

University of São Paulo
Institute of Mathematics and Statistics
USP Try-Outs 2025

Editorial

Authors: Pedro Tsuchie, Guilherme Kavakami, and Murilo Kataoka

Problem Explanations

A. Yuyuan Market

Tags: Dynamic Programming

Before introducing the DP, let us make one key observation. Suppose we select a pair whose first occurrence is at position L and whose second occurrence is at position R . We are then forced to select no element between L and R . Moreover, any choices made before L do not interfere with this interval. Therefore, whenever we select this pair, we only care about the best state achievable up to position $L - 1$.

This leads directly to the following transition. If position i is the second occurrence of a symbol—that is, $i = R$ for a pair beginning at L —we choose the better option between ignoring the current element and selecting its pair. In the latter case, we add the pair's contribution to the best result available before L :

$$DP[i] = \max(DP[i - 1], DP[L - 1] + \text{current_pair_gain}).$$

We want to maximize the number of selected pairs and, among solutions with the same number of pairs, minimize the total distance cost. We can therefore store each DP state as a pair (number of pairs, cost). The final transition becomes

$$DP[i] = \max(DP[i - 1], DP[L - 1] + (1, R - L)),$$

where the comparison first favors a larger number of pairs and then a smaller cost.

Each position is processed in $\mathcal{O}(1)$ time, giving an overall time complexity of $\mathcal{O}(N)$. This is easily fast enough for the one-second time limit with $N \leq 10^5$.

B. The Search for Balance

Tags: Meet in the Middle, Line Sweep

Before discussing the solution itself, it is worth paying attention to the number 35. It is too large for a complete *brute-force* search, since 2^{35} is greater than 10^{10} , yet small enough that the usual polynomial approaches are unlikely to fit the structure of the problem. Whenever we encounter this kind of constraint, **Meet in the Middle** becomes a strong candidate. After all, if the limit were slightly smaller and we could enumerate

every combination directly, the problem would be straightforward: generate all possible states and count those that satisfy the required conditions. If this technique is new to you, this classic CSES problem is an excellent place to start.

We split the power-ups into two halves of sizes 17 and 18, then generate every possible (Attack, Defense) pair obtainable from each half. The problem is now the following: for each pair generated from the first half, determine how many pairs from the second half make the combined attributes fall inside both target intervals, $[A_1, A_2]$ and $[D_1, D_2]$.

In practice, this is the classic problem of **counting points inside a two-dimensional rectangle**. We cannot scan the entire second list for every state from the first half, as that would result in *Time Limit Exceeded*. Instead, we use an efficient data structure. One option is a **line sweep** over the attack coordinate combined with a **Fenwick Tree (BIT)** over the defense coordinate. Another is a **Merge Sort Tree**. Both approaches count the points inside each rectangle in logarithmic time, which is enough to solve the problem.

C. Echoes of the Jade Library

Tags: Eertree, Persistent Segment Tree

This problem resembles the CSES problem Distinct Values Queries: we must answer Q distinct-value queries over intervals, except that our objects are palindromes rather than numbers. In addition, the queries must be answered online, so we are interested in the online solution to the CSES problem, which uses a Persistent Segment Tree.

To represent palindromes as integers, we maintain one global Eertree (Palindromic Tree), inserting separators between consecutive input strings. Whenever a character is inserted, the Eertree gives us the state corresponding to the longest palindromic suffix created at that step. The identifier of that state serves as the integer associated with the palindrome. As a result, each input string is represented by a list of palindrome identifiers.

We can now apply the same idea used in Distinct Values Queries. First consider all queries with $L = 1$. Build an array v such that v_i is the number of palindromes appearing for the first time at position i . Each palindrome is counted only at its first occurrence. A sum Segment Tree can then answer every query with $L = 1$ in $\mathcal{O}(\log N)$ time.

How do we handle larger values of L ? When L increases by one, every palindrome whose first valid occurrence was at the previous position L must now be counted at the next position i where it appears. We can therefore create a new Segment Tree root representing $L = 2$ and update only the nodes whose values change. Across the entire construction, this produces $\mathcal{O}(M \log N)$ updates, where M is the total number of characters in the input.

Repeating this process for every $L > 1$ gives us a Persistent Segment Tree capable of answering the desired range-sum queries. To answer a query $[L, R]$, we query the sum over $[L, R]$ using the root associated with L . The result is exactly the number of distinct palindromes appearing in that interval.

Building the Persistent Segment Tree costs $\mathcal{O}(M \log N)$, while answering all queries costs $\mathcal{O}(Q \log N)$. Therefore, the total complexity is

$$\mathcal{O}((M + Q) \log N).$$

D. The Seals of Shanghai

Tags: Number Theory, Ad Hoc

Before thinking about the simulation, let us establish an upper bound on the answer. What is the maximum number of rounds Lian could need in the worst case? If we choose $M = 2$, every number has remainder either 0 or 1. Thus, even in the worst case, all seals are divided into at most two remainder groups. This gives us a mathematical guarantee: **the minimum number of rounds is always either 1 or 2.**

When is the answer 1?

For all seals to be opened in a single round, every array element must have the same remainder modulo M . In other words,

$$A_i \equiv A_1 \pmod{M},$$

which means that M must divide $|A_i - A_1|$ for every i .

Therefore, we compute the greatest common divisor of all differences $|A_i - A_1|$. If this gcd is greater than 1, the answer is one round, and the number of valid values of M is simply the number of divisors of the gcd, excluding 1. There is one important corner case: if all array elements are equal, then the gcd is 0 and every M is valid, giving $10^9 - 1$ possibilities.

When is the answer 2?

If the gcd is 1, solving the problem in one round is impossible. We already know that the answer is two rounds, but how can we test values of M up to 10^9 without running into *Time Limit Exceeded*? This is where the **Pigeonhole Principle** comes in.

If the optimal answer is two rounds, then the chosen value of M splits the array into exactly two remainder groups. Consider only the **first three distinct values** in the array, v_1 , v_2 , and v_3 . We have three elements—the pigeons—but only two possible remainder groups—the holes. Therefore, **at least two of these values must have the same remainder.**

Consequently, M must give the same remainder to v_1 and v_2 , **or** to v_1 and v_3 , **or** to v_2 and v_3 . Hence, M must divide at least one of

$$|v_1 - v_2|, \quad |v_1 - v_3|, \quad |v_2 - v_3|.$$

That reduces an enormous search space to a very small set of divisors. We only need to simulate the process for those candidates.

E. Wise Partitioning Method

Tags: Divide and Conquer, Merge Sort Tree, Binary Search

The first idea might be to simulate the algorithm step by step, but that takes $\mathcal{O}(N^2)$ time in the worst case.

Step 1: Understanding the relative order

The key lies in the way Master Wei implemented the **pivot** function. Consider how elements are separated relative to the pivot x :

- If an element is smaller, $y < x$, it is placed at the beginning and the left pointer advances. Therefore, the smaller elements **preserve their original relative order exactly**.
- If an element is larger, $y > x$, it is placed at the end and the right pointer moves backward. The first larger element goes to the final position, the second goes to the second-to-last position, and so on. Thus, the larger elements **have their relative order completely reversed**.

Step 2: Recursion

Because Quicksort perfectly separates smaller values from larger ones, every recursive call works on a contiguous interval of values. If we imagine the array sorted by value, we can represent such an interval as $[L, R]$.

But in which physical order do those elements appear during the current recursive call? That depends only on how many times we have moved into a right child in the recursion tree, which we call the number of *flips*:

- If the number of *flips* is **even**, the reversals cancel out, and the elements appear in their **original order**, based on their initial indices.
- If the number of *flips* is **odd**, the elements appear in the **reverse of their original order**.

Step 3: Determining the pivot

The algorithm physically selects the person sitting in the middle position:

$$m = (l + r) // 2.$$

Since we know the number of elements in the current interval, sz , and whether their order is normal or reversed, we know exactly which position in the original ordering corresponds to the middle seat. The pivot is therefore the element with the **K -th smallest original index** among the values in $[L, R]$.

Step 4: Merge Sort Tree

To find that element efficiently, we build a **Merge Sort Tree** over the values in sorted order. Each tree node covers an interval of values and stores a sorted vector containing only the **original indices** of the elements in that interval.

We can then binary-search an original index X and ask the tree:

Among the values in $[L, R]$, how many elements originally appeared at an index $\leq X$?

The tree answers this query in $\mathcal{O}(\log^2 N)$ time using `upper_bound`.

Once we find the pivot's original index, an auxiliary array, `pos_ordenada`, tells us its rank by value. This rank lets us split the current interval $[L, R]$ into $[L, \text{Rank} - 1]$ for the smaller values and $[\text{Rank} + 1, R]$ for the larger values. The total time complexity is

$$\mathcal{O}(N \log^3 N).$$

F. Chinese Innovation

Tags: Dijkstra's Algorithm

Without teleporters, the problem is simply the classic shortest-path problem from vertex 1 to vertex N , which can be solved with Dijkstra's algorithm.

To incorporate teleporters, notice that we cannot add an edge between every pair of vertices sharing a teleporter of type u_i , since this could create $\mathcal{O}(N^2)$ edges in the worst case. Instead, we introduce an auxiliary vertex representing access to each teleporter type u_i . If a vertex v has access to teleporter type u_i at cost c_i , we add a directed edge from v to the auxiliary vertex u_i with cost c_i , and another directed edge from u_i back to v with cost 0.

The resulting graph has $\mathcal{O}(N + K)$ vertices and $\mathcal{O}(M + S)$ edges, where

$$S \leq \min(NK, 2 \cdot 10^5)$$

is the total number of teleporter accesses across all cities. We can therefore run Dijkstra's algorithm on this expanded graph. The auxiliary vertex corresponding to teleporter type u_i can be stored at index $N + u_i$.

G. Incompatible Pairs

Tags: Ad Hoc

Split the string into its smallest balanced blocks: whenever the parenthesis balance returns to zero, the current block ends. For example, " $((())())$ " is divided into " $((())())$ " and " $()$ ".

When is a pair incompatible? Consider an opening parenthesis at position i . If we match it with a closing parenthesis from the same block, all remaining parentheses in that block can still be matched normally, so the pair is compatible. On the other hand, if we choose a closing parenthesis from a later block, the block containing the opening parenthesis is left with an unmatched closing parenthesis, making it impossible to match all remaining parentheses.

Therefore, for every position i such that $s_i = '('$, we only need to count the positions j such that $s_j = ')'$ and j belongs to a block after the block containing i . Prefix sums allow us to compute this value efficiently. The total complexity is $\mathcal{O}(N)$.

H. The Wisdom of Master Wei

Tags: Mathematics

Given W and K , we need to find an integer X such that $X - W$ is twice $X - K$. Therefore,

$$X - W = 2(X - K) \iff X = 2K - W.$$

We only need to print $2K - W$.

I. Bath

Tags: Geometry, Floyd–Warshall

We must find the shortest path from the starting point to a target point inside the polygon, while staying entirely within the polygon. Observe that a shortest path between these two points consists of straight-line segments whose endpoints are polygon vertices or the target point.

Suppose that, for every pair of relevant points (u, v) , we know whether the segment connecting them lies entirely inside the polygon and, when it does, we know its length. We can then use the Floyd–Warshall algorithm to compute the shortest distance from the starting point to the target point.

For each pair (u, v) , let cur be the segment connecting them. We reject cur if there is a polygon edge $other$, neither of whose endpoints is u or v , such that cur intersects $other$. If such an intersection exists, the segment cannot be used.

Otherwise, cur lies either entirely inside or entirely outside the polygon. We can choose any point on cur —for example, its midpoint—and test whether that point lies inside the polygon. If it lies outside, the segment is invalid. If it lies inside, the segment is valid and its length is easy to compute. Checking all pairs in this way takes $\mathcal{O}(N^3)$ time.

Finally, we run Floyd–Warshall and multiply the resulting distance by 2.

J. The Messenger’s Disguise

Tags: Dynamic Programming

Let $dist[i]$ denote the minimum distance from S to vertex i . We begin by defining the DP used throughout the solution:

$$\begin{cases} DP[i][0] = \text{number of ways to reach } i \text{ using } dist[i] \text{ edges,} \\ DP[i][1] = \text{number of ways to reach } i \text{ using } dist[i] + 1 \text{ edges.} \end{cases}$$

Before deriving the transitions, divide the neighbors of a vertex i into two types:

- **Type 1:** neighbors v such that $dist[v] = dist[i]$;
- **Type 2:** neighbors v such that $dist[v] + 1 = dist[i]$.

We can now analyze each DP state.

- $DP[i][0]$: Reaching i with the minimum possible number of edges can only happen through a Type 2 neighbor. We reach such a neighbor v using $dist[v]$ edges, traverse the edge from v to i , and arrive using

$$dist[v] + 1 = dist[i]$$

edges. Therefore,

$$DP[i][0] = \sum_{\substack{v \sim i \\ dist[v] + 1 = dist[i]}} DP[v][0].$$

- $DP[i][1]$: To reach i using exactly $dist[i] + 1$ edges, the previous vertex may be of either type:
 - From a Type 1 neighbor v , we follow a shortest path to v and then use the edge (v, i) . Since $dist[v] = dist[i]$, the resulting path has $dist[i] + 1$ edges.
 - From a Type 2 neighbor v , we use a path that reaches v with $dist[v] + 1$ edges and then traverse (v, i) . The total is

$$dist[v] + 1 + 1 = dist[i] + 1.$$

Hence,

$$DP[i][1] = \sum_{\substack{v \sim i \\ dist[v] = dist[i]}} DP[v][0] + \sum_{\substack{v \sim i \\ dist[v] + 1 = dist[i]}} DP[v][1].$$

The answer is simply $DP[T][1]$.

Implementation details

We must be careful about the order in which the DP states are computed. The state of a vertex depends both on vertices in the previous BFS layer and on vertices in the same layer. It is therefore natural to process the graph using BFS.

The first BFS computes the distances and all values of $DP[i][0]$. However, the BFS order does not guarantee that every Type 1 neighbor of i has already been processed when we need to compute $DP[i][1]$. For this reason, we run a second BFS after the first one and use it to compute the second DP state.

We must also account for multiple edges. Since the time limit is tight, the implementation sorts each adjacency list and processes all repeated occurrences of the same neighbor at once. The final complexity is $\mathcal{O}(2N + N \log N)$.

K. Cake Hater

Tags: Marking Array

For this problem, we maintain a marking array indicating which ingredients Antônio dislikes. Then, for each recipe, we inspect its ingredients and count how many of them are disliked. As soon as this count becomes greater than one third of the number of ingredients in the recipe, we have found the answer.

L. Game of Life

Tags: Bitwise Operations, Matrix Exponentiation

The first major observation is that the rules determining whether a cell dies or remains alive can be simplified considerably. The next state of cell (i, j) can be written as

$$\begin{aligned} S'_{i,j} = & S_{i,j} \oplus S_{i-1,j-1} \oplus S_{i-1,j} \oplus S_{i-1,j+1} \\ & \oplus S_{i,j-1} \oplus S_{i,j+1} \\ & \oplus S_{i+1,j-1} \oplus S_{i+1,j} \oplus S_{i+1,j+1}. \end{aligned}$$

In other words, the next state of $S_{i,j}$ is the XOR of its current value and the values of all its neighbors.

To perform all K iterations within the time limit, we represent this transition as a matrix multiplication. The only difference from ordinary matrix multiplication is that, instead of summing the products, we take their XOR. We can then use fast matrix exponentiation to compute the final state of the board:

$$S \times T^K = R,$$

where S is the vectorized initial board, T is the transition matrix, K is the number of iterations, and R is the final board.

To vectorize the board, place the value in row i and column j at position $iN + j$ of S . The initial board therefore becomes a $1 \times N^2$ matrix, as does the resulting matrix R .

The transition matrix T has dimensions $N^2 \times N^2$. To determine the next value of the j -th entry of S , we multiply S by the j -th column of T . Since the new state of one cell is the XOR of that cell and all its neighbors, the j -th column contains 1 in the positions corresponding to the cell itself and its neighbors, and 0 everywhere else.

For example, when $N = 3$, the column corresponding to cell 0 is

$$\begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

If

$$S = [0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0],$$

then the next state of cell 0 is

$$\begin{aligned} & (S_0 T_{0,0}) \oplus (S_1 T_{1,0}) \oplus (S_2 T_{2,0}) \oplus (S_3 T_{3,0}) \oplus (S_4 T_{4,0}) \oplus \cdots \oplus (S_8 T_{8,0}) \\ &= (0 \cdot 1) \oplus (1 \cdot 1) \oplus (1 \cdot 0) \oplus (0 \cdot 1) \oplus (1 \cdot 1) \oplus \cdots \oplus (0 \cdot 0) = 0. \end{aligned}$$

Cells marked with # require one final adjustment. We set their values to 0 in the vectorized board and leave their corresponding columns in T entirely empty, ensuring that they remain blocked throughout the process.

M. Nomad

Tags: Ad Hoc

We divide the problem into three cases:

- **Every $d_i = 0$:** the nomad cannot stay anywhere.
- **Exactly one $d_i \neq 0$:** the only possible choice is to remain at that location.
- **More than one $d_i \neq 0$:** the nomad can stay for an unlimited number of days by alternating between any two locations whose values of d_i are nonzero.

Implementations

Code A - Yuyuan Market

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4
5  int main() {
6      ios_base::sync_with_stdio(false);
7      cin.tie(NULL);
8      int n;
9      cin >> n;
10
11     vector<ll> nums(2*n+1);
12     vector<ll> jafoi(2*n+1,0);
13     vector<ll> pos(2*n+1,-1);
14
15     for(int i=1; i<=2*n; i++) {
16         cin >> nums[i];
17         if(jafoi[nums[i]]) {
18             pos[i] = jafoi[nums[i]];
19         }
20         else {
21             jafoi[nums[i]] = i;
22         }
23     }
24
25     vector<pair<ll,ll>> dp(2*n+1, {0ll,0ll});
26
27     for(int i=1; i<=2*n; i++) {
28         if(pos[i] == -1) {
29             dp[i] = dp[i-1];
30         }
31         else {
32             pair<ll,ll> cand;
33             cand.first = dp[pos[i]-1].first + 1;
34             cand.second = dp[pos[i]-1].second + i - pos[i];
35
36             if(cand.first > dp[i-1].first) {
37                 dp[i] = cand;
38             }
39             else if(cand.first == dp[i-1].first) {
40                 dp[i] = min(dp[i-1], cand);
41             }
42             else {
43                 dp[i] = dp[i-1];
44             }
45         }
```

```

46     }
47
48     cout << dp[2*n].first << " " << dp[2*n].second << '\n';
49     return 0;
50 }

```

Code B - The Search for Balance

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  typedef long long ll;
5  typedef long double ld;
6  #define all(x) (x).begin(),(x).end()
7  struct Event {
8      ll x, y1, y2;
9      int type; // 0 = point from the second half, 1 = right boundary, -1 = left boundary
10     bool operator<(const Event& o) const {
11         if (x != o.x) return x<o.x;
12         return abs(type)<abs(o.type);
13     }
14 };
15 struct BIT {
16     int n;
17     vector<int> tree;
18     BIT(int n) : n(n), tree(n + 1, 0) {}
19
20     void update(int i, int val) {
21         for (; i<=n; i+=(i&-i)) tree[i]+=val;
22     }
23     int query(int i) {
24         int sum = 0;
25         for (; i > 0; i -= (i & -i)) sum += tree[i];
26         return sum;
27     }
28 };
29 vector<pair<ll,ll>> ResolveMetade(vector<pair<ll,ll>>&poderes){
30     int tam=poderes.size();
31     vector<pair<ll,ll>>possibilidades;
32     for(int i=0;i<(1<<tam);i++){
33         ll ataque=0,defesa=0;
34         for(int bit=0;bit<tam;bit++){
35             if(i>>bit&1){
36                 ataque+=poderes[bit].first;
37                 defesa+=poderes[bit].second;
38             }
39         }
40         possibilidades.push_back({ataque,defesa});

```

```

41     }
42     return posibilidades;
43 }
44 int main()
45 {
46     ios_base::sync_with_stdio(false); cin.tie(NULL);
47     int n;
48     cin>>n;
49     vector<pair<ll,ll>>poderes(n); // {attack, defense}
50     vector<pair<ll,ll>>metade1;
51     vector<pair<ll,ll>>metade2;
52     for(int i=0;i<n;i++)
53     {
54         cin>>poderes[i].first>>poderes[i].second;
55         if(i<=n/2){
56             metade1.push_back(poderes[i]);
57         }
58         else metade2.push_back(poderes[i]);
59     }
60     ll A1, D1, A2, D2;
61     cin >> A1 >> D1 >> A2 >> D2;
62     vector<pair<ll,ll>>Metade1=ResolveMetade(metade1);
63     vector<pair<ll,ll>>Metade2=ResolveMetade(metade2);
64     vector<Event>eventos;
65     vector<ll>ys;
66     for(auto p:Metade2){
67         eventos.push_back({p.first,p.second,0,0});
68         ys.push_back(p.second);
69     }
70     for(auto p : Metade1){
71         ll a = p.first, d = p.second;
72         ll left_x=A1 - a - 1;
73         ll right_x = A2 - a;
74         ll bottom_y = D1 - d;
75         ll top_y = D2 - d;
76
77         eventos.push_back({left_x, bottom_y, top_y, -1}); // Left boundary (subtract)
78         eventos.push_back({right_x, bottom_y, top_y, 1}); // Right boundary (add)
79
80         ys.push_back(bottom_y);
81         ys.push_back(top_y);
82     }
83     // Coordinate compression
84     sort(ys.begin(), ys.end());
85     ys.erase(unique(ys.begin(), ys.end()), ys.end());
86     auto get_id = [&](ll y) {
87         return lower_bound(ys.begin(), ys.end(), y) - ys.begin() + 1;
88     };
89     sort(all(eventos));

```

```

90     BIT bit(ys.size());
91     ll resp = 0;
92
93     for(auto ev : eventos) {
94         if (ev.type == 0) {
95             // Insert the point's defense value into the BIT
96             bit.update(get_id(ev.y1), 1);
97         } else {
98             int id1 = get_id(ev.y1);
99             int id2 = get_id(ev.y2);
100             ll count = bit.query(id2) - bit.query(id1 - 1);
101             // Add or subtract depending on the boundary (right or left)
102             resp += ev.type * count;
103         }
104     }
105
106     cout << resp << "\n";
107
108     return 0;
109 }

```

Code C - Echoes of the Jade Library

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  #define ll int
5  const int MAXN = 1e5+10;
6  const int MAXAO = 7e5+10;
7  const int MAX = 1e5+10, UPD = 1e5+10, LOG = 18;
8  const int MAXS = 3*MAX+3*UPD*LOG;
9
10 int t[MAXAO][28];
11 int s[MAXAO], len[MAXAO], suf[MAXAO];
12
13 struct eertree {
14     int n, last, sz;
15
16     eertree(int N) {
17         s[0] = -1;
18         suf[0] = 1, len[0] = 0, suf[1] = 1, len[1] = -1;
19         sz = 2, last = 0, n = 1;
20     }
21
22     void add(char c) {
23         s[n++] = c -= 'a';
24         while (s[n-len[last]-2] != c) last = suf[last];
25         if (!t[last][c]) {

```

```

26         int prev = suf[last];
27         while (s[n-len[prev]-2] != c) prev = suf[prev];
28         suf[sz] = t[prev][c];
29         len[sz] = len[last]+2;
30         t[last][c] = sz++;
31     }
32     last = t[last][c];
33 }
34 };
35
36 namespace perseg {
37     ll seg[MAXS];
38     int rt[UPD], L[MAXS], R[MAXS], cnt, t;
39     int n, *v;
40
41     ll build(int p, int l, int r) {
42         if (l == r) return seg[p] = v[l];
43         L[p] = cnt++, R[p] = cnt++;
44         int m = (l+r)/2;
45         return seg[p] = build(L[p], l, m) + build(R[p], m+1, r);
46     }
47     void build(int n2, int* v2) {
48         n = n2, v = v2;
49         rt[0] = cnt++;
50         build(0, 0, n-1);
51     }
52     ll query(int a, int b, int p, int l, int r) {
53         if (b < l or r < a) return 0;
54         if (a <= l and r <= b) return seg[p];
55         int m = (l+r)/2;
56         return query(a, b, L[p], l, m) + query(a, b, R[p], m+1, r);
57     }
58     ll query(int a, int b, int tt) {
59         return query(a, b, rt[tt], 0, n-1);
60     }
61     ll update(int a, int x, int lp, int p, int l, int r) {
62         if (l == r) return seg[p] = seg[lp]+x;
63         int m = (l+r)/2;
64         if (a <= m)
65             return seg[p] = update(a, x, L[lp], L[p]=cnt++, l, m) + seg[R[p]=R[lp]];
66         return seg[p] = seg[L[p]=L[lp]] + update(a, x, R[lp], R[p]=cnt++, m+1, r);
67     }
68     int update(int a, int x, int tt=t) {
69         int old = rt[tt];
70         rt[++t] = cnt++;
71         update(a, x, old, rt[t], 0, n-1);
72         return t;
73     }
74     int cur_update(int a, int x, int tt=t){

```

```

75         int new_root = cnt++;
76         update(a, x, rt[tt], new_root, 0, n-1);
77         rt[tt] = new_root;
78         return tt;
79     }
80 };
81
82
83 bool existe[MAXN];
84 int raiz[MAXN], tempo[MAXAO];
85 vector<int> prox[MAXN];
86
87 int32_t main(){
88     ios_base::sync_with_stdio(false); cin.tie(NULL);
89     int n, m; cin >> n >> m;
90
91     for(int i=0; i<MAXAO; ++i) tempo[i] = -1;
92     eertree et(5e5 + 2e5+10);
93
94     vector<int> zero(n, 0);
95     perseg::build(n, zero.data());
96
97     for(int i=0; i<n; ++i){
98         string s; cin >> s;
99         int qtd = 0;
100        for(int j=0; j<s.size(); ++j){
101            char c = s[j];
102            et.add(c);
103
104            if(tempo[et.last]==-1){
105                ++qtd;
106            }
107            if(tempo[et.last]!=i){
108                if(tempo[et.last]!=-1) prox[tempo[et.last]].push_back(i);
109                tempo[et.last] = i;
110            }
111        }
112
113        if(!existe[0]) existe[0] = 1, raiz[0] = perseg::update(i, qtd);
114        else raiz[0] = perseg::cur_update(i, qtd);
115
116        et.add('z'+1); et.add('z'+2);
117    }
118
119    int r = raiz[0];
120    for(int i=0; i<n-1; ++i){
121        for(int nxt:prox[i]){
122            if(nxt!=-1){
123                if(!existe[i+1]) existe[i+1] = 1, r = perseg::update(nxt, 1);
124                else r = perseg::cur_update(nxt, 1);

```

```

124         }
125     }
126     raiz[i+1] = r;
127 }
128
129 while(m--){
130     int l, r; cin >> l >> r;
131     --l, --r;
132     cout << perseg::query(l, r, raiz[l]) << endl;
133 }
134 }

```

Code D - The Seals of Shanghai

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  void add_divisors(ll dif,set<ll>&divs)
5  {
6      if(dif==0)return;
7      for(ll i=1;i*i<=dif;i++)
8      {
9          if(dif%i==0)
10         {
11             if(i>1)divs.insert(i);
12             if(dif/i>1)divs.insert(dif/i);
13         }
14     }
15 }
16 int main()
17 {
18     ios_base::sync_with_stdio(false);
19     cin.tie(NULL);
20     int n;
21     cin>>n;
22     vector<ll>a(n);
23     for(int i=0;i<n;i++)
24     {
25         cin>>a[i];
26     }
27     ll g=0;
28     for(int i=1;i<n;i++)
29     {
30         g=gcd(g,abs(a[i]-a[0]));
31     }
32     if(g==0)// Only when all elements are equal
33     {
34         cout<<1<<" "<<(ll)1e9-1<<"\n";

```

```

35         return 0;
36     }
37     if(g>1)
38     {
39         ll qtd=0;
40         for(ll i=1;i*i<=g;i++)
41         {
42             if(g%i==0)
43             {
44                 if(i>1)qtd++;
45                 if(g/i!=i&&(g/i)>1)qtd++;
46             }
47         }
48         cout<<1<<" "<<qtd<<'\n';
49         return 0;
50     }
51     vector<ll>uni;
52     for(int i=0;i<n;i++)
53     {
54         bool novo=true;
55         for(ll u:uni)
56         {
57             if(u==a[i])
58             {
59                 novo=false;
60                 break;
61             }
62         }
63         if(novo)
64         {
65             uni.push_back(a[i]);
66             if(uni.size()==3)break;
67         }
68     }
69     if(uni.size()==2)
70     {
71         ll dif=abs(uni[0]-uni[1]);
72         set<ll>divs;
73         add_divisors(dif,divs);
74         ll ans=((ll)1e9-1)-divs.size();
75         cout<<2<<" "<<ans<<'\n';
76         return 0;
77     }
78     set<ll>cand;
79     add_divisors(abs(uni[0]-uni[1]),cand);
80     add_divisors(abs(uni[0]-uni[2]),cand);
81     add_divisors(abs(uni[1]-uni[2]),cand);
82     ll resp=0;
83     for(ll m:cand)

```

```

84     {
85         ll r1=a[0]%m;
86         ll r2=-1;
87         bool ok=true;
88         for(int i=1;i<n;i++)
89         {
90             ll r=a[i]%m;
91             if(r!=r1)
92             {
93                 if(r2==-1)
94                 {
95                     r2=r;
96                 }
97                 else if(r!=r2)
98                 {
99                     ok=false;
100                     break;
101                 }
102             }
103         }
104         if(ok)resp++;
105     }
106     cout<<2<<" "<<resp<<'\n';
107 }

```

Code E - Wise Partitioning Method

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  const int MAXN=200010;
5  vector<int>Segtree[4*MAXN];
6  pair<int,int>a[MAXN];
7  int pos_ordenada[MAXN];
8  int n;
9  void build(int pos, int l, int r)
10 {
11     if(l==r)
12     {
13         Segtree[pos].push_back(a[l].second);
14         return;
15     }
16     int m=(l+r)/2;
17     build(2*pos,l,m);
18     build(2*pos+1,m+1,r);
19     merge(Segtree[2*pos].begin(),Segtree[2*pos].end(),
20           Segtree[2*pos+1].begin(),Segtree[2*pos+1].end(),
21           back_inserter(Segtree[pos]));

```

```

22 }
23 // How many original indices in [ql, qr] are <= x?
24 int query(int pos,int l,int r,int ql,int qr,int x){
25     if(ql>r||qr<l)return 0;
26     if(ql<=l&&r<=qr){
27         return upper_bound(Segtree[pos].begin(),Segtree[pos].end(),x)-Segtree[pos].begin();
28     }
29     int mid=l+(r-1)/2;
30     return query(2*pos,l,mid,ql,qr,x)+
31         query(2*pos+1,mid+1,r,ql,qr,x);
32 }
33 ll solve(int L,int R,int flips)
34 {
35     if(L>=R)return 0;
36     int sz=R-L+1;
37     ll cmps=sz;
38     int mid_pos=(sz-1)/2;
39
40     // If flips is even, the order is normal; otherwise, it is reversed.
41     int k=(flips%2==0)?(mid_pos+1):(sz-mid_pos);
42
43     // Binary-search the pivot's original index
44     int low=0,high=n-1,best_orig=-1;
45     while(low<=high){
46         int mid_idx=low+(high-low)/2;
47         if(query(1,0,n-1,L,R,mid_idx)>=k)
48         {
49             best_orig=mid_idx;
50             high=mid_idx-1;
51         }
52         else
53         {
54             low=mid_idx+1;
55         }
56     }
57     int p_rank=pos_ordenada[best_orig];
58     return cmps+solve(L,p_rank-1,flips)+solve(p_rank+1,R,flips+1);
59 }
60 int main(){
61     ios_base::sync_with_stdio(false);
62     cin.tie(NULL);
63     cin>>n;
64     for(int i=0;i<n;i++){
65         cin>>a[i].first;
66         a[i].second=i;
67     }
68
69     sort(a,a+n);
70

```

```

71     for(int i=0;i<n;i++){
72         pos_ordenada[a[i].second]=i;
73     }
74     build(1,0,n-1);
75     cout<<solve(0,n-1,0)<<'\n';
76     return 0;
77 }

```

Code F - Chinese Innovation

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  #define int long long
5  #define pii pair<int, int>
6  const int INF = 1e18+10;
7
8  int dijkstra(int n, int k, vector<vector<pii>> &grafo){
9      vector<int> dist(n+k+10, INF);
10     set<pii> s;
11     dist[1] = 0;
12     for(int i=1; i<=n+k; ++i) s.insert({dist[i], i});
13     while(s.size()){
14         int cur = s.begin()->second;
15         s.erase(s.begin());
16         for(auto [viz, c]:grafo[cur]){
17             if(dist[cur]+c<dist[viz]){
18                 s.erase({dist[viz], viz});
19                 dist[viz] = dist[cur] + c;
20                 s.insert({dist[viz], viz});
21             }
22         }
23     }
24     return dist[n];
25 }
26
27 int32_t main(){
28     ios_base::sync_with_stdio(false); cin.tie(NULL);
29
30     int n, m, k; cin >> n >> m >> k;
31     vector<vector<pii>> grafo(n+k+10);
32     for(int i=0; i<m; ++i){
33         int u, v, c; cin >> u >> v >> c;
34         grafo[u].push_back({v, c});
35         grafo[v].push_back({u, c});
36     }
37
38     for(int v=1; v<=n; ++v){

```

```

39         int t; cin >> t;
40         for(int i=1; i<=t; ++i){
41             int u, c; cin >> u >> c;
42             grafo[v].push_back({n+u, c});
43             grafo[n+u].push_back({v, 0});
44         }
45     }
46
47     cout << dijkstra(n, k, grafo) << "\n";
48 }

```

Code G - Incompatible Pairs

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  #define int long long
5
6  int32_t main(){
7      ios_base::sync_with_stdio(false); cin.tie(NULL);
8
9      int n; cin >> n;
10     string s; cin >> s;
11
12     vector<int> fim(n+10), pref(n+10);
13     int cur = 0, l = 0, tot = 0;
14     for(int i=0; i<n; ++i){
15         if(s[i]=='(') ++cur;
16         if(s[i]==')') --cur, ++tot;
17         if(cur==0){
18             pref[i] = tot;
19             for(int j=i; j>=l; --j){
20                 fim[j] = i;
21             }
22             l = i+1;
23         }
24     }
25
26     int resp = 0;
27     for(int i=0; i<n; ++i){
28         if(s[i]=='(') resp += tot - pref[fim[i]];
29     }
30     cout << resp << "\n";
31 }

```

Code H - The Wisdom of Master Wei

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  int32_t main(){
5      ios_base::sync_with_stdio(false); cin.tie(NULL);
6      int w, h; cin >> w >> h;
7      cout << 2*h-w << "\n";
8  }
```

Code I - Bath

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  typedef double ld;
5  const ld DINF = 1e18;
6  const ld pi = acos(-1.0);
7  const ld eps = 1e-9;
8  const int MAXN = 510;
9
10 #define sq(x) ((x)*(x))
11
12 bool eq(ld a, ld b) {
13     return abs(a - b) <= eps;
14 }
15
16 struct pt { // point
17     ld x, y;
18     pt(ld x_ = 0, ld y_ = 0) : x(x_), y(y_) {}
19     bool operator < (const pt p) const {
20         if (!eq(x, p.x)) return x < p.x;
21         if (!eq(y, p.y)) return y < p.y;
22         return 0;
23     }
24     bool operator == (const pt p) const {
25         return eq(x, p.x) and eq(y, p.y);
26     }
27     pt operator + (const pt p) const { return pt(x+p.x, y+p.y); }
28     pt operator - (const pt p) const { return pt(x-p.x, y-p.y); }
29     pt operator * (const ld c) const { return pt(x*c, y*c); }
30     pt operator / (const ld c) const { return pt(x/c, y/c); }
31     ld operator * (const pt p) const { return x*p.x + y*p.y; }
32     ld operator ^ (const pt p) const { return x*p.y - y*p.x; }
33     friend istream& operator >> (istream& in, pt& p) {
34         return in >> p.x >> p.y;
35     }
36 }
```

```

35     }
36 };
37
38 ld dist(pt p, pt q) { // distance
39     return hypot(p.y - q.y, p.x - q.x);
40 }
41
42 struct line { // line
43     pt p, q;
44     line() {}
45     line(pt p_, pt q_) : p(p_), q(q_) {}
46     friend istream& operator >> (istream& in, line& r) {
47         return in >> r.p >> r.q;
48     }
49 };
50
51 ld sarea(pt p, pt q, pt r) { // signed area
52     return ((q-p)^(r-q))/2;
53 }
54
55 bool ccw(pt p, pt q, pt r) { // whether p, q, r are counterclockwise
56     return sarea(p, q, r) > eps;
57 }
58
59 bool isinseg(pt p, line r) { // whether p belongs to segment r
60     pt a = r.p - p, b = r.q - p;
61     return eq((a ^ b), 0) and (a * b) < eps;
62 }
63
64 bool interseg(line r, line s) { // whether segments r and s intersect
65     if (isinseg(r.p, s) or isinseg(r.q, s)
66         or isinseg(s.p, r) or isinseg(s.q, r)) return 1;
67
68     return ccw(r.p, r.q, s.p) != ccw(r.p, r.q, s.q) and
69           ccw(s.p, s.q, r.p) != ccw(s.p, s.q, r.q);
70 }
71
72 // Point-in-polygon test: returns 0 outside,
73 // 1 in the interior, and 2 on the boundary
74 int inpol(vector<pt>& v, pt p) { // O(n)
75     int qt = 0;
76     for (int i = 0; i < v.size(); i++) {
77         if (p == v[i]) return 2;
78         int j = (i+1)%v.size();
79         if (eq(p.y, v[i].y) and eq(p.y, v[j].y)) {
80             if ((v[i]-p)*(v[j]-p) < eps) return 2;
81             continue;
82         }
83         bool baixo = v[i].y+eps < p.y;

```

```

84         if (baixo == (v[j].y+eps < p.y)) continue;
85         auto t = (p-v[i])^(v[j]-v[i]);
86         if (eq(t, 0)) return 2;
87         if (baixo == (t > eps)) qt += baixo ? 1 : -1;
88     }
89     return qt != 0;
90 }
91
92 ld dp[MAXN][MAXN];
93
94 void fw(int n){
95     for(int k=0; k<=n; ++k){
96         for(int i=0; i<=n; ++i){
97             for(int j=0; j<=n; ++j){
98                 dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j]);
99             }
100         }
101     }
102 }
103
104 int32_t main(){
105     ios_base::sync_with_stdio(false); cin.tie(NULL);
106
107     int n; cin >> n;
108     vector<pt> pol(n+1), soh_pol(n);
109     for(int i=0; i<=n; ++i){
110         cin >> pol[i].x >> pol[i].y;
111         if(i<n) soh_pol[i] = pol[i];
112     }
113
114     for(int i=0; i<=n; ++i) for(int j=0; j<=n; ++j) dp[i][j] = DINF;
115     for(int i=0; i<=n; ++i) dp[i][i] = 0;
116
117     for(int i=0; i<=n; ++i){
118         for(int j=i+1; j<=n; ++j){
119             line cur(pol[i], pol[j]);
120             bool dah = true;
121             for(int k=0; k<n-1; ++k){
122                 if(k==i || k==j || k+1==i || k+1==j) continue;
123                 line outro(pol[k], pol[k+1]);
124                 if(interseg(cur, outro)) dah = false;
125             }
126             int k = n-1;
127             if(!(k==i || k==j || 0==i || 0==j)){
128                 line outro(pol[k], pol[0]);
129                 if(interseg(cur, outro)) dah = false;
130             }
131
132             pt mid = (pol[i] + pol[j]) / 2;

```

```

133         if(!inpol(soh_pol, mid)) dah = false;
134
135         if(dah) dp[i][j] = dp[j][i] = dist(pol[i], pol[j]);
136     }
137 }
138
139 fw(n);
140
141 cout << setprecision(10) << fixed;
142 cout << 2*dp[0][n] << "\n";
143 }

```

Code J - The Messenger's Disguise

```

1  #include<bits/stdc++.h>
2  #define int long long
3  using namespace std;
4
5  const int MOD = 1e9 + 7;
6  const int MAXN = 1e6 + 10;
7  const int INF = 1e9;
8
9  int dist[MAXN], marc[MAXN];
10 vector<int> grafo[MAXN];
11 int dp[MAXN][2];
12 int N,M,S,T;
13
14 void bfs()
15 {
16     for(int i = 1; i <= N; i++) dist[i] = INF;
17
18     dist[S] = 0;
19     dp[S][0] = 1;
20     queue<int> q; q.push(S);
21
22     while(!q.empty())
23     {
24         int cur = q.front(); q.pop();
25
26         for(int i = 0; i < grafo[cur].size(); i++)
27         {
28             int viz = grafo[cur][i];
29             int mult = 1;
30
31             // Process every repeated edge at once.
32             while(i < grafo[cur].size() - 1 and grafo[cur][i + 1] == viz)
33             {
34                 i++;

```

```

35         mult++;
36     }
37
38     if(dist[viz] == INF)
39     {
40         dist[viz] = dist[cur] + 1;
41         q.push(viz);
42     }
43
44     if(dist[cur] == dist[viz] + 1)
45     {
46         dp[cur][0] += (dp[viz][0]*mult)%MOD;
47         dp[cur][0] %= MOD;
48     }
49 }
50 }
51 }
52
53 void calc()
54 {
55     queue<int> q; q.push(S);
56     marc[S] = 1;
57     while(!q.empty())
58     {
59         int cur = q.front(); q.pop();
60         for(int i = 0; i < grafo[cur].size(); i++)
61         {
62             int viz = grafo[cur][i];
63             int mult = 1;
64
65             // Process every repeated edge at once.
66             while(i < grafo[cur].size() - 1 and grafo[cur][i + 1] == viz)
67             {
68                 i++;
69                 mult++;
70             }
71
72             if(dist[viz] == dist[cur])
73             {
74                 dp[cur][1] += (dp[viz][0]*mult)%MOD;
75                 dp[cur][1] %= MOD;
76             }
77
78             if(dist[cur] == dist[viz] + 1)
79             {
80                 dp[cur][1] += (dp[viz][1]*mult)%MOD;
81                 dp[cur][1] %= MOD;
82             }
83

```

```

84         if(dist[viz] == dist[cur] + 1 and !marc[viz])
85         {
86             marc[viz] = 1;
87             q.push(viz);
88         }
89     }
90 }
91 }
92
93 int32_t main()
94 {
95     cin.tie(0)->sync_with_stdio(0);
96
97     cin >> N >> M >> S >> T;
98
99     for(int i = 1; i <= M; i++)
100     {
101         int a,b; cin >> a >> b;
102         grafo[a].push_back(b);
103         grafo[b].push_back(a);
104     }
105
106     // Sort the adjacency lists to handle multiple edges.
107     for(int i = 1; i <= N; i++)
108         sort(grafo[i].begin(),grafo[i].end());
109
110     bfs();
111     calc();
112
113     cout << dp[T][1] << "\n";
114 }

```

Code K - Cake Hater

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  int main()
5  {
6      cin.tie(0)->sync_with_stdio(0);
7
8      int N; cin >> N;
9
10     int A,B; cin >> A >> B;
11
12     vector<int> ingredients(A + 1);
13     for(int i = 1; i <= B; i++)
14     {

```

```

15         int x; cin >> x;
16         ingredients[x] = 1;
17     }
18
19     for(int i = 1; i <= N; i++)
20     {
21         int M; cin >> M;
22
23         int missing = 0;
24         for(int j = 1; j <= M; j++)
25         {
26             int ing; cin >> ing;
27
28             if(ingredients[ing]) missing++;
29         }
30
31         if(missing > M/3)
32         {
33             cout << i << "\n";
34             return 0;
35         }
36     }
37
38     cout << "-1\n";
39 }

```

Code L - Game of Life

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4
5  template<typename T> struct matrix : vector<vector<T>> {
6      int n,m;
7
8      matrix(int n_, int m_, bool ident = false)
9          : vector<vector<T>>(n_,vector<T>(m_,0)),n(n_),m(m_)
10     {
11         if(ident)
12         {
13             for(int i = 0; i < n; i++) (*this)[i][i] = 1;
14         }
15     }
16
17     matrix(const vector<vector<T>>& c)
18         : vector<vector<T>>(c), n(c.size()),m(c[0].size()) {}
19
20     matrix<T> operator*(matrix<T>& r)

```

```

21     {
22         matrix<T> M(n,r.m);
23
24         for(int i = 0; i < n; i++) for(int k = 0; k < m; k++)
25         {
26             for(int j = 0; j < r.m; j++)
27             {
28                 T add = (*this)[i][k]*r[k][j];
29                 M[i][j] ^= add;
30             }
31         }
32         return M;
33     }
34
35     matrix<T> operator^(ll e)
36     {
37         matrix<T> M(n,n,true), at = *this;
38
39         while(e) {
40             if(e&1) M = M*at;
41             e >>= 1;
42             at = at*at;
43         }
44         return M;
45     }
46 };
47
48 int main()
49 {
50     cin.tie(0)->sync_with_stdio(0);
51
52     int N,K; cin >> N >> K;
53
54     string grid[N];
55     for(int i = 0; i < N; i++)
56     {
57         cin >> grid[i];
58     }
59
60     vector<vector<int>> v(1);
61
62     for(int i = 0; i < N; i++)
63     {
64         for(int j = 0; j < N; j++)
65         {
66             if(grid[i][j] == '1') v[0].push_back(1);
67             else v[0].push_back(0);
68         }
69     }

```

```

70
71     matrix<int> estado(v);
72     vector<vector<int>> trans(N*N,vector<int>(N*N));
73
74     int dx[] = {1,1,1,-1,-1,-1,0,0,0};
75     int dy[] = {1,0,-1,1,0,-1,1,0,-1};
76     for(int i = 0; i < N; i++)
77     {
78         for(int j = 0; j < N; j++)
79         {
80             if(grid[i][j] == '#') continue;
81             for(int k = 0; k < 9; k++)
82             {
83                 int nx = i + dx[k];
84                 int ny = j + dy[k];
85
86                 if(nx < 0 or nx >= N or ny < 0 or ny >= N or
87                    grid[nx][ny] == '#') continue;
88
89                 trans[N*nx + ny][N*i + j] = 1;
90             }
91         }
92     }
93
94     matrix<int> mtrans(trans);
95
96     mtrans = mtrans^K;
97
98     estado = estado*mtrans;
99
100    for(int i = 0; i < N; i++)
101    {
102        for(int j = 0; j < N; j++)
103        {
104            if(grid[i][j] == '#') cout << grid[i][j];
105            else cout << estado[0][N*i + j];
106        }
107        cout << "\n";
108    }
109 }

```

Code M - Nomad

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  int main()
5  {

```

```

6      cin.tie(0)->sync_with_stdio(0);
7
8      int N,D; cin >> N >> D;
9
10     vector<pair<int,int>> non_zero;
11     for(int i = 1; i <= N; i++)
12     {
13         int x; cin >> x;
14         if(x != 0) non_zero.push_back({x,i});
15     }
16
17     if(non_zero.size() == 0)
18     {
19         cout << "-1\n";
20     }
21     else if(non_zero.size() == 1)
22     {
23         if(non_zero[0].first < D)
24             cout << "-1\n";
25         else
26         {
27             for(int i = 1; i <= D; i++)
28                 cout << non_zero[0].second << " ";
29             cout << "\n";
30         }
31     }
32     else
33     {
34         int cur = 0;
35         for(int i = 1; i <= D; i++)
36         {
37             for(int j = 1; j <= non_zero[cur].first and i <= D; j++, i++)
38             {
39                 cout << non_zero[cur].second << " ";
40             }
41             i--;
42             if(cur == 0) cur = 1;
43             else cur = 0;
44         }
45         cout << "\n";
46     }
47 }

```
