920) = 40.320
77 sigma1(32.760) = 131.040
78 sigma1(35.280) = 137.826
79 sigma1(36.960) = 145.152
80 sigma1(37.800) = 148.800
81 sigma1(60.480) = 243.840
82 sigma1(64.680) = 246.240
83 sigma1(65.520) = 270.816
84 sigma1(70.560) = 280.098
85 sigma1(95.760) = 386.880
86 sigma1(98.280) = 403.200
87 sigma1(100.800) = 409.448
88 sigma1(491.400) = 2.083.200
89 sigma1(498.960) = 2.160.576
90 sigma1(514.080) = 2.177.280
91 sigma1(982.800) = 4.305.280
92 sigma1(997.920) = 4.390.848

```

```

16 vector<int> pi_combined(combined.size());
17 for (int i = 1; i < combined.size(); i++) {
18     int j = pi_combined[i - 1];
19     while (j > 0 && combined[i] != combined[j]) {
20         j = pi_combined[j - 1];
21     }
22     if (combined[i] == combined[j]) j++;
23     pi_combined[i] = j;
24 }
25 vector<int> occurrences;
26 for (int i = pattern.size() + 1; i < combined.size(); i++) {
27     if (pi_combined[i] == pattern.size()) {
28         occurrences.push_back(i - 2 * pattern.size());
29     }
30 }
31 // pi[i] = longitud del prefijo mas largo que es sufijo en s[0..i]
32 // occurrences contiene las posiciones donde pattern aparece en text

```

## 11.2. Trie

```

1 struct Trie {
2     struct Node {
3         vector<int> next;
4         bool is_end;
5         int count;
6         Node() : next(26, -1), is_end(false), count(0) {}
7     };
8     vector<Node> nodes;
9     Trie() { nodes.emplace_back(); }
10
11 void insert(string& s) {
12     int cur = 0;
13     for (char c : s) {
14         int idx = c - 'a';
15         if (nodes[cur].next[idx] == -1) {
16             nodes[cur].next[idx] = nodes.size();
17             nodes.emplace_back();
18         }
19         cur = nodes[cur].next[idx];
20         nodes[cur].count++;
21     }
22     nodes[cur].is_end = true;
23 }
24
25 bool search(string& s) {
26     int cur = 0;
27     for (char c : s) {
28         int idx = c - 'a';
29         if (nodes[cur].next[idx] == -1) return false;
30         cur = nodes[cur].next[idx];
31     }
32     return nodes[cur].is_end;
33 }
34
35 int count_prefix(string& s) {
36     int cur = 0;
37     for (char c : s) {
38         int idx = c - 'a';
39         if (nodes[cur].next[idx] == -1) return 0;

```

```

45     return 1;
46 }

```

#### 10.4. Modular Slae

```

1 int gauss (vector < bitset<N> > a, int n, int m, bitset<N> & ans) {
2     vector<int> where (m, -1);
3     for (int col=0, row=0; col<m && row<n; ++col) {
4         for (int i=row; i<n; ++i)
5             if (a[i][col]) {
6                 swap (a[i], a[row]);
7                 break;
8             }
9             if (! a[row][col])
10                 continue;
11             where[col] = row;
12
13             for (int i=0; i<n; ++i)
14                 if (i != row && a[i][col])
15                     a[i] ^= a[row];
16             ++row;
17     }
18     // The rest of implementation is the same as above
19 }

```

#### 10.5. Simpson'S Integration

```

1 // Integration by Simpson's formula
2 const int N = 1000 * 1000; // number of steps (already multiplied by 2)
3
4 double simpson_integration(double a, double b){
5     double h = (b - a) / N;
6     double s = f(a) + f(b); // a = x_0 and b = x_2n
7     for (int i = 1; i <= N - 1; ++i) { // Refer to final Simpson's formula
8         double x = a + h * i;
9         s += f(x) * ((i & 1) ? 4 : 2);
10    }
11    s *= h / 3;
12    return s;
13 }

```

### 11. String

#### 11.1. Kmp

```

1 string s; cin >> s;
2 int n = s.size();
3 vector<int> pi(n);
4 for (int i = 1; i < n; i++) {
5     int j = pi[i - 1];
6     while (j > 0 && s[i] != s[j]) {
7         j = pi[j - 1];
8     }
9     if (s[i] == s[j]) j++;
10    pi[i] = j;
11 }
12
13 string pattern, text;
14 cin >> pattern >> text;
15 string combined = pattern + "#" + text;

```

```

93 sigma1(1.048.320) = 4.464.096
94 sigma1(4.979.520) = 22.189.440
95 sigma1(4.989.600) = 22.686.048
96 sigma1(5.045.040) = 23.154.768
97 sigma1(9.896.040) = 44.323.200
98 sigma1(9.959.040) = 44.553.600
99 sigma1(9.979.200) = 45.732.192
100
101 // Factoriales
102 0! = 1 (int)
103 1! = 1
104 2! = 2
105 3! = 6
106 4! = 24
107 5! = 120
108 6! = 720
109 7! = 5.040
110 8! = 40.320
111 9! = 362.880
112 10! = 3.628.800
113 11! = 39.916.800
114 12! = 479.001.600 (int)
115 13! = 6.227.020.800 (11)
116 14! = 87.178.291.200
117 15! = 1.307.674.368.000
118 16! = 20.922.789.888.000
119 17! = 355.687.428.096.000
120 18! = 6.402.373.705.728.000
121 19! = 121.645.100.408.832.000
122 20! = 2.432.902.008.176.640.000 (11)
123 21! = 51.090.942.171.709.400.000 (_int128_t)
124
125 // Límites de enteros
126 max signed char = 127
127 max unsigned char = 255
128 max signed int = 2.147.483.647
129 max unsigned int = 4.294.967.295
130 max signed long long = 9.223.372.036.854.775.807
131 max unsigned long long = 18.446.744.073.709.551.615
132 max signed __int128_t = 170.141.183.460.469.231.731.687.303.715.884.105.727
133 max unsigned __int128_t = 340.282.366.920.938.463.463.374.607.431.768.211.456

```

#### 1.5. Two Pointers

```

1 int n, target; cin >> n >> target;
2 vector<int> a(n);
3 for (int i = 0; i < n; i++) cin >> a[i];
4
5 // encontrar subarray con suma = target
6 int l = 0, sum = 0;
7 for (int r = 0; r < n; r++) {
8     sum += a[r];
9     while (sum > target && l <= r) {
10         sum -= a[l++];
11     }
12     if (sum == target) {
13         // subarray [l, r] tiene suma = target
14     }
15 }

```

```

16 // encontrar número de subarrays con suma <= target
17 l = 0, sum = 0;
18 long long count = 0;
19 for (int r = 0; r < n; r++) {
20     sum += a[r];
21     while (sum > target && l <= r) {
22         sum -= a[l++];
23     }
24     count += r - l + 1;
25 }
26 // count = número de subarrays con suma <= target

```

## 2. Bit Manipulation

Técnicas para manipular bits individuales y operaciones a nivel de bit. Incluye macros útiles para competencias de programación.

### 2.1. Bits

Macros esenciales para manipulación de bits: verificar potencias de 2, establecer/limpiar bits, contar bits, y operaciones con LSB/MSB.

```

1 using ull = unsigned long long;
2 const ull UNSIGNED_LL_MAX = 18'446'744'073'709'551'615;
3 // Verifica si S es potencia de dos (y distinto de cero)
4 #define isPowerOfTwo(S) ((S) && !((S) & ((S) - 1)))
5 // Retorna la potencia de dos más cercana a S
6 #define nearestPowerOfTwo(S) (1LL << lround(log2(S)))
7 // Calcula S % N cuando N es potencia de dos
8 #define modulo(S, N) ((S) & ((N) - 1))
9
10 // Verifica si el bit está encendido (bit en 1)
11 #define isOn(S, i) ((S) & (1LL<<(i)))
12 // Enciende el bit (Lo pone en 1)
13 #define setBit(S, i) ((S) |= (1LL<<(i)))
14 // Apaga el bit (Lo pone en 0)
15 #define clearBit(S, i) ((S) &= ~(1LL<<(i)))
16 // Invierte el estado del bit (0 <-> 1)
17 #define toggleBit(S, i) ((S) ^= (1LL<<(i)))
18 // Enciende los primeros 'n' bits (idx-0)
19 #define setAll(S, n) ((S) = ((n)>=64 ? ~0LL : (1LL << (n))-1))
20
21 // Extrae el bit menos significativo 0100 (Least Significant Bit)
22 #define lsb(S) ((S) & -(S))
23 // Número de ceros a la derecha (Posición del LSB, idx-0)
24 #define idxLastBit(x) __builtin_ctzll(x)
25 // Extrae el bit más significativo 0100 (Most Significant Bit)
26 #define msb(S) (1LL << (63 - __builtin_clzll(S)))
27 // Posición del MSB (63 - ceros a la izquierda, idx-0)
28 #define idxFirstBit(x) (63 - __builtin_clzll(x))
29
30 #define countAllOnes(x) __builtin_popcountll(x)
31 // Apaga el último bit encendido (el menos significativo)
32 #define turnOffLastBit(S) ((S) & ((S) - 1))
33 // Enciende el último cero menos significativo
34 #define turnOnLastZero(S) ((S) | ((S) + 1))
35 // Apaga todos los bits encendidos más a la derecha consecutivos
36 #define turnOffLastConsecutiveBits(S) ((S) & ((S) + 1))
37 // Enciende los ceros consecutivos más a la derecha

```

```

17 if (i != k)
18     det = -det;
19 det *= a[i][i];
20 for (int j=i+1; j<n; ++j)
21     a[i][j] /= a[i][i];
22 for (int j=0; j<n; ++j)
23     if (j != i && abs (a[j][i]) > EPS)
24         for (int k=i+1; k<n; ++k)
25             a[j][k] -= a[i][k] * a[j][i];
26 }
27 cout << det;

```

### 10.3. Linear Equations

```

1 // Gauss method for solving system of linear equations
2 const double EPS = 1e-9;
3 const int INF = 2; // it doesn't actually have to be infinity or a big number
4
5 int gauss (vector < vector<double> > a, vector<double> & ans) {
6     int n = (int) a.size();
7     int m = (int) a[0].size() - 1;
8
9     vector<int> where (m, -1);
10    for (int col=0, row=0; col<m && row<n; ++col) {
11        int sel = row;
12        for (int i=row; i<n; ++i)
13            if (abs (a[i][col]) > abs (a[sel][col]))
14                sel = i;
15        if (abs (a[sel][col]) < EPS)
16            continue;
17        for (int i=col; i<=m; ++i)
18            swap (a[sel][i], a[row][i]);
19        where[col] = row;
20
21        for (int i=0; i<n; ++i)
22            if (i != row) {
23                double c = a[i][col] / a[row][col];
24                for (int j=col; j<=m; ++j)
25                    a[i][j] -= a[row][j] * c;
26            }
27        ++row;
28    }
29
30    ans.assign (m, 0);
31    for (int i=0; i<m; ++i)
32        if (where[i] != -1)
33            ans[i] = a[where[i]][m] / a[where[i]][i];
34    for (int i=0; i<n; ++i) {
35        double sum = 0;
36        for (int j=0; j<m; ++j)
37            sum += ans[j] * a[i][j];
38        if (abs (sum - a[i][m]) > EPS)
39            return 0;
40    }
41
42    for (int i=0; i<m; ++i)
43        if (where[i] == -1)
44            return INF;

```

- Suma de los primeros  $n$  cubos

$$\sum_{i=1}^n i^3 = 1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$

- Suma de cuadrados

$$\sum_{i=1}^n i^2 = 1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

- Suma de potencias cuartas

$$\sum_{i=1}^n i^4 = 1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

- Serie Geométrica (Finita)

$$\sum_{i=0}^n a \cdot r^i = a + ar + ar^2 + \cdots + ar^n = \frac{a(r^{n+1} - 1)}{r - 1} \quad (r \neq 0, r \neq 1)$$

- Serie Geométrica (Infinita)

$$\sum_{i=0}^{\infty} a \cdot r^i = a + ar + ar^2 + \cdots = \frac{a}{1-r} \quad (|r| < 1)$$

- Suma potencias de  $1/2$  (Infinita)

$$\sum_{i=1}^{\infty} \left(\frac{1}{2}\right)^i = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots = 1$$

- Suma  $i \cdot 2^i$

$$\sum_{i=1}^n i \cdot 2^i = 1 \cdot 2^1 + 2 \cdot 2^2 + 3 \cdot 2^3 + \cdots + n \cdot 2^n = (n-1)2^{n+1} + 2$$

- Suma  $i/2^i$

$$\sum_{i=1}^n \frac{i}{2^i} = \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \cdots + \frac{n}{2^n} = 2 - \frac{n+2}{2^n}$$

## 10.2. Determinant Of A Matrix

```

1 // Calculating the determinant of a matrix by Gauss
2 const double EPS = 1E-9;
3 int n;
4 vector<vector<double>> a (n, vector<double> (n));
5
6 double det = 1;
7 for (int i=0; i<n; ++i) {
8     int k = i;
9     for (int j=i+1; j<n; ++j)
10        if (abs (a[j][i]) > abs (a[k][i]))
11            k = j;
12        if (abs (a[k][i]) < EPS) {
13            det = 0;
14            break;
15        }
16        swap (a[i], a[k]);

```

```

38 #define turnOnLastConsecutiveZeroes(S) ((S) | ((S) - 1))
39
40 // Máscara de bits (mask -> subconjunto) 0(2^N)
41 for (int mask = 0; mask < (1 << N); mask++)
42
43 // Recorrer subconjuntos de un superconjunto (menos el vacío)
44 int b = 0b1011; // Representación binaria de un decimal en int
45 for (int i = b; i; i = (i - 1) & b) {
46     cout << bitset<4>(i) << "\n";
47 }
48
49 void printBin(ll x) {
50     // 63 -> unsigned 11, 62 -> 11, 31 -> unsigned int, 30 -> int
51     for (ll i = 63; i >= 0; i--)
52         cout << ((x >> i) & 1);
53     cout << '\n';
54 }

```

## 3. Combinatory

### 3.1. Combi Brute Sin Mod

```

1 // nCk brute force sin MOD n <= 20
2 long long nCk_bruteforce(long long n, long long k) {
3     if (k < 0 || k > n) return 0;
4     long long res = 1;
5     for (long long i = 1; i <= k; i++) {
6         res = res * (n - i + 1) / i; // aquí la división es exacta
7     }
8     return res;
9 }
10
11 // nPk brute force sin MOD n <= 20
12 long long nPk_bruteforce(long long n, long long k) {
13     if (k < 0 || k > n) return 0;
14     long long res = 1;
15     for (long long i = 0; i < k; i++) {
16         res *= (n - i);
17     }
18     return res;
19 }

```

### 3.2. Combinatory

OJO: Es necesario usar binpow con MOD primo

```

1 // Devuelve el inverso modular de a mod MOD
2 // Usa el Teorema Pequeño de Fermat: a^(MOD-2) === a^(-1) (mod MOD)
3 // (válido solo si MOD es primo)
4 ll inv(ll a, ll p = MOD) {
5     return binpow(a, p - 2, p);
6 }
7
8 // Factoriales e inversos factoriales precomputados
9 // fact[n] = n! mod MOD
10 // invf[n] = (n!)^(-1) mod MOD
11 // Precomputa en 0(n)
12 vector<ll> fact(MAXN + 1), invf(MAXN + 1);
13

```

```

14 void precompute_factorials() {
15     fact[0] = 1;
16     for (int i = 1; i <= MAXN; i++) {
17         fact[i] = fact[i - 1] * i % MOD;
18     }
19     invf[MAXN] = inv(fact[MAXN]);
20     for (int i = MAXN; i > 0; i--) {
21         invf[i - 1] = invf[i] * i % MOD;
22     }
23 }
24
25 // Combinatoria de n en k: nCk(n, k) para n <= 10^6
26 // "n choose k" = n! / (k! * (n-k)!) mod MOD
27 // Retorna 0 si k > n
28 ll nCk(ll n, ll k) {
29     if (k < 0 || k > n) return 0;
30     return fact[n] * invf[k] % MOD * invf[n - k] % MOD;
31 }
32
33 // Permutación de n en k: nPk(n, k) para n <= 10^6
34 // Calcula permutaciones: "n permute k" = n! / (n-k)! mod MOD
35 // Retorna 0 si k > n
36 ll nPk(ll n, ll k) {
37     if (k < 0 || k > n) return 0;
38     return fact[n] * invf[n - k] % MOD;
39 }

```

## 4. Data Structures

### 4.1. Fenwick Tree

```

1 struct FenwickTree {
2     vector<long long> tree;
3     int n;
4
5     FenwickTree(int size) : n(size + 1) {
6         tree.assign(n, 0);
7     }
8
9     void update(int idx, long long delta) {
10        for (idx++; idx < n; idx += idx & -idx) {
11            tree[idx] += delta;
12        }
13    }
14
15    long long query(int idx) {
16        long long sum = 0;
17        for (idx++; idx > 0; idx -= idx & -idx) {
18            sum += tree[idx];
19        }
20        return sum;
21    }
22
23    long long range_query(int l, int r) {
24        return query(r) - query(l - 1);
25    }
26
27 };

```

```

8     for (int j = i * i; j < MAX_V; j += i) {
9         composite[j] = true;
10    }
11 }
12 }
13
14 int main() {
15     sieve();
16     for (int i = 2; i < 100; i++) {
17         cout << i << " is_prime: " << !composite[i] << '\n';
18     }
19 }

```

## 9.8. Sum Of Divisors

```

1 // Sum of divs
2 long long SumOfDivisors(long long num) {
3     long long total = 1;
4
5     for (int i = 2; (long long)i * i <= num; i++) {
6         if (num % i == 0) {
7             int e = 0;
8             do {
9                 e++;
10                num /= i;
11            } while (num % i == 0);
12
13            long long sum = 0, pow = 1;
14            do {
15                sum += pow;
16                pow *= i;
17            } while (e-- > 0);
18            total *= sum;
19        }
20
21        if (num > 1) {
22            total *= (1 + num);
23        }
24    }
25    return total;

```

## 10. Numerical Methods

### 10.1. Sumas Notables

#### ■ Suma de los primeros n números naturales (Gauss)

$$\sum_{i=1}^n i = 1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$$

#### ■ Suma de los primeros n números pares

$$\sum_{i=1}^n 2 \cdot i = 2 + 4 + 6 + \dots + 2n = n(n+1)$$

#### ■ Suma de los primeros n números impares

$$\sum_{i=1}^n (2 \cdot i - 1) = 1 + 3 + 5 + \dots + (2n - 1) = n^2$$

```

16 ///* phi(n) -> complejo: O(log(log(n)))
17 void phi_1_to_n(int n) {
18     vector<int> phi(n + 1);
19     for (int i = 0; i <= n; i++)
20         phi[i] = i;
21
22     for (int i = 2; i <= n; i++) {
23         if (phi[i] == i) {
24             for (int j = i; j <= n; j += i)
25                 phi[j] -= phi[j] / i;
26         }
27     }
28 }

```

### 9.5. Potenciacion Binaria

```

1 using ll = long long;
2 const int MAXN = 1e6; // límite superior de n
3 const ll MOD = 1e9 + 7; // primo grande
4
5 // Potenciacion binaria modular a^b mod p
6 ll binpow(ll a, ll b, ll m = MOD) {
7     a %= m;
8     ll res = 1;
9     while (b > 0) {
10         if (b & 1)
11             res = res * a % m;
12         a = a * a % m;
13         b >>= 1;
14     }
15     return res;
16 }

```

### 9.6. Sieve

```

1 // Criba de Eratostenes: Hasta N = 10^6
2 void sieve(vector<bool>& is_prime) {
3     int N = (int) is_prime.size();
4     if (!is_prime[0]) is_prime.assign(N+1, true);
5     is_prime[0] = is_prime[1] = false;
6     for (int p = 2; p * p <= N; p++) {
7         if (is_prime[p]) {
8             for (int i = p * p; i <= N; i += p) {
9                 is_prime[i] = false;
10            }
11        }
12    }
13 }

```

### 9.7. Sieve Bitset

```

1 // Hasta N = 10^8 aprox en 1s
2 const int MAX_V = 1e7 + 5;
3 bitset<MAX_V> composite;
4 void sieve() {
5     composite[0] = composite[1] = true;
6     for (int i = 2; i * i < MAX_V; i++) {
7         if (composite[i]) continue;

```

```

28 // uso:
29 // int n; cin >> n;
30 // FenwickTree ft(n);
31 // for (int i = 0; i < n; i++) {
32 //     long long x; cin >> x;
33 //     ft.update(i, x);
34 // }
35 // ft.update(idx, delta); // actualizar elemento en idx
36 // long long sum = ft.range_query(l, r); // suma en rango [l, r]

```

### 4.2. Find Two Numbers

```

1 // "find two number where the sum is x, and gcd(a, b) > 1" b
2 auto find = [&](ll x) {
3     for(int d = 2; d <= x / 2; d++){
4         if(x % d == 0){
5             ll m = 1, n = (x / d) - 1;
6             ll a = d * m, b = d * n;
7             if(__gcd(a, b) > 1){
8                 cout << a << ' ' << b;
9                 ps();
10                return;
11            }
12        }
13    }
14 };

```

### 4.3. Segment Tree

```

1 // "This segment_tree I understand better how it works"
2 template<typename T>
3 struct seg_tree {
4     int N;
5     T Z = 0;
6     vector<T> tree;
7
8     seg_tree(int N) : N(N) {
9         tree.resize(4 * N);
10    }
11
12    seg_tree(vector<T>& A) {
13        N = (int)A.size();
14        tree.resize(4 * N);
15        build(A, 1, 0, N-1);
16    }
17
18 private:
19    T op(T a, T b) {
20        return a + b;
21    }
22
23    void build(vector<T>& a, int node, int left, int right) {
24        if(left == right) {
25            tree[node] = a[left];
26            return;
27        }
28        int mid = (left + right) >> 1;
29        build(a, 2 * node, left, mid);
30        build(a, 2 * node + 1, mid + 1, right);
31        tree[node] = op(tree[2 * node], tree[2 * node + 1]);

```

```

32 }
33
34 void modify(int pos, T value, int node, int left, int right) {
35     if(left == right) {
36         tree[node] = value;
37         return;
38     }
39     int mid = (left + right) >> 1;
40     if(pos <= mid)
41         modify(pos, value, 2 * node, left, mid);
42     else
43         modify(pos, value, 2 * node + 1, mid + 1, right);
44     tree[node] = op(tree[2 * node], tree[2 * node + 1]);
45 }
46
47 T query(int l, int r, int node, int left, int right) {
48     if(r < left || l > right) return Z;
49     if(l <= left && right <= r) return tree[node];
50     int mid = (left + right) >> 1;
51     T leftSum = query(l, r, 2 * node, left, mid);
52     T rightSum = query(l, r, 2 * node + 1, mid + 1, right);
53     return op(leftSum, rightSum);
54 }
55
56 public:
57     void build(vector<T>& a) { build(a, 1, 0, N-1); }
58     void modify(int pos, T value) { modify(pos, value, 1, 0, N-1); }
59     T query(int l, int r) { return query(l, r, 1, 0, N-1); }
60 };

```

#### 4.4. Sparse Table

```

1 int n; cin >> n;
2 vector<long long> a(n);
3 for (int i = 0; i < n; i++) cin >> a[i];
4 int k = log2(n) + 1;
5 vector<vector<long long>> st(n, vector<long long>(k));
6 for (int i = 0; i < n; i++) st[i][0] = a[i];
7 for (int j = 1; j < k; j++) {
8     for (int i = 0; i + (1 << j) <= n; i++) {
9         st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
10    }
11 }
12 function<long long(int, int)> query = [&](int l, int r) {
13     int j = log2(r - l + 1);
14     return min(st[l][j], st[r - (1 << j) + 1][j]);
15 };
16 // query(l, r) = mínimo en rango [l, r] en O(1)
17 // cambiar min por max para máximo
18 // cambiar min por gcd para GCD en rango

```

### 5. Dp

#### 5.1. Digit Dp Pattern

```

1 string pattern; cin >> pattern; // ejemplo: "xxxxx3xxxx" donde x = dígito libre
2 int n = pattern.size();
3 long long k; cin >> k; // modulo
4

```

11 }

#### 9.3. Number Theory

```

1 // Divisores de N: Hasta N = 10^6
2 vector<int> divisores(int N) {
3     vector<int> divs;
4     for (int d = 1; d * d <= N; d++) {
5         if (N % d == 0) {
6             divs.push_back(d);
7             if (N / d != d) divs.push_back(N / d);
8         }
9     }
10    return divs;
11 }
12
13 // Factorizacion de N: Hasta N = 10^6
14 vector<pair<int, int>> factorizar(int N) {
15     vector<pair<int, int>> facts;
16     for (int p = 2; p * p <= N; p++) {
17         if (N % p == 0) {
18             int exp = 0;
19             while (N % p == 0) {
20                 exp++;
21                 N /= p;
22             }
23             facts.push_back({ p, exp });
24         }
25     }
26     if (N > 1) facts.push_back({ N, 1 });
27     return facts;
28 }
29
30 // Primalidad: Hasta N = 10^6 - O(sqrt(N))
31 bool isPrime(int N) {
32     if (N < 2) return false;
33     for (int d = 2; d * d <= N; d++) {
34         if (N % d == 0) return false;
35     }
36     return true;
37 }

```

#### 9.4. Phi Euler

$\Phi(n)$  = contar la cantidad de numeros coprimos entre 1 a n

```

1 int phi(int n) {
2     int ans = n;
3     for(int i = 2; i * i <= n; i++) {
4         if(n % i == 0) {
5             while (n % i == 0) {
6                 n /= i;
7             }
8             ans -= ans / i;
9         }
10    }
11    if(n > 1) {
12        ans -= ans / n;
13    }
14    return ans;
15 }

```

```

29         else swap(p.x, p.y);
30     }
31     return edges;
32 }
33 }
```

## 9. Number Theory

### 9.1. Euler Toliente

```

1 class EulerTotiente {
2     public:
3     /* metodo en O(sqrt(n))
4     template <typename T>
5     T euler_classic(T n) {
6         T result = n;
7         for(T i = 2; i * i <= n; i++) {
8             if(n % i == 0) {
9                 while(n % i == 0) n /= i;
10                result -= result / i;
11            }
12        }
13        if(n > 1) {
14            result -= result / n;
15        }
16        return result;
17    }
18
19    /* metodo en O(nlog(log(n)))
20    void euler_faster(int n) {
21        vector<int> phi(n + 1);
22        for(int i = 0; i <= n; i++) {
23            phi[i] = i;
24        }
25        for(int i = 2; i <= n; i++) {
26            if(phi[i] == i) {
27                for(int j = i; j <= n; j += i) {
28                    phi[j] -= phi[j] / i;
29                }
30            }
31        }
32        for(int i = 1; i <= n; i++) {
33            cout << i << ' ' << phi[i] << '\n';
34        }
35    }
36 };
```

### 9.2. Gcd Lcm

```

1 // Maximo comun divisor (GCD): Algoritmo de Euclides
2 int gcd(int a, int b) {
3     if (a > b) swap(a, b);
4     if (a == 0) return b;
5     return gcd(b % a, a);
6 }
7
8 // Minimo comun multiplo (LCM): Calculado con GCD
9 int lcm(int a, int b) {
10    return (a * b) / gcd(a, b);
11 }
```

```

5     vector<vector<vector<long long>>> dp(n, vector<vector<long long>>(k, vector<long long>(2,
6         -1)));
7
8     function<long long(int, int, bool, bool)> solve = [&](int pos, int rem, bool tight, bool
9         started) {
10         if (pos == n) {
11             return (started && rem == 0) ? 1LL : 0LL;
12         }
13         if (started && !tight && dp[pos][rem][tight ? 1 : 0] != -1) {
14             return dp[pos][rem][tight ? 1 : 0];
15         }
16         long long res = 0;
17         if (pattern[pos] != 'x' && pattern[pos] != 'X') {
18             int fixed_digit = pattern[pos] - '0';
19             bool new_tight = tight && (fixed_digit == 9);
20             bool new_started = started || (fixed_digit > 0);
21             int new_rem = (rem * 10 + fixed_digit) % k;
22             res += solve(pos + 1, new_rem, new_tight, new_started);
23         } else {
24             int limit = tight ? 9 : 9;
25             int start_digit = (pos == 0) ? 1 : 0; // primer dígito no puede ser 0
26             for (int d = start_digit; d <= limit; d++) {
27                 bool new_tight = tight && (d == limit);
28                 bool new_started = started || (d > 0);
29                 int new_rem = (rem * 10 + d) % k;
30                 res += solve(pos + 1, new_rem, new_tight, new_started);
31             }
32         }
33         if (started && !tight) {
34             dp[pos][rem][tight ? 1 : 0] = res;
35         }
36     };
37
38     long long result = solve(0, 0, true, false);
39     // result = cantidad de números que siguen el patrón y son divisibles por k
40     // ejemplo: pattern = "xxxxx3xxxx", k = 7
41     // cuenta números tipo 1234534567 que son divisibles por 7
42     // x o X = dígito libre, cualquier otro carácter = dígito fijo
43 }
```

### 5.2. Digit Dp

```

1 string s; cin >> s; // número como string (puede ser muy grande, tipo 10^100)
2 int n = s.size();
3 long long k; cin >> k; // modulo
4
5 vector<vector<vector<long long>>> dp(n, vector<vector<long long>>(k, vector<long long>(2,
6         -1)));
7
8     function<long long(int, int, bool, bool)> solve = [&](int pos, int rem, bool tight, bool
9         started) {
10         if (pos == n) {
11             return (started && rem == 0) ? 1LL : 0LL;
12         }
13         if (started && !tight && dp[pos][rem][tight ? 1 : 0] != -1) {
14             return dp[pos][rem][tight ? 1 : 0];
15         }
16         long long res = 0;
17         int limit = tight ? (s[pos] - '0') : 9;
18 }
```

```

16 for (int d = 0; d <= limit; d++) {
17     bool new_tight = tight && (d == limit);
18     bool new_started = started || (d > 0);
19     int new_rem = (rem * 10 + d) % k;
20     res += solve(pos + 1, new_rem, new_tight, new_started);
21 }
22 if (started && !tight) {
23     dp[pos][rem][tight ? 1 : 0] = res;
24 }
25 return res;
26 };
27
28 long long result = solve(0, 0, true, false);
29 // result = cantidad de números <= s que son divisibles por k
30 // para contar en rango [a, b]: result_b - result_a-1
31 // ejemplo: s = "1000000", k = 7 -> contar números de 0 a 1000000 divisibles por 7

```

### 5.3. Edit Distance

```

1 string s1, s2; cin >> s1 >> s2;
2 int n = s1.size(), m = s2.size();
3 vector<vector<int>> dp(n + 1, vector<int>(m + 1));
4 for (int i = 0; i <= n; i++) dp[i][0] = i;
5 for (int j = 0; j <= m; j++) dp[0][j] = j;
6 for (int i = 1; i <= n; i++) {
7     for (int j = 1; j <= m; j++) {
8         if (s1[i - 1] == s2[j - 1]) {
9             dp[i][j] = dp[i - 1][j - 1];
10        } else {
11            dp[i][j] = 1 + min({dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]});
12        }
13    }
14 }
15 // dp[n][m] = edit distance (mínimo número de operaciones: insertar, eliminar, reemplazar)
16 // para convertir s1 en s2

```

### 5.4. Knapsack

```

1 int n, capacity; cin >> n >> capacity;
2 vector<int> weight(n), value(n);
3 for (int i = 0; i < n; i++) {
4     cin >> weight[i] >> value[i];
5 }
6 vector<long long> dp(capacity + 1, 0);
7 for (int i = 0; i < n; i++) {
8     for (int w = capacity; w >= weight[i]; w--) {
9         dp[w] = max(dp[w], dp[w - weight[i]] + value[i]);
10    }
11 }
12 // dp[capacity] = valor máximo que se puede obtener con capacidad máxima
13 // para versión con items ilimitados, cambiar el loop: for (int w = weight[i]; w <= capacity
14     ; w++)

```

### 5.5. Lcs

```

1 string s1, s2; cin >> s1 >> s2;
2 int n = s1.size(), m = s2.size();
3 vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
4 for (int i = 1; i <= n; i++) {
5     for (int j = 1; j <= m; j++) {

```

```

21     for (int v : adj[u]) {
22         if (--indeg[v] == 0) {
23             pq.push(v);
24         }
25     }
26 }
27 if ((int)order.size() != n) { return {}; }
28 return order;
29 }

```

## 8. Manhattan Distance

### 8.1. Farthest Pair Of Points

```

1 long long ans = 0;
2 for (int msk = 0; msk < (1 << d); msk++) {
3     long long mx = LLONG_MIN, mn = LLONG_MAX;
4     for (int i = 0; i < n; i++) {
5         long long cur = 0;
6         for (int j = 0; j < d; j++) {
7             if (msk & (1 << j)) cur += p[i][j];
8             else cur -= p[i][j];
9         }
10        mx = max(mx, cur);
11        mn = min(mn, cur);
12    }
13    ans = max(ans, mx - mn);
14 }

```

### 8.2. Nearest Neighbor In Each Octant

```

1 // Nearest Neighbor in each Octant in O(n log n)
2 struct point {
3     long long x, y;
4 };
5
6 // Returns a list of edges in the format (weight, u, v).
7 // Passing this list to Kruskal algorithm will give the Manhattan MST.
8 vector<tuple<long long, int, int>> manhattan_mst_edges(vector<point> ps) {
9     vector<int> ids(ps.size());
10    iota(ids.begin(), ids.end(), 0);
11    vector<tuple<long long, int, int>> edges;
12    for (int rot = 0; rot < 4; rot++) { // for every rotation
13        sort(ids.begin(), ids.end(), [&](int i, int j){
14            return (ps[i].x + ps[i].y) < (ps[j].x + ps[j].y);
15        });
16        map<int, int, greater<int>> active; // (xs, id)
17        for (auto i : ids) {
18            for (auto it = active.lower_bound(ps[i].x); it != active.end();
19                 active.erase(it++)) {
20                int j = it->second;
21                if (ps[i].x - ps[i].y > ps[j].x - ps[j].y) break;
22                assert(ps[i].x >= ps[j].x && ps[i].y >= ps[j].y);
23                edges.push_back({(ps[i].x - ps[j].x) + (ps[i].y - ps[j].y), i, j});
24            }
25            active[ps[i].x] = i;
26        }
27        for (auto &p : ps) { // rotate
28            if (rot & 1) p.x *= -1;

```

```

52     if(ind[i] == -1) dfs(i);
53
54     // reverse(components.begin(), components.end()); return components; // SCC in
55     // topological order
56     return components; // SCC in reverse topological order
57 }

```

### 7.13. Topo Sort Dfs

```

1 // O(N + M). 0-indexed. Retorna cualquier toposort válido (no necesariamente
2     lexicográficamente mínima)
3 vector<int> topo_sort(vector<vector<int>>& adj){
4     int n = adj.size();
5     bool cycle = false;
6     vector<int> topo, color(n); // 0 = no visitado, 1 = visitando, 2 = terminado
7
8     function<void(int)> dfs = [&](int u){
9         color[u] = 1;
10        for (int v : adj[u]){
11            if (color[v] == 0 && !cycle) dfs(v);
12            else if (color[v] == 1) cycle = true; // ciclo detectado
13        }
14        color[u] = 2;
15        topo.push_back(u);
16    };
17
18    for (int i = 0; i < n; i++){
19        if (color[i] == 0 && !cycle) dfs(i);
20    }
21    if (cycle) return {}; // no existe toposort
22    reverse(topo.begin(), topo.end());
23    return topo;
24 }

```

### 7.14. Topo Sort Kahns Bfs

```

1 // O((N + M)*logN). 0-indexed. Topological sort (Kahn BFS) con min-heap (lexicográficamente
2     mínimo)
3 vector<int> topo_sort(int n, const vector<vector<int>>& adj) {
4     vector<int> indeg(n, 0); // in-degree de cada nodo
5     for (int u = 0; u < n; u++) {
6         for (int v : adj[u]) {
7             indeg[v]++;
8         }
9     }
10    // min-heap para siempre sacar el nodo de menor índice
11    priority_queue<int, vector<int>, greater<int>> pq;
12    for (int u = 0; u < n; u++) {
13        if (indeg[u] == 0) pq.push(u);
14    }
15
16    vector<int> order;
17    order.reserve(n);
18    while (!pq.empty()) {
19        int u = pq.top();
20        pq.pop();
21        order.push_back(u);
22    }
23
24 }

```

```

6     if (s1[i - 1] == s2[j - 1]) {
7         dp[i][j] = dp[i - 1][j - 1] + 1;
8     } else {
9         dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
10    }
11 }
12 }
13 // dp[n][m] = longitud de LCS (Longest Common Subsequence)
14
15 string reconstruct_lcs() {
16     string lcs = "";
17     int i = n, j = m;
18     while (i > 0 && j > 0) {
19         if (s1[i - 1] == s2[j - 1]) {
20             lcs += s1[i - 1];
21             i--;
22             j--;
23         } else if (dp[i - 1][j] > dp[i][j - 1]) {
24             i--;
25         } else {
26             j--;
27         }
28     }
29     reverse(lcs.begin(), lcs.end());
30     return lcs;
31 } // lcs = string de la LCS

```

## 6. Geometry

### 6.1. 2D Geometry

Cookbook Geometría 2D - Operaciones con Puntos, Vectores, Líneas y Polígonos

#### PROBLEMAS TÍPICOS Y SUS SOLUCIONES

##### ■ ¿Un segmento toca un rectángulo?

Construir 4 lados del rectángulo como Line, usar `segIntersect(seg, lado_i)` para  $i = 1..4$ , y verificar con `between/extremos dentro` para cubrir el caso “segmento completamente dentro”.

Funciones clave: `segIntersect`, `between`, `Point/Line`.

##### ■ ¿Dos segmentos se cruzan?

Usar `segIntersect(11, 12)` si contar colineales y tocar endpoints, o `segStrictlyIntersect(11, 12)` si solo quieres cruce estricto.

##### ■ Distancia mínima entre dos segmentos

Real  $d = \text{segDist}(11, 12)$ ; Si  $d = 0$  entonces se tocan o cruzan.

##### ■ Distancia mínima de un punto a un segmento

Real  $d = \text{pointToSegDist}(p, seg)$ ;

##### ■ Distancia mínima de un punto a una recta infinita

Real  $d = \text{pointToLineDist}(p, line)$ ;

##### ■ ¿Punto dentro de un polígono cualquiera?

bool  $\text{inside} = \text{pointInPoly}(p, poly)$ ;

##### ■ Área de un polígono (ordenado CCW o CW)

$T_{\text{twice}} = \text{area}(poly)$ ; Real  $A = \text{fabs}((\text{Real})_{\text{twice}}) / 2.0$ ;

- ¿Cuándo un triángulo es degenerado?

Un triángulo con vértices  $a, b, c$  es degenerado (área = 0) si: (1) sus puntos son colineales  $\text{sign}(\text{cross}(b-a, c-a)) == 0$ , o (2) con lados  $a, b, c$ , falla la desigualdad triangular estricta ( $a+b > c, a+c > b, b+c > a$ ), i.e.,  $a + b = c$  (se aplana).

- ¿Punto sobre el borde de un polígono?

Recorrer lados  $[i, i + 1]$ , verificar si  $\text{pointOnSeg}(p, \text{Line}(a_i, a_j))$ .

- ¿Dos rectas infinitas se cruzan? Dame el punto

Primero verificar que no sean paralelas:  $\text{sign}(\text{cross}(\text{direction}(11), \text{direction}(12))) != 0$ , luego  $P<\text{Real}>\text{inter} = \text{lineIntersection}(11, 12)$ ;

- Ordenar vectores/puntos por ángulo alrededor del origen

$\text{sort}(\text{v.begin}(), \text{v.end}(), \text{polar} < \text{i64} >)$ ; o usar lambda con  $\text{up}()$  y  $\text{cross}()$ .

- Vector normal a un segmento

$P<\text{T}>\text{d} = \text{direction}(\text{seg}); P<\text{T}>\text{n} = \text{rotate90}(\text{d});$  (normal 90° CCW) o  $P<\text{Real}>\text{nu} = \text{normal}(\text{d});$  (normal unitaria).

- Detectar si un punto está a la izquierda/derecha de un borde

$\text{int s} = \text{side}(p, a, b)$ ; donde  $s > 0$  izquierda (CCW),  $s < 0$  derecha (CW),  $s = 0$  colineal.

- Clasificar orientación de un polígono

$\text{T twice} = \text{area}(\text{poly});$  Si  $\text{twice} > 0$  es CCW, si  $\text{twice} < 0$  es CW.

- Reflexión de un punto respecto a una recta

$P<\text{Real}>\text{pr} = \text{reflection}(p, \text{line});$

- Proyección de un punto sobre una recta

$P<\text{Real}>\text{proj} = \text{projection}(p, \text{line});$

- Intersección de rayo con segmento

$\text{rayIntersect}(\text{rayo1}, \text{rayo2})$  si ambos modelados como Line con origen en  $1.1[0]$  y dirección  $1.1[1]-1.1[0]$ . Para rayo vs segmento, combinar con  $\text{segIntersect}$  o rediseñar con  $\text{side}()$  y  $\text{dot}()$ .

- Rotar un punto/vector alrededor del origen

$P<\text{Real}>\text{rotado} = \text{rotate}(p, \text{angulo_en_radianes});$  Para 90° CCW rápido:  $\text{rotate90}(p);$

- Calcular ángulo de un vector respecto al eje X

$\text{Real ang} = \text{angle}(p);$  devuelve  $\text{atan2}(p.y, p.x)$  en radianes.

### TIPS IMPORTANTES

- Usa  $\text{Point} = P<\text{i64}>$  para coords enteras, evita overflow en  $\text{cross}/\text{dot}$
- Usa  $P<\text{Real}>$  (long double) para resultados con decimales (distancias, intersecciones)
- $\text{EPS} = 1e-9$  para comparaciones de flotantes
- $\text{sign}()$  y  $\text{cmp}()$  manejan tolerancia automáticamente
- Para convex hull y polígonos, mantén puntos en sentido CCW
- Verifica casos especiales: colineales, segmentos degenerados (mismo punto)
- Al leer entrada:  $\text{Point p}; \text{cin} >> p;$  (operador sobrecargado)

```

21   }
22   }
23 }
24 // mst_cost = costo del MST (Minimum Spanning Tree)

```

### 7.12. Scc

Algoritmo de Tarjan para encontrar componentes fuertemente conexas (SCC) en un grafo dirigido.

```

1 // "These works to find a componente fuertemente conexa that it's in directed graph"
2 struct SCC {
3     int N = 0, id;
4     vector<vector<int>> adj;
5     vector<int> ind, low;
6     stack<int> s;
7     vector<bool> in_stack;
8     vector<vector<int>> components;
9     vector<int> component_id;
10
11 //1-indexed
12 SCC(int n = 0){ N = n + 1, adj.assign(N, {}); }
13 SCC(const vector<vector<int>> &_adj){ adj = _adj, N = adj.size(); }
14
15 void add_edge(int from, int to){
16     adj[from].push_back(to);
17 }
18
19 void dfs(int u){
20     low[u] = ind[u] = id++;
21     s.push(u);
22     in_stack[u] = true;
23     for(int v : adj[u]){
24         if(ind[v] == -1){
25             dfs(v);
26             low[u] = min(low[u], low[v]);
27         }else if(in_stack[v]){
28             low[u] = min(low[u], ind[v]);
29         }
30     }
31     if(low[u] == ind[u]){
32         components.emplace_back();
33         vector<int> & comp = components.back();
34         while(true){
35             assert(!s.empty());
36             int x = s.top(); s.pop();
37             in_stack[x] = false;
38             component_id[x] = components.size() - 1;
39             comp.push_back(x);
40             if(x == u) break;
41         }
42     }
43 }
44
45 vector<vector<int>> get(){
46     ind.assign(N, -1); low.assign(N, -1); component_id.assign(N, -1);
47     s = stack<int>();
48     in_stack.assign(N, false);
49     id = 0;
50     components = {};
51     for(int i = 1; i < N; i++)

```

```

26     up[u][level] = up[up[u][level - 1]][level - 1];
27 }
28 for(int v : adj[u]){
29     if(v == p) continue;
30     dfs(v, u);
31 }
32 out[u] = ++timer;
33 }
34
35 bool is_ancestor(int p, int u){
36     return in[p] <= in[u] && out[p] >= out[u];
37 }
38
39 int query(int u, int v){
40     if(is_ancestor(u, v)) return u;
41     if(is_ancestor(v, u)) return v;
42
43     for(int bit = 1; bit >= 0; bit--){
44         if(is_ancestor(up[u][bit], v)) continue;
45         u = up[u][bit];
46     }
47     return up[u][0];
48 }
49
50 int ancestor(int u, int k){
51     if(depth[u] <= k) return -1;
52     for(int bit = 0; bit <= 1; bit++){
53         if(k >> bit & 1) u = up[u][bit];
54     }
55     return u;
56 }
57
58 int distance(int u, int v){
59     return depth[u] + depth[v] - 2 * depth[query(u, v)];
60 }
61 };

```

### 7.11. Prim

```

1 int n, m; cin >> n >> m;
2 vector<vector<pair<int, long long>>> adj(n + 1);
3 for (int i = 0; i < m; i++) {
4     int u, v; cin >> u >> v;
5     long long w; cin >> w;
6     adj[u].emplace_back(v, w);
7     adj[v].emplace_back(u, w);
8 }
9 vector<bool> vis(n + 1);
10 pqg<pair<long long, int>> pq;
11 pq.push({OLL, 1});
12 long long mst_cost = 0;
13 while (!pq.empty()) {
14     auto [w, u] = pq.top(); pq.pop();
15     if (vis[u]) continue;
16     vis[u] = true;
17     mst_cost += w;
18     for (auto [v, weight] : adj[u]) {
19         if (!vis[v]) {
20             pq.push({weight, v});

```

```

1 // ===== Tipos base =====
2 // - i64: reemplazo al long long
3 // - Real: para cálculos con flotantes de más precisión (distancias, intersecciones, etc.).
4 using i64 = long long;
5 using Real = long double;
6 constexpr Real EPS = 1e-9;
7
8 // ===== sign / cmp =====
9 // - sign(x): devuelve -1, 0, 1 (según x < 0, x == 0, x > 0)
10 // - cmp(a, b): compara a y b con tolerancia (para Real).
11 //   cmp(a, b) == 0 -> a "igual" a b, cmp(a, b) < 0 -> a < b, etc.
12 template <typename T>
13 int sign(T x) {
14     return (x > 0) - (x < 0);
15 }
16 int sign(Real x) {
17     return (x > EPS) - (x < -EPS);
18 }
19
20 template <typename T>
21 int cmp(T a, T b) {
22     return sign(a - b);
23 }
24
25 // ===== Punto y Línea =====
26 // - con T = i64 es usual en problemas de coordenadas enteras.
27 // - P<T>: punto/vector en 2D con componentes de tipo T.
28 // - L<T>: línea o segmento definido por dos puntos (l[0], l[1]).
29 template <typename T>
30 struct P {
31     T x = 0, y = 0;
32     P(T x = 0, T y = 0) : x(x), y(y) {}
33     friend istream& operator>>(istream &is, P &p) { return is >> p.x >> p.y; }
34     friend ostream& operator<<(ostream &os, P p) { return os << p.x << ' ' << p.y; }
35     friend bool operator==(P a, P b) { return cmp(a.x, b.x) == 0 && cmp(a.y, b.y) == 0; }
36     friend bool operator!=(P a, P b) { return !(a == b); }
37     P operator-() const { return P(-x, -y); }
38     P& operator+=(P a) {
39         x += a.x; y += a.y;
40         return *this;
41     }
42     P& operator-=(P a) {
43         x -= a.x; y -= a.y;
44         return *this;
45     }
46     P& operator*=(T d) {
47         x *= d; y *= d;
48         return *this;
49     }
50     P& operator/=(T d) {
51         x /= d; y /= d;
52         return *this;
53     }
54     friend P operator+(P a, P b) { return P(a) += b; }
55     friend P operator-(P a, P b) { return P(a) -= b; }
56     friend P operator*(P a, T d) { return P(a) *= d; }
57     friend P operator/(P a, T d) { return P(a) /= d; }
58     friend bool operator<(P a, P b) {
59         int sx = cmp(a.x, b.x);

```

```

60     return (sx != 0 ? sx == -1 : cmp(a.y, b.y) == -1);
61 }
62 };
63
64 template <typename T>
65 struct L {
66     array<P<T>, 2> l;
67     L(P<T> a = {}, P<T> b = {}) : l{a, b} {}
68 };
69
70 // ===== Operaciones vectoriales básicas =====
71 // - dot(a, b): producto escalar.
72 // - cross(a, b): producto cruzado escalar (a.x * b.y - a.y * b.x).
73 // - cross(p, a, b): cross(a - p, b - p) orientación de p respecto a ab.
74 // - square(a): |a|^2.
75 // - dist2(a, b): |a-b|^2, sin sqrt.
76 // - length(a): |a|.
77 // - dist(a, b): distancia euclíadiana entre a y b.
78 template <typename T>
79 T dot(P<T> a, P<T> b) { return a.x * b.x + a.y * b.y; }
80 template <typename T>
81 T cross(P<T> a, P<T> b) { return a.x * b.y - a.y * b.x; }
82 template <typename T>
83 T cross(P<T> p, P<T> a, P<T> b) { return cross(a - p, b - p); }
84 template <typename T>
85 T square(P<T> a) { return dot(a, a); }
86 template <typename T>
87 T dist2(P<T> a, P<T> b) { return square(a - b); }
88 template <typename T>
89 Real length(P<T> a) { return sqrtl(square(a)); }
90 template <typename T>
91 Real dist(P<T> a, P<T> b) { return length(a - b); }
92
93 // ===== Direcciones, ángulos, normales =====
94 // - normal(a): vector unitario en dirección de a.
95 // - up(a): true si el vector está en el semiplano "de arriba" (para ordenar por ángulo).
96 // - polar(a, b): criterio de orden por ángulo polar (para sort).
97 // - parallel(a, b): vectores paralelos.
98 // - sameDirection(a, b): vectores paralelos y apuntando en misma dirección.
99 // - angle(p): atan2(y, x).
100 // - rotate90(p): rota 90° CCW (útil para una normal rápida).
101 // - rotate(p, ang): rota un vector por un ángulo ang en radianes.
102 template <typename T>
103 P<Real> normal(P<T> a) {
104     Real len = length(a);
105     return P<Real>(a.x / len, a.y / len);
106 }
107 template <typename T>
108 bool up(P<T> a) {
109     return sign(a.y) > 0 || (sign(a.y) == 0 && sign(a.x) > 0);
110 }
111 // 3 colinear? recuerda remover (0,0) si lo usas en ordenamientos polares
112 template <typename T>
113 bool polar(P<T> a, P<T> b) {
114     bool ua = up(a), ub = up(b);
115     return ua != ub ? ua : sign(cross(a, b)) == 1;
116 }
117 template <typename T>
118 bool parallel(P<T> a, P<T> b) {

```

```

6 }
7 int find(int x) {
8     if (p[x] != x) p[x] = find(p[x]);
9     return p[x];
10 }
11 void merge(int x, int y) {
12     x = find(x), y = find(y);
13     if (x == y) return;
14     if (size[x] < size[y]) swap(x, y);
15     size[x] += size[y];
16     p[y] = x;
17 }
18 }
19
20 int n, m; cin >> n >> m;
21 vector<tuple<long long, int, int>> edges;
22 for (int i = 0; i < m; i++) {
23     int u, v; cin >> u >> v;
24     long long w; cin >> w;
25     edges.emplace_back(w, u, v);
26 }
27 sort(edges.begin(), edges.end());
28 DSU dsu(n);
29 long long mst_cost = 0;
30 for (auto [w, u, v] : edges) {
31     if (dsu.find(u) != dsu.find(v)) {
32         dsu.merge(u, v);
33         mst_cost += w;
34     }
35 }
36 // mst_cost = costo del MST (Minimum Spanning Tree)

```

## 7.10. Lowest Common Ancestor Lca

```

1 struct LCA{
2     int n, l, timer = 0;
3     vector<vector<int>> up, adj;
4     vector<int> depth, in, out;
5
6     LCA(int _n) {
7         n = _n + 1;
8         l = ceil(log2(n));
9         up.resize(n, vector<int>(l + 1)());
10        adj.resize(n);
11        depth.resize(n);
12        in.resize(n);
13        out.resize(n);
14    }
15
16    void add_edge(int p, int u){
17        adj[p].push_back(u);
18        adj[u].push_back(p);
19    }
20
21    void dfs(int u = 1, int p = 1){
22        up[u][0] = p;
23        depth[u] = depth[p] + 1;
24        in[u] = ++timer;
25        for(int level = 1; level <= l; level++){

```

```

19     }
20 }
21 }
22 for (int u = 1; u <= N; u++) {
23     cout << dis[u] << " ";
24 }

```

## 7.7. Disjoint Set Union Dsu

```

1 struct DSU {
2     vector<int> p, size;
3     DSU(int n) {
4         p.resize(n + 1), size.resize(n + 1, 1);
5         for (int i = 1; i <= n; i++) p[i] = i;
6     }
7     int find(int x) {
8         if (p[x] != x) p[x] = find(p[x]);
9         return p[x];
10    }
11
12    void merge(int x, int y) {
13        x = find(x), y = find(y);
14        if (x == y) return;
15        if (size[x] < size[y]) swap(x, y);
16        size[x] += size[y];
17        p[y] = x;
18    }
19 };

```

## 7.8. Floyd Warshall

```

1 int n; cin >> n;
2 vector<vector<long long>> dist(n + 1, vector<long long>(n + 1, inf));
3 for (int i = 1; i <= n; i++) dist[i][i] = 0;
4 int m; cin >> m;
5 for (int i = 0; i < m; i++) {
6     int u, v; cin >> u >> v;
7     long long w; cin >> w;
8     dist[u][v] = min(dist[u][v], w);
9     dist[v][u] = min(dist[v][u], w);
10 }
11 for (int k = 1; k <= n; k++) {
12     for (int i = 1; i <= n; i++) {
13         for (int j = 1; j <= n; j++) {
14             if (dist[i][k] != inf && dist[k][j] != inf) {
15                 dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
16             }
17         }
18     }
19 }
20 // dist[i][j] = distancia minima entre nodo i y nodo j

```

## 7.9. Kruskal

```

1 struct DSU {
2     vector<int> p, size;
3     DSU(int n) {
4         p.resize(n + 1), size.resize(n + 1, 1);
5         for (int i = 1; i <= n; i++) p[i] = i;

```

```

119     return sign(cross(a, b)) == 0;
120 }
121 template <typename T>
122 bool sameDirection(P<T> a, P<T> b) {
123     return sign(cross(a, b)) == 0 && sign(dot(a, b)) == 1;
124 }
125 template <typename T>
126 Real angle(P<T> p) {
127     return atan2((Real)p.y, (Real)p.x);
128 }
129 template <typename T>
130 P<T> rotate90(P<T> p) {
131     return P<T>(-p.y, p.x);
132 }
133 P<Real> rotate(P<Real> p, Real ang) {
134     return P<Real>(p.x * cosl(ang) - p.y * sinl(ang),
135                     p.x * sinl(ang) + p.y * cosl(ang));
136 }
137
138 // ===== Dirección de una línea =====
139 // - direction(l): vector l.l[1] - l.l[0] (dirección del segmento/recta).
140 // - parallel(l1, l2) / sameDirection(l1, l2): igual que para vectores pero con líneas.
141 template <typename T>
142 P<T> direction(L<T> l) {
143     return l.l[1] - l.l[0];
144 }
145 template <typename T>
146 bool parallel(L<T> l1, L<T> l2) {
147     return sameDirection(direction(l1), direction(l2));
148 }
149 template <typename T>
150 bool sameDirection(L<T> l1, L<T> l2) {
151     return sameDirection(direction(l1), direction(l2));
152 }
153
154 // ===== Proyección, reflexión, distancias a recta =====
155 // - projection(p, l): proyección ortogonal de p sobre la recta (infinita) l.
156 // - reflection(p, l): reflejo de p respecto a la recta l.
157 // - pointToLineDist(p, l): distancia mínima de p a la recta infinita que pasa por l.
158 P<Real> projection(P<Real> p, L<Real> l) {
159     auto d = direction(l);
160     return l.l[0] + d * (dot(p - l.l[0], d) / (Real)square(d));
161 }
162 P<Real> reflection(P<Real> p, L<Real> l) {
163     return projection(p, l) * 2 - p;
164 }
165 template <typename T>
166 Real pointToLineDist(P<T> p, L<T> l) {
167     if (l.l[0] == l.l[1]) return dist(p, l.l[0]);
168     return fabsl(cross(l.l[0] - l.l[1], l.l[0] - p)) / length(direction(l));
169 }
170
171 // ===== Intersección de líneas (rectas infinitas) =====
172 // - lineIntersection(l1, l2): punto de intersección de las rectas infinitas
173 // definidas por l1 y l2.
174 // - OJO: no chequea si son paralelas; debes verificar antes que cross != 0.
175 template <typename T>
176 P<Real> lineIntersection(L<T> l1, L<T> l2) {
177     auto d1 = direction(l1);

```

```

178 auto d2 = direction(l2);
179 auto num = (Real)cross(d2, l1.l[0] - l2.l[0]);
180 auto den = (Real)cross(d2, d1);
181 return P<Real>(l1.l[0]) - d1 * (num / den);
182 }

183 // ===== Side / Between =====
184 // - side(p, a, b): orientación de p respecto al vector ab.
185 //   > 0: izquierda (CCW), < 0: derecha (CW), 0: colineal.
186 // - side(p, l): igual que antes pero con línea l.
187 // - between(m, l, r): true si m está entre l y r (incluyendo bordes).
188 template <typename T>
189 int side(P<T> p, P<T> a, P<T> b) {
190     return sign(cross(p, a, b));
191 }
192 template <typename T>
193 int side(P<T> p, L<T> l) {
194     return side(p, l.l[0], l.l[1]);
195 }
196 template <typename T>
197 bool between(T m, T l, T r) {
198     return cmp(l, m) == 0 || cmp(m, r) == 0 || (l < m) != (r < m);
199 }
200 }

201 // ===== Puntos sobre segmento =====
202 // - pointOnSeg(p, l): true si p está sobre el segmento l (incluye endpoints).
203 // - pointStrictlyOnSeg(p, l): true si p está sobre el segmento pero no en los endpoints.
204 template <typename T>
205 bool pointOnSeg(P<T> p, L<T> l) {
206     return side(p, l) == 0 &&
207         between(p.x, l.l[0].x, l.l[1].x) &&
208         between(p.y, l.l[0].y, l.l[1].y);
209 }
210 template <typename T>
211 bool pointStrictlyOnSeg(P<T> p, L<T> l) {
212     if (side(p, l) != 0) return false;
213     auto d = direction(l);
214     return sign(dot(p - l.l[0], d)) * sign(dot(p - l.l[1], d)) < 0;
215 }
216 }

217 // ===== Solapamiento de intervalos =====
218 // - overlap(l1, r1, l2, r2): true si [l1, r1] y [l2, r2] se solapan (1D).
219 template <typename T>
220 bool overlap(T l1, T r1, T l2, T r2) {
221     if (l1 > r1) swap(l1, r1);
222     if (l2 > r2) swap(l2, r2);
223     return cmp(r1, l2) != -1 && cmp(r2, l1) != -1;
224 }

225 // ===== Intersección de segmentos / rayos =====
226 // - segIntersect(l1, l2): true si los segmentos se tocan o cortan
227 //   (incluye colineales solapados y tocar en vértices).
228 // - segStrictlyIntersect(l1, l2): true si se cortan estrictamente
229 //   (no cuenta tocar solo en un endpoint).
230 // - rayIntersect(l1, l2): considera l1 y l2 como rayos, intersectan "hacia adelante"
231 //   (no cuenta si solo coincide el origen).
232 template <typename T>
233 bool segIntersect(L<T> l1, L<T> l2) {
234     auto [p1, p2] = l1.l;

```

```

9
10 for (int u = 1; u <= n; u++) {
11     if (vis[u]) continue;
12     dfs(u);
13 }

```

### 7.5. Dfs 2D

```

1 int N, M; cin >> N >> M;
2 vector<vector<char>> grid(N, vector<char>(M));
3 for (int i = 0; i < N; i++) {
4     for (int j = 0; j < M; j++) {
5         cin >> grid[i][j];
6     }
7 }
8
9 vector<vector<bool>> vis(N, vector<bool>(M));
10 int dx[4] = {-1, 1, 0, 0}, dy[4] = {0, 0, -1, 1};
11 function<void(int, int)> dfs = [&](int x, int y) {
12     vis[x][y] = 1;
13
14     for (int d = 0; d < 4; d++) {
15         int nx = x + dx[d], ny = y + dy[d];
16         if (0 <= nx && 0 <= ny && nx < N && ny < M && grid[nx][ny] == '.' && !vis[nx][ny]) {
17             dfs(nx, ny);
18         }
19     }
20 };
21
22 int comp = 0;
23 for (int i = 0; i < N; i++) {
24     for (int j = 0; j < M; j++) {
25         if (vis[i][j] || grid[i][j] == '#') continue;
26         dfs(i, j);
27         comp++;
28     }
29 }
30
31 cout << comp;

```

### 7.6. Dijkstra

```

1 int N, M; cin >> N >> M;
2 vector<vector<pair<int, long long>>> adj(N + 1);
3 for (int i = 0; i < M; i++) {
4     int u, v; cin >> u >> v;
5     long long w; cin >> w;
6     adj[u].emplace_back(v, w);
7 }
8
9 vector<long long> dis(N + 1, inf);
10 pqg<pair<long long, int>> pq;
11 dis[1] = 0;
12 pq.push({0LL, 1});
13 while (!pq.empty()) {
14     auto [d, node] = pq.top(); pq.pop();
15     if (dis[node] != d) continue;
16     for (auto [v, w] : adj[node]) {
17         if (d + w < dis[v]) {
18             dis[v] = d + w;
19             pq.push({dis[v], v});
20         }
21     }
22 }
23
24 cout << dis[N];

```

```

11     int u = q.front();
12     q.pop();
13     for (int& v : adj[u]) {
14         if (vis[v]) continue;
15         vis[v] = true;
16         q.push(v);
17     }
18 }
19 }
20 };
21 for (int u = 1; u <= n; u++) {
22     if (vis[u]) continue;
23     bfs(u);
24 }
25 }
```

### 7.3. Bipartite

```

1 int N, M; cin >> N >> M;
2 vector<vector<int>> adj(N + 1);
3 while (M--) {
4     int u, v; cin >> u >> v;
5     adj[u].push_back(v);
6     adj[v].push_back(u);
7 }
8
9 vector<bool> vis(N + 1);
10 vector<int> col(N + 1, 0);
11 // bipartite graph
12 function<bool(int, int)> dfs = [&](int u, int c) {
13     vis[u] = 1;
14     col[u] = c;
15
16     for (auto v : adj[u]) {
17         if (vis[v] && col[u] == col[v]) return false;
18         else if (!vis[v] && !dfs(v, c ^ 1)) return false;
19     }
20     return true;
21 };
22
23 for (int i = 1; i <= N; i++) {
24     if (vis[i]) continue;
25     if (dfs(i, 1) == false) {
26         cout << "IMPOSSIBLE";
27         return;
28     }
29 }
30
31 for (int i = 1; i <= N; i++) cout << (col[i] ? 1 : 2) << 'u';

```

### 7.4. Dfs

```

1 vector<bool> vis(n+1);
2 function<void(int)> dfs = [&](int u) {
3     vis[u] = true;
4     for (int& v : adj[u]) {
5         if (vis[v]) continue;
6         dfs(v);
7     }
8 };

```

```

237     auto [q1, q2] = 12.1;
238     return overlap(p1.x, p2.x, q1.x, q2.x) &&
239         overlap(p1.y, p2.y, q1.y, q2.y) &&
240         side(p1, 12) * side(p2, 12) <= 0 &&
241         side(q1, 11) * side(q2, 11) <= 0;
242 }
243 template <typename T>
244 bool segStrictlyIntersect(L<T> l1, L<T> l2) {
245     auto [p1, p2] = l1.1;
246     auto [q1, q2] = l2.1;
247     return side(p1, l2) * side(p2, l2) < 0 &&
248         side(q1, l1) * side(q2, l1) < 0;
249 }
250 template <typename T>
251 bool rayIntersect(L<T> l1, L<T> l2) {
252     auto v1 = direction(l1);
253     auto v2 = direction(l2);
254     int x = sign(cross(v1, v2));
255     return x != 0 && side(l1.1[0], l2) == x && side(l2.1[0], l1) == -x;
256 }
257
258 // ===== Distancias punto-segmento / segmento-segmento =====
259 // - pointToSegDist(p, l): distancia mínima de p al segmento l.
260 // - segDist(l1, l2): distancia mínima entre dos segmentos (0 si se intersectan).
261 template <typename T>
262 Real pointToSegDist(P<T> p, L<T> l) {
263     auto d = direction(l);
264     if (sign(dot(p - l.1[0], d)) >= 0 && sign(dot(p - l.1[1], d)) <= 0) {
265         return pointToLineDist(p, l);
266     } else {
267         return min(dist(p, l.1[0]), dist(p, l.1[1]));
268     }
269 }
270 template <typename T>
271 Real segDist(L<T> l1, L<T> l2) {
272     if (segIntersect(l1, l2)) return 0;
273     return min(
274         pointToSegDist(l1.1[0], l2),
275         pointToSegDist(l1.1[1], l2),
276         pointToSegDist(l2.1[0], l1),
277         pointToSegDist(l2.1[1], l1)
278     );
279 }
280
281 // ===== Área de polígono y punto en polígono =====
282 // - area(a): devuelve 2 * área con signo del polígono a (ordenado).
283 // >0 CCW, <0 CW, abs(area)/2.0 área real.
284 // - pointInPoly(p, a): true si p está dentro o sobre el borde del polígono a
285 // (no necesariamente convexo).
286 template <typename T>
287 T area(vector<P<T>> a) {
288     T res = 0;
289     int n = (int)a.size();
290     for (int i = 0; i < n; i++) {
291         res += cross(a[i], a[(i + 1) % n]);
292     }
293     return res;
294 }
295 template <typename T>
```

```

296 bool pointInPoly(P<T> p, vector<P<T>> a) {
297     int n = (int)a.size(), res = 0;
298     for (int i = 0; i < n; i++) {
299         P<T> u = a[i], v = a[(i + 1) % n];
300         if (pointOnSeg(p, L<T>(u, v))) return true;
301         if (cmp(u.y, v.y) <= 0) swap(u, v);
302         if (cmp(p.y, u.y) > 0 || cmp(p.y, v.y) <= 0) continue;
303         res += cross(p, u, v) > 0;
304     }
305     return res;
306 }
307 // ===== Aliases finales =====
308 // - Point = P<i64>    puntos con coordenadas enteras.
309 // - Line = L<i64>    segmentos/líneas con endpoints enteros.
310 // - Usa Real (long double) para distancias si necesitas precisión extra.
311 using Point = P<i64>;
312 using Line = L<i64>;
313
314 // ejemplo de uso rápido:
315 // Point a, b; cin >> a >> b;
316 // Line seg(a, b);
317 // if (segIntersect(seg, Line(Point(0,0), Point(10,0)))) { ... }

```

## 6.2. Convex Hull

```

1 struct Point {
2     long long x, y;
3     Point(long long x = 0, long long y = 0) : x(x), y(y) {}
4     Point operator-(const Point& p) const { return Point(x - p.x, y - p.y); }
5     long long cross(const Point& p) const { return x * p.y - y * p.x; }
6     long long cross(const Point& a, const Point& b) const { return (a - *this).cross(b - *this)
7         ); }
8     bool operator<(const Point& p) const { return x < p.x || (x == p.x && y < p.y); }
9     bool operator==(const Point& p) const { return x == p.x && y == p.y; }
10 };
11
12 vector<Point> convex_hull(vector<Point>& points) {
13     int n = points.size();
14     if (n <= 1) return points;
15     sort(points.begin(), points.end());
16     vector<Point> hull;
17     for (int i = 0; i < n; i++) {
18         // Si quiero incluir coords colineales (< 0), si no lo quiero (<= 0)
19         while (hull.size() >= 2 && hull[hull.size() - 2].cross(hull.back(), points[i]) <= 0)
20             hull.pop_back();
21         hull.push_back(points[i]);
22     }
23     int lower = hull.size();
24     for (int i = n - 2; i >= 0; i--) {
25         // Si quiero incluir coords colineales (< 0), si no lo quiero (<= 0)
26         while (hull.size() > lower && hull[hull.size() - 2].cross(hull.back(), points[i]) <= 0)
27             {
28                 hull.pop_back();
29             }
30         hull.push_back(points[i]);
31     }

```

```

31     hull.pop_back();
32     return hull;
33 }
34
35 // uso:
36 // int n; cin >> n;
37 // vector<Point> points(n);
38 // for (int i = 0; i < n; i++) {
39 //     cin >> points[i].x >> points[i].y;
40 // }
41 // vector<Point> hull = convex_hull(points);
42 // hull contiene los puntos del convex hull en orden counter-clockwise (CCW / Antihorario)
43 // reverse(hull) para obtenerlo en orden CW / horario

```

## 7. Graph

Algoritmos de grafos: DFS, BFS, componentes fuertemente conexas, y otras estructuras de datos para problemas de grafos.

### 7.1. Bellman Ford

```

1 int n, m; cin >> n >> m;
2 vector<tuple<int, int, long long>> edges;
3 for (int i = 0; i < m; i++) {
4     int u, v; cin >> u >> v;
5     long long w; cin >> w;
6     edges.emplace_back(u, v, w);
7 }
8 vector<long long> dist(n + 1, inf);
9 dist[1] = 0;
10 for (int i = 0; i < n - 1; i++) {
11     for (auto [u, v, w] : edges) {
12         if (dist[u] != inf) {
13             dist[v] = min(dist[v], dist[u] + w);
14         }
15     }
16 }
17 bool has_negative_cycle = false;
18 for (auto [u, v, w] : edges) {
19     if (dist[u] != inf && dist[u] + w < dist[v]) {
20         has_negative_cycle = true;
21         break;
22     }
23 }
24 // dist[u] = distancia mínima desde nodo 1 hasta u
25 // has_negative_cycle = true si hay ciclo negativo alcanzable desde nodo 1

```

### 7.2. Bfs

```

1 vector<bool> vis(n+1);
2 queue<int> q;
3 function<void(int)> bfs = [&](int start) {
4     vis[start] = true;
5     q.push(start);
6     int levels = 1;
7     while (!q.empty()) {
8         int sz = q.size();
9         levels++;
10        while (sz--) {

```