

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

Editorial

Autores: Pedro Tsuchie, Guilherme Kavakami e Murilo Kataoka

Explicações dos Problemas

A. Yuyuan Market

Tags: Programação Dinâmica

Antes de partirmos para a DP, vale termos a seguinte observação: quando escolhemos pegar um elemento cuja segunda posição (mais à direita) é R e a primeira é L , somos obrigados a não pegar ninguém entre L e R . E mais do que isso, as escolhas que fizemos antes de L não interferem no meio do caminho, logo, queremos sempre o melhor estado obtível até a posição $L - 1$.

Assim, chegamos na seguinte transição. Se estivermos em uma posição i que é a segunda ocorrência de um símbolo (ou seja, $i = R$ para algum par com início em L), a nossa escolha será o máximo entre ignorar o elemento atual ou pegá-lo (somando seu custo e quantidade de pares ao melhor resultado que tínhamos antes de L):

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

Como queremos maximizar a quantidade de pares e, em caso de empate, minimizar o custo da distância, podemos tratar o valor da DP como um par (quantidade, custo). A transição final fica:

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

Essa abordagem nos garante que cada posição do vetor seja avaliada em $\mathcal{O}(1)$, resultando em uma complexidade final de tempo $\mathcal{O}(N)$, o que é perfeitamente suficiente para passar no tempo limite de 1 segundo para $N \leq 10^5$.

B. The Search for Balance

Tags: Meet in the Middle, Line Sweep

Antes de irmos para a solução em si, convém falarmos do número 35. Ele é interessante pois não é tão pequeno a ponto de permitir um *bruteforce* completo (já que 2^{35} são mais de 10^{10} operações), mas também é pequeno o suficiente para descartarmos soluções polinomiais comuns. Quando nos deparamos com esse tipo de situação, a técnica de **Meet in the Middle** se torna uma forte candidata. Afinal, se o limite fosse um pouco menor

e pudéssemos testar todas as combinações de uma vez, o problema seria trivial: apenas guardaríamos todos os estados gerados e verificaríamos quais batem com a resposta. Se você nunca ouviu falar nessa técnica, este clássico do CSES é um ótimo problema para começar!

Ao aplicar o *Meet in the Middle*, dividimos os power-ups em duas metades (tamanhos 17 e 18) e geramos duas listas com todas as possibilidades de pares (Ataque, Defesa) obtidas em cada metade. O problema então se transforma: para cada par gerado na primeira metade, precisamos descobrir quantos pares da segunda metade possuem os atributos necessários para que a soma caia exatamente nos intervalos desejados de balanceamento ($[A_1, A_2]$ e $[D_1, D_2]$).

Na prática, isso se resume a um problema clássico de **contar pontos dentro de um retângulo 2D**. Para contar isso rapidamente sem precisar iterar pela segunda lista inteira (o que daria *Time Limit Exceeded*), utilizamos estruturas de dados eficientes. Você pode resolver isso aplicando a técnica de **Line Sweep (Linha de Varredura)** no eixo do ataque junto com uma **Fenwick Tree (BIT)** para as defesas, ou, utilizar uma **Merge Sort Tree**. Ambas as técnicas contam os pontos no retângulo em tempo logarítmico, resolvendo o problema!

C. Echoes of the Jade Library

Tags: Eertree, Segment Tree Persistente

Note que esse problema é similar ao Distinct Values Queries do CSES: queremos responder Q queries de distintos em intervalos, mas ao invés de números estamos interessados em palíndromos. Além disso, o problema deve ser resolvido de forma online, então estamos interessados na solução online do problema do CSES, que usa Segment Tree Persistente.

Para transformar os palíndromos em números, podemos manter uma Eertree (Palindromic Tree) global que usa separadores entre as strings da entrada. Sempre que um caractere de uma string é inserido, a Eertree tem acesso ao estado do maior palíndromo que ele forma, e o número que representa esse estado será justamente o número associado a esse palíndromo. Dessa forma, cada string da entrada terá uma lista de números associados a ela.

Agora, basta usar a ideia do Distinct Values Queries para resolver o problema: considere inicialmente as queries com $L = 1$. Podemos montar um vetor v de modo que v_i representa o número de novos palíndromos que aparecem na posição i . Cada palíndromo só é contado na primeira vez que ele aparece. Com isso, podemos usar uma Segment Tree de soma para calcular a resposta para todas as queries que possuem $L = 1$ em tempo $O(\log N)$.

Como calcular a resposta para " L "s maiores? Quando incrementamos L em 1, todos os palíndromos que apareciam na posição antiga de L agora precisam ser contados na próxima posição i que eles aparecem. Isso significa que podemos criar uma nova raiz na Segment Tree, que representa a posição $L = 2$, e atualizar apenas os nodes que se alteram. Assim, no total faremos $O(M \log N)$ updates, onde M é o número total de caracteres.

Fazendo isso para todo $L > 1$, teremos uma Segment Tree Persistente que é capaz de responder as queries de soma sobre os intervalos que queremos. Assim, para responder a uma query $[L, R]$, perguntamos a soma $[L, R]$ da raiz L , e o resultado é o número de palíndromos distintos nesse intervalo, exatamente o que queremos.

Construir a Segment Tree persistente tem custo $O(M \log N)$, e responder as queries tem custo $O(Q \log N)$. Portanto, a complexidade final é $O((M + Q) \log N)$.

D. The Seals of Shanghai

Tags: Teoria dos números, ad-hoc

Antes de irmos para a simulação em si, convém pensarmos sobre o limite superior da nossa resposta. Qual é o máximo de rodadas que Lian precisaria no pior cenário possível? Se escolhermos $M = 2$, os únicos restos possíveis para qualquer número são 0 ou 1. Ou seja, na pior das hipóteses, todos os selos serão divididos em exatamente dois grupos. Isso nos dá uma garantia matemática: **a resposta para o número mínimo de rodadas será sempre 1 ou 2!**

Quando a resposta é 1? Para que todos os selos sejam abertos em uma única rodada, todos os elementos do vetor devem deixar o mesmo resto quando divididos por M . Matematicamente, isso significa que $A_i \equiv A_1 \pmod{M}$, o que implica que M deve ser um divisor da diferença $|A_i - A_1|$ para todo i . Portanto, basta calcularmos o Máximo Divisor Comum (GCD) de todas as diferenças $|A_i - A_1|$. Se esse $GCD > 1$, a resposta é 1 rodada, e a quantidade de valores de M que possibilitam isso é simplesmente o número de divisores desse GCD (excluindo o número 1). Vale lembrar do *corner case*: se todos os números do vetor forem iguais, $GCD = 0$, e qualquer M é válido (totalizando $10^9 - 1$ opções).

Quando a resposta é 2: Se o $GCD = 1$, é impossível resolver em 1 rodada. Sabemos que a resposta é 2, mas como testar os valores de M até 10^9 sem tomar *Time Limit Exceeded*? É aqui que entra o **Princípio da Casa dos Pombos**.

Se a resposta ótima é 2 rodadas, significa que nosso M ideal dividirá a array em exatamente 2 grupos de restos. Vamos olhar apenas para os **3 primeiros números distintos** do nosso vetor (v_1, v_2 e v_3). Se temos 3 elementos (pombos) e apenas 2 grupos de restos possíveis (casas), **obrigatoriamente pelo menos dois desses elementos terão o mesmo resto!**

Isso significa que o nosso M tem que deixar o mesmo resto para v_1 e v_2 , **OU** para v_1 e v_3 , **OU** para v_2 e v_3 . Logo, M é obrigatoriamente um divisor de $|v_1 - v_2|$, $|v_1 - v_3|$ ou $|v_2 - v_3|$.

Pronto! Reduzimos infinitas possibilidades para uma lista minúscula de divisores. Com esses candidatos em mãos, basta simular.

E. Wise Partitioning Method

Tags: Divide and Conquer, Merge Sort Tree, Busca Binária

A primeira ideia para resolver este problema seria simular o algoritmo passo a passo, mas isso é N^2 no pior caso.

Passo 1: Entendendo a ordem relativa

O grande segredo está na forma como o Mestre Wei escreveu a função *pivot*. Observe a separação dos elementos em relação ao pivô (x):

- Se o elemento for menor ($y < x$), ele é colocado no início e o ponteiro da esquerda avança. Isso significa que os elementos menores **mantêm exatamente a sua ordem relativa original**.

- Se o elemento for maior ($y > x$), ele é colocado no final e o ponteiro da direita recua. Ou seja, o primeiro cara maior vai para o final, o segundo vai para a penúltima posição, e assim por diante. Os elementos maiores **invertem completamente a sua ordem relativa!**

Passo 2: Recursão

Como o Quicksort sempre separa perfeitamente os valores menores dos maiores, qualquer chamada recursiva atua sobre um intervalo de valores estritamente contíguo, que podemos representar como $[L, R]$ se imaginarmos o vetor ordenado. Mas em qual ordem física eles estão sentados nessa chamada? Depende apenas de quantas vezes fomos para a chamada da direita na árvore de recursão (*flips*):

- Se *flips* for **par**, as inversões se anulam. Os elementos estão dispostos na sua **ordem original** (baseada no índice em que começaram).
- Se *flips* for **ímpar**, os elementos estão dispostos exatamente **de trás para frente** em relação à sua ordem original.

Passo 3: Adivinhando o Pivô

O algoritmo escolhe fisicamente o cara da cadeira do meio: $m = (l + r) // 2$. Como sabemos a quantidade de elementos na sala (sz) e se a fila está normal ou invertida, sabemos exatamente qual posição da ordem original essa cadeira do meio representa. O pivô será o elemento que possui o **K -ésimo menor índice original** dentre os elementos do intervalo $[L, R]$.

Passo 4: MergeSortTree

Para achar rápido quem é esse cara, construímos uma **Merge Sort Tree** sobre os valores ordenados. Cada nó da árvore cobre um intervalo de valores, mas guarda internamente um vetor ordenado apenas com os **índices originais** daqueles elementos. Com isso, podemos fazer uma **Busca Binária** chutando um índice original X e perguntando à árvore: “No intervalo de valores $[L, R]$, quantos elementos vieram de um índice original $\leq X$?”. A árvore responde em $\mathcal{O}(\log^2 N)$ usando **upper_bound**.

Ao encontrar o índice original correto do pivô, usamos um vetor auxiliar (**pos_ordenada**) para descobrir qual é a posição por valor desse elemento. Esse Rank é exatamente o que nos permite quebrar o intervalo atual $[L, R]$ em $[L, Rank - 1]$ (menores) e $[Rank + 1, R]$ (maiores) para as próximas chamadas recursivas, em tempo total $\mathcal{O}(N \log^3 N)$.

F. Chinese Innovation

Tags: Dijkstra

Primeiro, note que o problema sem os teleportes é o problema clássico de menor caminho do vértice 1 até o vértice N , que pode ser resolvido com o algoritmo de Dijkstra.

Para lidar com os teleportes, perceba que não é possível criar uma aresta entre todos os vértices que compartilham um teleporte de tipo u_i , pois no pior caso podem existir $\mathcal{O}(N^2)$ arestas. Assim, podemos criar vértices auxiliares para representar o acesso a um teleporte de tipo u_i : se um vértice v possui acesso a um teleporte u_i com custo c_i , então existe uma aresta direcionada de v para u_i com custo c_i , e uma aresta direcionada de u_i para v com custo 0.

Dessa forma, o número de vértices do novo grafo é $\mathcal{O}(N + K)$ e o número de arestas é $\mathcal{O}(M + S)$, onde $S \leq \min(NK, 2 \cdot 10^5)$ é o número total de teleportes entre todas as

idades. Portanto, basta aplicar o algoritmo de Dijkstra nesse novo grafo. Para lidar com os novos vértices auxiliares, podemos usar os índices $u_i + N$.

G. Incompatible Pairs

Tags: Ad-hoc

Vamos dividir a string em seus blocos balanceados mínimos: sempre que o balanço volta para zero, termina um bloco. Por exemplo, " $((()))()$ " é dividido em " $((()))$ " e " $()$ ".

Quando um par é incompatível? Considere um '(' na posição i . Se o emparelharmos com um ')' que está no mesmo bloco, os demais parênteses desse bloco continuam podendo ser emparelhados normalmente. Portanto, esse par é compatível. Por outro lado, se escolhermos um ')' que está em um bloco posterior, o bloco do '(' ficará com um fechamento sobrando, e não será mais possível emparelhar todos os parênteses restantes.

Logo, para cada posição i tal que $s_i = '('$, basta contar quantos j existem tal que $s_j = ')'$ e j está em um bloco posterior ao bloco de i . Podemos usar soma de prefixo para calcular esse valor. Complexidade: $O(N)$.

H. The Wisdom of Master Wei

Tags: Matemática

Dados W e K , o problema pede um inteiro X tal que $X - W$ é o dobro de $X - K$, ou seja,

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

Portanto, basta imprimir $2K - W$.

I. Bath

Tags: Geometria, Floyd-Warshall

O problema pede que encontremos o menor caminho entre o ponto inicial e um ponto final no interior do polígono, passando apenas pelo interior do polígono. Observe que um caminho mínimo entre o ponto inicial e o ponto final é formado por um conjunto de segmentos de reta, onde as extremidades dos segmentos de reta são vértices do polígono ou o ponto final.

Assim, suponha que para todo par de pontos (u, v) sabemos se existe um segmento de reta que conecta u e v passando apenas pelo interior do polígono, assim como seu comprimento. Nesse caso, basta usar o algoritmo de Floyd-Warshall para calcular a distância entre o ponto inicial e o ponto final.

Vamos verificar para todo par de pontos (u, v) se tal segmento de reta existe, e se existir, calcular seu comprimento. Seja cur o segmento de reta que liga u até v , cur pode ser descartado se existir um lado do polígono $outro$ tal que u e v não são seus vértices e cur e $outro$ se intersectam. Caso tenha intersecção, não existe tal segmento entre u e v . Caso contrário, cur está inteiramente dentro ou inteiramente fora do polígono, então basta escolher um ponto qualquer de cur e verificar se ele está dentro ou fora do polígono. Caso esteja fora, não existe tal segmento entre u e v . Caso contrário, esse segmento existe e seu comprimento pode ser facilmente calculado. Esse processo tem complexidade $O(N^3)$.

Agora, basta usar o algoritmo de Floyd-Warshall para calcular a distância e multiplicar o resultado por 2.

J. The Messenger's Disguise

Tags: Programação Dinâmica

Nesse problema vamos utilizar que $dist[i]$ = distância mínima de S até i .

Vamos começar já definindo a DP que iremos utilizar para resolver esse problema:

$$\begin{cases} DP[i][0] = \text{maneiras de chegar em } i \text{ com } dist[i] \text{ arestas} \\ DP[i][1] = \text{maneiras de chegar em } i \text{ utilizando } dist[i] + 1 \text{ arestas} \end{cases}$$

Antes de analisar as transições da DP, considere os seguintes tipos de vizinhos de um certo vértice i :

- **Tipo1:** Vizinhos tais que $dist[i] = dist[Tipo1]$.
- **Tipo2:** Vizinhos tais que $dist[i] = dist[Tipo2] + 1$.

Agora, vamos analisar as transições da DP:

- $DP[i][0]$: Chegar em i com a menor distância possível só acontece quando estamos vindo de um vizinho **Tipo2**, pois chegamos em **Tipo2** utilizando $dist[Tipo2]$ arestas, utilizamos a aresta de **Tipo2** para i e chegamos em i utilizando $dist[Tipo2] + 1 = dist[i]$ arestas. Então:

$$DP[i][0] = \sum DP[Tipo2][0]$$

- $DP[i][1]$: Para chegar em i utilizando $dist[i] + 1$ arestas, podemos ter vindo de dois tipos de vértice:
 - **Tipo1:** Nesse caso, basta utilizarmos um caminho mínimo para chegar até **Tipo1** e depois utilizar a aresta entre **Tipo1** e i para chegar em i utilizando $dist[i] + 1$ arestas.
 - **Tipo2:** Nesse caso, basta utilizarmos um caminho que chegue no vértice do **Tipo2** usando $dist[Tipo2] + 1$ arestas e depois utilizar a aresta entre **Tipo2** e i . Assim, chegamos em i utilizando $dist[Tipo2] + 1 + 1 = dist[i] + 1$ arestas.

Então, podemos escrever a transição da DP como:

$$DP[i][1] = \sum DP[Tipo1][0] + \sum DP[Tipo2][1]$$

Assim, a resposta é simplesmente $DP[T][1]$.

Detalhes de Implementação

Temos que tomar um certo cuidado com a ordem em que calculamos a nossa DP. Perceba que a $DP[i]$ depende tanto de vértices com distância menor que i quanto de vértices com distância igual a i . Por isso, é justo calcularmos a nossa DP ao longo de uma BFS.

Como precisamos de $DP[Tipo1][0]$ para calcular a $DP[i][1]$ e a ordem da BFS não garante que já teremos processados todos os vértices do *Tipo1* antes de i , eu optei por calcular a $DP[i][1]$ em uma segunda BFS, feita depois da primeira que calcula tanto as distâncias quanto $DP[i][0]$.

Outra questão é sobre como lidar com arestas múltiplas. Como nesse problema o tempo limite é apertado, eu optei por fazer isso ordenando os vértices na lista de adjacência e lidando com todos os repetidos de uma só vez. A complexidade final fica $O(2N + N \times \log N)$.

K. Cake Hater

Tags: Vetor de Marcação

Para esse problema, basta mantermos, com um vetor de marcação, quais ingredientes Antônio não gosta. Depois, para cada receita, basta checar para cada ingrediente se Antônio gosta ou não dele e parar quando essa contagem for maior que um terço dos ingredientes na receita.

L. Game of Life

Tags: Operações Binárias, Exponenciação de Matriz

A primeira grande observação desse problema é que podemos simplificar muito as condições de como uma célula morre ou fica viva. Podemos substituir as condições do enunciado pela seguinte condição:

$$S_{i,j} = S_{i,j} \oplus S_{i-1,j-1} \oplus S_{i-1,j} \oplus S_{i-1,j+1} \oplus S_{i,j-1} \oplus S_{i,j+1} \oplus S_{i+1,j-1} \oplus S_{i+1,j} \oplus S_{i+1,j+1}$$

Ou seja, o estado de $S_{i,j}$ na próxima iteração é o **XOR** dele com todos os vizinhos.

Então, para conseguirmos realizar todas as iterações dentro do tempo limite, vamos transformar a transição numa multiplicação de matriz (mas ao invés de somarmos os produtos das casas, vamos calcular o **XOR** desses produtos) e depois utilizar exponenciação rápida de matriz para calcularmos rapidamente o estado final do tabuleiro. Ou seja, para resolver o problema basta fazermos:

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

Onde S é o tabuleiro vetorizado, T é a matriz de transição, K é a quantidade de iterações e R é a matriz final.

Vamos entender como obter todas essas partes da resposta. Para vetorizar o tabuleiro, basta colocar o valor da casa na linha i e coluna j na posição $i \times N + j$ do vetor S . Assim, o nosso tabuleiro inicial vira uma matriz $1 \times N^2$, assim como a matriz resposta R .

Agora, vamos entender como funciona a matriz de transição T , que é uma matriz $N^2 \times N^2$. Para descobrir o próximo estado da j -ésima casa de S estaremos multiplicando S pela j -ésima coluna de T . Como vimos, para calcular o próximo estado de uma única casa, queremos calcular o XOR dela com as casas vizinhas. Para considerarmos só as casas vizinhas, basta colocarmos, na linha j de T , 1 nas casas que são ou vizinhas ou ela mesma e 0 nas casas que não são vizinhas. A coluna 0 de T , para $N = 3$, por exemplo, é

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

Se considerarmos S como $[0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0]$. O próximo estado da casa 0 será:

$$(S_0 \times T_{(0,0)}) \oplus (S_1 \times T_{(1,0)}) \oplus (S_2 \times T_{(2,0)}) \oplus (S_3 \times T_{(3,0)}) \oplus (S_4 \times T_{(4,0)}) \oplus \dots \oplus (S_8 \times T_{(8,0)}) = (0 \times 1) \oplus (1 \times 1) \oplus (1 \times 0) \oplus (0 \times 1) \oplus (1 \times 1) \oplus \dots \oplus (0 \times 0) = 0$$

Ainda temos que lidar com as casas do tipo #, mas isso é fácil, basta, para toda casa desse tipo deixa-lá como 0 na versão vetorizada e também não colocar nada na sua respectiva coluna de T .

M. Nomad

Tags:Ad-Hoc

Vamos dividir esse problema em 3 casos:

- **Todo** $d_i = 0$: Nesse caso, ele não consegue ficar em lugar nenhum.
- **Só um** $d_i \neq 0$: Nesse caso, a única coisa que ele pode fazer é ficar nesse lugar que $d_i \neq 0$.
- **Mais de um** $d_i \neq 0$: Nesse caso ele consegue ficar uma quantidade ilimitada de dias, basta ficar alternando entre dois dos lugares tais que $d_i \neq 0$.

Implementações

Código 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 }

```

Código 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 =Ponto da Metade 2,1=Borda Direita,-1 =Borda Esquerda
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 possibilidades;
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); //{a,d}
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}); // Borda Esquerda (Subtrai)
78         eventos.push_back({right_x, bottom_y, top_y, 1}); // Borda Direita (Soma)
79
80         ys.push_back(bottom_y);
81         ys.push_back(top_y);
82     }
83     //compressao
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             // Insere a defesa do ponto na 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             // Soma ou subtrai dependendo da borda (Direita ou Esquerda)
102             resp += ev.type * count;
103         }
104     }
105
106     cout << resp << "\n";
107
108     return 0;
109 }

```

Código 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            if(tempo[et.last]==-1){
104                ++qtd;
105            }
106            if(tempo[et.last]!=i){
107                if(tempo[et.last]!=-1) prox[tempo[et.last]].push_back(i);
108                tempo[et.last] = i;
109            }
110        }
111
112        if(!existe[0]) existe[0] = 1, raiz[0] = perseg::update(i, qtd);
113        else raiz[0] = perseg::cur_update(i, qtd);
114
115        et.add('z'+1); et.add('z'+2);
116    }
117
118    int r = raiz[0];
119    for(int i=0; i<n-1; ++i){
120        for(int nxt:prox[i]){
121            if(nxt!=-1){
122                if(!existe[i+1]) existe[i+1] = 1, r = perseg::update(nxt, 1);
123                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 }

```

Código 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)//apenas se todos os elementos sao iguais
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 }

```

Código 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 // Quantos indices originais no intervalo [ql, qr] sao <= 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-l)/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     // Se flips eh par, ordem normal. Se impar, ordem invertida.
41     int k=(flips%2==0)?(mid_pos+1):(sz-mid_pos);
42
43     // Busca binaria pra adivinhar qual eh o indice original do pivo
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 }

```

Código 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 }

```

Código 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 }

```

Código 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  }
```

Código 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 { // ponto
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) { // distancia
39     return hypot(p.y - q.y, p.x - q.x);
40 }
41
42 struct line { // reta
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) { // area com sinal
52     return ((q-p)^(r-q))/2;
53 }
54
55 bool ccw(pt p, pt q, pt r) { // se p, q, r sao ccw
56     return sarea(p, q, r) > eps;
57 }
58
59 bool isinseg(pt p, line r) { // se p pertence ao seg de 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) { // se o seg de r intersecta o seg de s
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 // se o ponto ta dentro do poligono: retorna 0 se ta fora,
73 // 1 se ta no interior e 2 se ta na borda
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 }

```

Código 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             //Processo todas as arestas iguais de uma só vez.
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
54 void calc()
55 {
56     queue<int> q; q.push(S);
57     marc[S] = 1;
58     while(!q.empty())
59     {
60         int cur = q.front(); q.pop();
61         for(int i = 0; i < grafo[cur].size(); i++)
62         {
63             int viz = grafo[cur][i];
64             int mult = 1;
65
66             //Processo todas as arestas iguais de uma só vez.
67             while(i < grafo[cur].size() - 1 and grafo[cur][i + 1] == viz)
68             {
69                 i++;
70                 mult++;
71             }
72
73             if(dist[viz] == dist[cur])
74             {
75                 dp[cur][1] += (dp[viz][0]*mult)%MOD;
76                 dp[cur][1] %= MOD;
77             }
78
79             if(dist[cur] == dist[viz] + 1)
80             {
81                 dp[cur][1] += (dp[viz][1]*mult)%MOD;
82                 dp[cur][1] %= MOD;
83             }

```

```

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

```

Código 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 }

```

Código 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) : vector<vector<T>>(n_,vector<T>(m_,0)),n(n_),m(m_)
9      {
10         if(ident)
11         {
12             for(int i = 0; i < n; i++) (*this)[i][i] = 1;
13         }
14     }
15
16     matrix(const vector<vector<T>>& c) : vector<vector<T>>(c), n(c.size()),m (c[0].size()) {}
17
18     matrix<T> operator*(matrix<T>& r )
19     {
20         matrix<T> M(n,r.m);
21
22         for(int i = 0; i < n; i++) for(int k = 0; k < m; k++)
23         {

```

```

24         for(int j = 0; j < r.m; j++)
25         {
26             T add = (*this)[i][k]*r[k][j];
27             M[i][j] ^= add;
28         }
29     }
30     return M;
31 }
32
33 matrix<T> operator^(ll e)
34 {
35     matrix<T> M(n,n,true), at = *this;
36
37     while(e) {
38         if(e&1) M = M*at;
39         e >>= 1;
40         at = at*at;
41     }
42     return M;
43 }
44 };
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

```

```

73     vector<vector<int>> trans(N*N,vector<int>(N*N));
74
75     int dx[] = {1,1,1,-1,-1,-1,0,0,0};
76     int dy[] = {1,0,-1,1,0,-1,1,0,-1};
77     for(int i = 0; i < N; i++)
78     {
79         for(int j = 0; j < N; j++)
80         {
81             if(grid[i][j] == '#') continue;
82             for(int k = 0; k < 9; k++)
83             {
84                 int nx = i + dx[k];
85                 int ny = j + dy[k];
86
87                 if(nx < 0 or nx >= N or ny < 0 or ny >= N or grid[nx][ny] == '#') co
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 }

```

Código 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 }

```
