II OIFem (2022)

Final: Día 2

Soluciones

Olimpiada Informática Femenina

Saltando múltiplos

Autora del problema: Blanca Huergo Muñoz

Teoría

- Condicionales
- Bucles

Solución

Para este problema, bastaba con simular el enunciado y probar todas las posibilidades, implementando el código con cuidado para evitar bugs.

Código- C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;


int main() {
    int t;
    cin >> t;
    while(t--) {
        int n;
        cin >> n;
        int a[n];
        long long int suma = 0;
        for (int i = 0; i < n; i++) {
            cin >> a[i];
            suma += a[i];
        }

        bool ganaIzan = false;
        for (int k = 2; k <= n; k++) {
            long long int sumaActual = 0;
            int j = k;
            while(j <= n) {
                sumaActual += a[j-1];
                j += k;
            }
            if (sumaActual > suma-sumaActual) {
                ganaIzan = true;
                break;
            }
        }

        if (ganaIzan)
            cout << "SI\n";
```

```
            else cout << "NO\n";
        }
        return 0;
    }
```

Código- Python

```
t = int(input())
for xx in range(t):
    a = list(map(int, input().split()))
    n = a[0]
    a = a[1:]
    suma = sum(a)
    ganaIzan = False
    for k in range(2, n+1):
        sumaActual = sum([a[j-1] for j in range(k, n+1, k)])
        if sumaActual > suma-sumaActual:
            ganaIzan = True
            break
    if ganaIzan:
        print("SI")
    else:
        print("NO")
```

Arrancando páginas

Autora del problema: Victoria Durán Fernández

Teoría

- Progresiones aritméticas
- Factorizar polinomios
- Hacer mates con un lenguaje de programación, cuidando overflows y división de enteros

Solución

Usando la fórmula de la suma de progresiones aritméticas, tenemos que la suma de las caras de un libro (sin haber arrancado ninguna hoja) es $1+2+3+4+\ldots+(2n-1)+2n = \frac{2n(2n+1)}{2} = n(2n+1)$. Por tanto, tras arrancar las páginas $2k-1$ y $2k$, lo que nos queda es que $S = n(2n+1)-(4k-1)$.

Como $4k-1 \geq 0$, tenemos que necesariamente $n(2n+1)-S \geq 0$. Como conocemos S, esto es simplemente una desigualdad cuadrática en n: si hallamos las raíces del polinomio (en la variable n) $2n^2+n-S$, vemos (por Cardano-Vieta) que una raíz es positiva y otra negativa. Nuestro n buscado ha de ser mayor que dicha raíz positiva, que es precisamente (por la fórmula de las ecuaciones de segundo grado) $\frac{-1+\sqrt{8S+1}}{4}$. Así que simplemente tenemos que hallar dicho número (¡cuidado con posibles overflows al calcular $8S$!) y redondearlo hacia arriba para obtener el n que buscamos.

Una vez tenemos n, se comprueba que simplemente despejando obtenemos $k = \frac{n(2n+1)-S+1}{4}$, y ya podemos imprimir $2k-1$ y $2k$ para terminar el problema.

Código- C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <assert.h>
using namespace std;

int main() {
    int t;
    cin >> t;
    assert(1 <= t && t <= 100000);
    while(t--) {
        long long int S;
        cin >> S;
        long long int n = (long long int) (sqrt(0.5*S+0.0625)-0.25) + 1;
        long long int k = (n*(2*n+1) - S + 1)/4;
        assert (1 <= n && n <= 1000000000);
        assert(1 <= k && k <= n);
        cout << 2*k-1 << ' ' << 2*k << endl;
    }
    return 0;
}
```

Código- Python

```python
from math import sqrt

t = int(input())
for xx in range(t):
    S = int(input())
    n = int(sqrt(8*S+1)-1)//4 + 1
    k = (n*(2*n+1) - S + 1)//4
    print(2*k-1, 2*k)
```

Carrera de obstáculos

Autor del problema: Jose Mallma

Teoría

- Greedy
- Two pointers
- Búsqueda binaria

Solución

Este problema se podía implementar como greedy + two pointers ($O(N \log N)$) o con búsqueda binaria ($O(N \log K)$), la solución incluida aquí. La idea consiste en sumar a cada altura la energía necesaria para llegar a ella y, tras esto, hacer que `h[i]` sea la mayor de `h[0..i]`. Podemos hacer una búsqueda binaria sobre este array por cada perro, encontrando hasta dónde llega.

Código- C++

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <utility>
#include <vector>
#include <algorithm>
#include <map>
using namespace std;

int n, k;

void solve(vector<int> & e, vector<int> & h) {
    h[0]++;
    for(int i=1;i<k;i++){
        h[i]+=i+1;
        h[i] = max(h[i], h[i-1]);
    }
    vector<int> ans(n, 0);
    for (int i = 0; i < n; i++) {
        int low = 0, high = k-1;
        while (low <= high) {
            int mid = low + (high-low)/2;
            if (h[mid] <= e[i]) {
                ans[i] = mid+1;
                low = mid+1;
            } else {
                high = mid-1;
            }
        }
    }
    cout << ans[0];
    for (int i = 1; i < n; i++)
        cout << ' ' << ans[i];
    cout << '\n';
```

```cpp
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int t;
    cin>>t;
    while(t--){
        cin>>n>>k;
        vector<int>e(n);
        vector<int>h(k);
        for(int i=0;i<n;i++){
            cin>> e[i];
        }
        for(int i=0;i<k;i++){
            cin>>h[i];
        }
        solve(e, h);
    }
    return 0;
}
```

Código- Python

```python
import sys
t = int(sys.stdin.readline())
for xx in range(t):
    n, k = map(int, sys.stdin.readline().split())
    e = list(map(int, sys.stdin.readline().split()))
    h = list(map(int, sys.stdin.readline().split()))
    h[0] += 1
    for i in range(1, k):
        h[i]+=i+1
        h[i] = max(h[i], h[i-1])

    ans = ["0" for i in range(n)]
    for i in range(n):
        low = 0
        high = k-1
        while (low <= high):
            mid = low + (high-low)//2
            if (h[mid] <= e[i]):
                ans[i] = str(mid+1)
                low = mid+1
            else:
                high = mid-1
    sys.stdout.write(" ".join(ans))
    sys.stdout.write('\n')
```

Potencias enormes 2

Autor del problema: Félix Moreno Peñarrubia

Teoría

- Criba de Eratóstenes
- Búsqueda binaria
- Fórmula de Legendre para contar primos en factoriales (clasificatorio)

Solución

Encontraremos k haciendo una búsqueda binaria entre sus posibles valores. Por tanto, a partir de ahora, podemos suponer que conocemos k, y únicamente tenemos que comprobar si el enunciado es cierto o no para dicho k.

Para ello, descompongamos n en factores primos: $n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_r^{\alpha_r}$, y por tanto $n^m = p_1^{\alpha_1 m} \cdot p_2^{\alpha_2 m} \cdot \ldots \cdot p_r^{\alpha_r m}$. Vamos a resolver el problema cada factor primo de n por separado, y después argumentaremos que podemos unir los resultados de todos estos primos y obtener la solución de nuestro problema.

Tenemos que contestar entonces la pregunta: dado un primo p y un natural α, ¿existen k naturales consecutivos cuyo producto no sea divisible por p^α?

El producto de k naturales consecutivos lo podemos expresar de la forma $a(a+1) \cdot \ldots \cdot (a+k-1)$ para cierto natural a. Como de cada p números naturales consecutivos, uno ha de ser necesariamente múltiplo de p, al menos tenemos $\lfloor \frac{k}{p} \rfloor$ múltiplos de p entre los números $a, a+1, \ldots, a+k-1$.

De manera similar, como de cada p^2 naturales consecutivos hay necesariamente uno que es múltiplo de p^2, hay al menos $\lfloor \frac{k}{p^2} \rfloor$ múltiplos de p^2 entre los números $a, a+1, \ldots, a+k-1$. Como los múltiplos de p^2 también lo son de p, en realidad estos nuevos números solo aportan (cada uno) un nuevo factor p al producto $a(a+1) \cdot \ldots \cdot (a+k-1)$ (el otro factor p ya lo tuvimos en cuenta en el párrafo anterior).

Si seguimos razonando de la misma forma con potencias sucesivas de p, encontramos que la cantidad de factores p que hay necesariamente en $a(a+1) \cdot \ldots \cdot (a+k-1)$ es

$$\left\lfloor \frac{k}{p} \right\rfloor + \left\lfloor \frac{k}{p^2} \right\rfloor + \left\lfloor \frac{k}{p^3} \right\rfloor + \ldots = \sum_{i=1}^{\infty} \left\lfloor \frac{k}{p^i} \right\rfloor,$$

que es básicamente la fórmula de Legendre para saber cuántos factores p tiene $k!$.

De esta forma, es sencillo darse cuenta de que si $\sum_{i=1}^{\infty} \left\lfloor \frac{k}{p^i} \right\rfloor \geq \alpha$, entonces necesariamente p^α divide a $a(a+1) \cdot \ldots \cdot (a+k-1)$ para cualquier natural a. Mientras que si $\sum_{i=1}^{\infty} \left\lfloor \frac{k}{p^i} \right\rfloor < \alpha$, entonces siempre existirá algún natural a de forma que p^α no divida a $a(a+1) \cdot \ldots \cdot (a+k-1)$ (simplemente habrá que elegir a con una determinada congruencia módulo p^α).

Con esto ya podemos terminar el problema: tras factorizar n, comprobamos si para cada $j = 1, 2, \ldots, r$ se tiene que $\sum_{i=1}^{\infty} \left\lfloor \frac{k}{p_j^i} \right\rfloor \geq \alpha_j m$ (o si se tiene el $<$). En cuanto haya algún j para el cual se da el $<$, simplemente sabemos que habrá un a asociado que cumple que $p_j^{\alpha_j m}$ no divide a $a(a+1) \cdot \ldots \cdot (a+k-1)$, y por tanto tampoco lo hará n^m. En cambio, si para todos los j se tiene el $\geq$, necesariamente n^m divide a $a(a+1) \cdot \ldots \cdot (a+k-1)$ para cualquier natural a. Con esto ya podemos decidir si el k que teníamos fijado cumple el enunciado o no. Algunos comentarios para la implementación:

- Realmente no hace falta descomponer n en factores primos para cada k que nos salga en nuestra búsqueda binaria: podemos descomponer n al principio y reutilizar el resultado para cada k.
- Realmente en las sumas $\sum_{i=1}^{\infty} \left\lfloor \frac{k}{p^i} \right\rfloor$ solo hay que calcular una cantidad finita de términos: en cuanto $\left\lfloor \frac{k}{p^i} \right\rfloor = 0$ (lo cual ocurre necesariamente para i suficientemente frande), podemos parar de calcular la suma.
- En C++ hay que tener cuidado con los overflows al usar long long, pues hay algunos cálculos que exceden 10^{19}.
- Para los propósitos de este problema, dados los límites de n y m, basta buscar k entre 0 y 10^{18} (por lo tanto la búsqueda binaria hará como mucho $\log_2(10^{18}) \approx 60$ iteraciones).

Código- C++

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

typedef long long ll;

bitset<100'000> bs; // 10^5
vector<ll> primos;
map<ll, ll> factores;

// Criba de Eratostenes
void criba() {
    bs.set();
    bs[0] = bs[1] = 0;
    for (ll i = 2; i < bs.size(); i++) if (bs[i]) {
        for (ll j = i * i; j < bs.size(); j += i) bs[j] = 0;
        primos.push_back(i);
    }
}

// Se parece a la formula de Legendre para contar primos en factoriales
ll legendre(ll p, ll k){
    ll acum = 0, P = p;
    while(k / P != 0) {
        acum += k/P;
        if(P > 1e18/p) break;
        P *= p;
    }
    return acum;
}

// Por CRT, basta fijarse en cada factor primo de n^m por separado:
// si tenemos un factor primo p, basta con que haya menos factores
// en k consecutivos cualesquiera que los que hay en n^m
bool divisible(ll k){
    for(auto it = factores.begin(); it != factores.end(); ++it)
        if(legendre(it->first, k) < it->second) return false;
    return true;
}

int main(){
```

```
    criba();
    int tt; cin >> tt;
    while(tt--){
        ll n, m; cin >> n >> m;
        if(n == 1) {cout << "0\n"; continue;}
        // Factorizamos n, con multiplicidades
        factores.clear();
        ll P_id = 0, P = primos[0];
        while (P * P <= n) {
            if(n % P == 0){
                ll mult = 0;
                while (n % P == 0) {
                    n /= P;
                    mult++;
                }
                factores[P] = mult * m;
            }
            P = primos[++P_id];
        }
        if (n != 1) factores[n] = m; // N es primo
        // Ahora calculamos el minimo k con busqueda binaria
        ll l = 0, r = 1e18-100;
        while(l + 1 < r){
            ll m = (l+r)/2;
            if(divisible(m)) r = m-1;
            else l = m;
        }
        ll act;
        for(act = r; act >= l; --act){
            if(!divisible(act)) break;
        }
        cout << act << "\n";
    }
    return 0;
}
```

Código- Python

```
def times(n, p):
    P = p
    cnt = 0
    while P <= n:
        cnt += n//P
        P *= p
    return cnt

def candidate(p, e):
    lo = 1
    hi = 10**50
    k = 0
    while lo <= hi:
        mid = (lo+hi)//2
        if times(mid, p) < e:
            k = mid
```

```
            lo = mid+1
        else:
            hi = mid-1
    return k


t = int(input())
for _ in range(t):
    n, m = map(int, input().split())
    ans = 0
    p = 2
    while p*p <= n:
        if n % p == 0:
            e = 0
            while n % p == 0:
                n //= p
                e += 1
            e *= m
            ans = max(ans, candidate(p, e))
        p += 1
    if n > 1:
        ans = max(ans, candidate(n, m))
    print(ans)
```

¿Es una expansión?

Autor del problema: Félix Moreno Peñarrubia

Teoría

- Grafos bipartitos

Solución

Debemos comprobar que para cada componente conexa, se cumple que esta se puede partir en dos tal que una corresponda a los vértices del grafo original y la otra a los vértices añadidos. Sabremos que este es el caso si una de las dos partes cumple que todos sus vértices tienen grado 2 y no se conecta dos veces el mismo par de vértices en la otra componente.

Código- C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <unordered_set>
using namespace std;

int n, m;
vector<vector<int>> adjList;
vector<int> color, CC0, CC1;
int ccActual;
bool posible;

bool dfs(int x) {
    if (!posible)
        return false;
    for (int & vecino: adjList[x]) {
        if (color[vecino] == -1) {
            color[vecino] = ccActual + (1 - (color[x]-ccActual));
            if (color[vecino] == ccActual)
                CC0.push_back(vecino);
            else
                CC1.push_back(vecino);
            if (!dfs(vecino))
                return posible = false;
        } else if (color[vecino] == color[x])
            return posible = false;
    }
    return true;
}

bool componenteEsExpansion(int cc) {
    bool todoBien = true;
    unordered_set<string> parejasConectadas;
    for (int & nodo: CC0) {
        if (adjList[nodo].size() != 2) {
```

```
            todoBien = false;
            break;
        }
        int a = adjList[nodo][0];
        int b = adjList[nodo][1];
        if (a > b)
            swap(a, b);
        string S = to_string(a) + "," + to_string(b);
        if (parejasConectadas.find(S) != parejasConectadas.end()) {
            todoBien = false;
            break;
        }
        parejasConectadas.insert(S);
    }
    if (todoBien)
        return true;

    todoBien = true;
    parejasConectadas.clear();
    for (int & nodo: CC1) {
        if (adjList[nodo].size() != 2) {
            todoBien = false;
            break;
        }
        int a = adjList[nodo][0];
        int b = adjList[nodo][1];
        if (a > b)
            swap(a, b);
        string S = to_string(a) + "," + to_string(b);
        if (parejasConectadas.find(S) != parejasConectadas.end()) {
            todoBien = false;
            break;
        }
        parejasConectadas.insert(S);
    }
    return todoBien;
}

bool solve() {
    color.assign(n, -1);
    ccActual = 0;
    posible = true;
    for (int i = 0; i < n; i++) {
        if (color[i] == -1) {
            color[i] = ccActual;
            CC0.clear();
            CC0.push_back(i);
            CC1.clear();
            dfs(i);
            if (!posible)
                return false;
            if (!componenteEsExpansion(ccActual))
                return false;
            ccActual += 2;
        }
```

```
    }
    return true;
}


int main() {
    int t;
    cin >> t;
    while(t--) {
        cin >> n >> m;
        adjList = vector<vector<int>>(n);
        for (int i = 0; i < m; i++) {
            int u, v;
            cin >> u >> v;
            adjList[u].push_back(v);
            adjList[v].push_back(u);
        }
        if (solve())
            cout << "SI\n";
        else
            cout << "NO\n";
    }
    return 0;
}
```

Código- Python

```
from collections import deque

t = int(input())

# devuelve False si el grafo no es bipartito y pinta los vertices
def pintar_bfs(comienzo, color, adj, componente):
    q = deque()
    q.append(comienzo)
    color[comienzo] = 0

    while len(q) > 0:
        a = q.popleft()
        componente.append(a)

        for b in adj[a]:
            if color[b] == color[a]:
                return False
            elif color[b] == -1:
                color[b] = 0 if color[a] == 1 else 1
                q.append(b)

    return True

while t > 0:
    t -= 1

    line = input()
```

```
    n = int(line.split(" ")[0])
    m = int(line.split(" ")[1])

    adj = [[] for i in range(n)]

    for i in range(m):
        line = input()
        x = int(line.split(" ")[0])
        y = int(line.split(" ")[1])

        adj[x].append(y)
        adj[y].append(x)

    posible = True
    color = [-1 for i in range(n)]
    edges = set()
    componente = []

    for i in range(n):
        if color[i] != -1:
            continue

        posible = pintar_bfs(i, color, adj, componente)

        if posible == False:
            break

        # los vertices de color 0/1 pueden ser aristas?
        aristas = [True, True]

        for x in componente:
            if color[x] == 0 and len(adj[x]) != 2:
                aristas[0] = False
            elif color[x] == 1 and len(adj[x]) != 2:
                aristas[1] = False

        if (not aristas[0]) and (not aristas[1]):
            posible = False
            break

        # habria algun edge duplicado en el grafo original (no expandido)?
        color_aristas = 0 if aristas[0] == True else 1

        for x in componente:
            if color[x] == color_aristas:
                a = min(adj[x][0], adj[x][1])
                b = max(adj[x][0], adj[x][1])
                if (a, b) in edges:
                    posible = False
                    break
                else:
                    edges.add((a, b))

        componente = []

```

```
83      if posible:
84          print("SI")
85      else:
86          print("NO")
```

Girando espejos

Autora del problema: Blanca Huergo Muñoz

Teoría

- Programación dinámica
- 0-1 BFS / Dijkstra

Solución

En primer lugar, para cada punto de interés (espejos y el punto inicial y final), calculamos el punto de interés más cercano para cada una de las 4 direcciones. Esto lo podemos hacer en $\mathcal{O}(klog(k))$ ordenando los puntos de interes primero por la coordenada x y luego por la y, obteniendo así los puntos de interés más cercanos que esten arriba y abajo. Luego ordenamos los puntos de interes primero por la coordenada y y luego por la x para obtener los puntos de interés más cercanos hacia la izquierda y derecha.

Ahora consideramos una programación dinámica, donde dp$[v][d]$ = el mínimo número de espejos que hemos cambiado para llegar del punto inicial al punto de interés v con dirección d. Sobre los estados de la dinámica hacemos un 0-1 BFS (o un Dijkstra, ya que el cuello de botella era ordenar los puntos). En particular, si estamos en el estado v, d, con dp$[v][d] = c$, y sea d' la dirección a la que iríamos si no movemos el espejo. Podemos ir a nuestro vecino más cercano en dirección d' con coste c o a los vecinos de las otras tres direcciones con coste $c + 1$ y con la dirección correspondiente.

Código- C++

```cpp
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <set>
#include <queue>

using namespace std;
typedef vector <int> vi;
typedef vector <vi> vvi;

vvi G;
vi C; // 0 ->. 1 \>. 2 <-. 3 <\.

int next_dir(int current, int obstacle){
    if (obstacle == -1) return current;
    if (obstacle == 0 and (current & 1)) return current;
    if (obstacle == 0) return (current + 2) % 4;
    if (obstacle == 2 and (current & 1)) return (current + 2) % 4;
    if (obstacle == 2) return current;
    if (obstacle == 1 and current == 0) return 1;
    if (obstacle == 1 and current == 1) return 0;
    if (obstacle == 1 and current == 2) return 3;
    if (obstacle == 1 and current == 3) return 2;
    if (obstacle == 3 and current == 0) return 3;
    if (obstacle == 3 and current == 1) return 2;
    if (obstacle == 3 and current == 2) return 1;
```

```
    if (obstacle == 3 and current == 3) return 0;

    return -1; //shouldn't happen
}

int dijkstra(){
    int n = G.size();

    priority_queue <vi> Q;
    Q.push({-0, 0, 0});
    vvi visited(n, vi(4, 1e9));
    visited[0][0] = 0;

    while (not Q.empty()){
        vi aux = Q.top(); Q.pop();

        int d = -aux[0];
        int v = aux[1];
        int dir = aux[2];

        if (d > visited[v][dir]) continue;
        if (v == 1) return d;

        int follow_dir = next_dir(dir, C[v]);

        for (int i = 0; i < 4; ++i){
            if (G[v][i] == -1) continue;

            if (i == follow_dir and visited[G[v][i]][i] > d) {
                visited[G[v][i]][i] = d;
                Q.push({-d, G[v][i], i});
            }
            else if (v != 0 and visited[G[v][i]][i] > d + 1) {
                visited[G[v][i]][i] = d + 1;
                Q.push({-(d+1), G[v][i], i});
            }
        }
    }

    return -1;
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int t;
    cin >> t;

    for (int ww = 0; ww < t; ++ww){
        int m, n, k;
        cin >> m >> n;

        int x1, y1, x2, y2;
        cin >> y1 >> x1 >> y2 >> x2;
```

```
        y1 *= -1;
        y2 *= -1;

        cin >> k;

        vvi V(k, vi(2));
        C = vi(k+2, -1);

        for (int i = 0; i < k; ++i){
            cin >> V[i][1] >> V[i][0] >> C[i+2];
            V[i][1] *= -1;
        }

        vvi ver(k+2, vi(3));
        vvi hoz(k+2, vi(3));

        hoz[0] = {x1, y1, 0};
        ver[0] = {y1, x1, 0};
        hoz[1] = {x2, y2, 1};
        ver[1] = {y2, x2, 1};

        for (int i = 0; i < k; ++i){
            hoz[i+2] = {V[i][0], V[i][1], 2+i};
            ver[i+2] = {V[i][1], V[i][0], 2+i};
        }

        sort(hoz.begin(), hoz.end());
        sort(ver.begin(), ver.end());

        G = vvi(k+2, vi(4, -1));

        for (int i = 1; i < hoz.size(); ++i){
            if (hoz[i-1][0] != hoz[i][0]) continue;
            int x = hoz[i-1][2], y = hoz[i][2];

            G[x][3] = y;
            G[y][1] = x;
        }

        for (int i = 1; i < ver.size(); ++i){
            if (ver[i-1][0] != ver[i][0]) continue;
            int x = ver[i-1][2], y = ver[i][2];

            G[x][0] = y;
            G[y][2] = x;
        }

        cout << dijkstra() << '\n';
    }
}
```

Código- Python

```python
import heapq

def next_dir(current, obstacle):
    if (obstacle == -1): return current;
    if (obstacle == 0 and (current & 1)): return current;
    if (obstacle == 0): return (current + 2) % 4;
    if (obstacle == 2 and (current & 1)): return (current + 2) % 4;
    if (obstacle == 2): return current;
    if (obstacle == 1 and current == 0): return 1;
    if (obstacle == 1 and current == 1): return 0;
    if (obstacle == 1 and current == 2): return 3;
    if (obstacle == 1 and current == 3): return 2;
    if (obstacle == 3 and current == 0): return 3;
    if (obstacle == 3 and current == 1): return 2;
    if (obstacle == 3 and current == 2): return 1;
    if (obstacle == 3 and current == 3): return 0;

def dijkstra(G, C):
    n = int(len(G))

    Q = []
    heapq.heappush(Q, (0, 0, 0))

    visited = [[1e9 for _ in range(4)] for i in range(n)]

    visited[0][0] = 0

    while (len(Q)):
        aux = heapq.heappop(Q)

        d = aux[0]
        v = aux[1]
        current = aux[2]

        if (d > visited[v][current]):
            continue

        if (v == 1):
            return d

        follow_dir = next_dir(current, C[v])

        for i in range(4):
            if (G[v][i] == -1):
                continue

            if (i == follow_dir and visited[G[v][i]][i] > d):
                visited[G[v][i]][i] = d
                heapq.heappush(Q, (d, G[v][i], i))

            elif (v != 0 and visited[G[v][i]][i] > d + 1):
                visited[G[v][i]][i] = d + 1
                heapq.heappush(Q, (d+1, G[v][i], i))

    return -1
```

```
56
57
58  t = int(input())
59
60  for _ in range(t):
61      arr = input().split(' ')
62      m = int(arr[0])
63      n = int(arr[1])
64      y1 = -int(arr[2])
65      x1 = int(arr[3])
66      y2 = -int(arr[4])
67      x2 = int(arr[5])
68
69      k = int(input())
70
71      C = [-1 for _ in range(k+2)]
72      V = [0 for _ in range(k)]
73
74      arr = input().split(' ')
75
76      for i in range(0, 3*k, 3):
77          indx = i // 3
78
79          V[indx] = [int(arr[i+1]), -int(arr[i])]
80          C[indx+2] = int(arr[i+2])
81
82      ver = [[]]
83      hoz = [[]]
84
85      hoz[0] = [x1, y1, 0]
86      ver[0] = [y1, x1, 0]
87      hoz.append([x2, y2, 1])
88      ver.append([y2, x2, 1])
89
90      for i in range(k):
91          hoz.append([V[i][0], V[i][1], i+2])
92          ver.append([V[i][1], V[i][0], i+2])
93
94      hoz = sorted(hoz)
95      ver = sorted(ver)
96
97      G = [[-1 for _ in range(4)] for i in range(k+2)]
98
99      for i in range(1, len(hoz)):
100         if (hoz[i-1][0] != hoz[i][0]):
101             continue
102
103         x = hoz[i-1][2]
104         y = hoz[i][2]
105
106         G[x][3] = y
107         G[y][1] = x
108
109     for i in range(1, len(ver)):
110         if (ver[i-1][0] != ver[i][0]):
```

```
111                 continue
112
113             x = ver[i-1][2]
114             y = ver[i][2]
115
116             G[x][0] = y
117             G[y][2] = x
118
119         print(dijkstra(G, C))
```

Sábado, 9 de abril de 2022