Book – Problemas – Dani

Algorithms grafos

```cpp
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
#define fast ios_base::sync_with_stdio(false); cin.tie(NULL);
//#define IOF freopen("hps.in", "r", stdin); freopen("hps.out", "w",
stdout); //caso archivos

#define MAX_N 1005          // Tamaño máximo para arrays o tablas DP
#define INF 1e9             // Valor infinito para inicializar tablas DP
#define LL_INF LLONG_MAX
#define MOD 1000000007      // Módulo para valores en programación dinámica

#define endl "\n"
#define f first
#define s second
#define pb push_back
#define ll long long int
#define ull unsigned long long // solo valores positivos
#define l size() //comentar en algunos algoritmos

#define all(x) x.begin(), x.end()

void solve() {

}

int main() {
    fast
    int t = 1;
    //cin>>t;
    while(t--)
    solve();
    return 0;
}
```

Dijkstra

```
void Dijkstra(vector<ll>& dist, vector<vector<pair<int, int>>>& graph, int
start){
    priority_queue<pair<long long, int>, vector<pair<long long, int>>,
greater<pair<long long, int>>> pq;
    //priority_queue<pair<ll, int>> pq;
    dist[start] = 0;
    pq.push({0, start});

    while (!pq.empty()) {
        int node = pq.top().second;
        ll x = pq.top().first;
        pq.pop();

        if (x != dist[node]) continue;

        for (auto [next, cost] : graph[node]) {
            if (dist[next] > dist[node] + cost) {
                dist[next] = dist[node] + cost;
                pq.push({dist[next], next});
            }
        }
    }
}
```

DFS

```
void DFS(int v, vector<bool>& visited, vector<vector<int>>& graph) {
    visited[v] = true;
    //cout << v << " ";
    for (int i = 0; i < graph[v].size(); ++i) {
        int x = graph[v][i];
        if (!visited[x])
            DFS(x, visited, graph);
    }
}
```

Algoritmo de Euclides

Lcm - Mínimo común múltiplo

```
int gcd(int a, int b){
    if(b == 0){
        return abs(a);
    }else{
        return gcd(b, a%b);
    }
}

int lcm(int a, int b){
    return a / gcd(a, b) * b;
}
```

Factorización de un numero entero

```
map<int, int> factorize(int n){
    // Usualmente guardamos pares de la forma (p, e), donde p es el primo y
e cuántas veces se repite
    // Podemos usar un vector de pairs o un map para ello
    map<int, int> fact;
    for(int p = 2; p*p <= n; p++){
        if(n % p == 0){ // Encontramos un divisor p no trivial de n, y tiene
que ser primo
            int exponent = 0;
            while(n % p == 0){
                // Mientras n sea divisible entre p, contamos cuántas veces
lo divide
                n /= p;
                exponent++;
            }
            fact[p] = exponent; // Guardamos el par (p, e) en la lista
        }
    }
    if(n > 1){ // Si llegamos a este punto, la n' restante a factorizar
tiene que ser primo, pues ningún primo menor a sqrt(n) la dividió
        fact[n] = 1;
    }
    return fact;
}
```

Caso factorizar n!

```
int legendre(int n, int p){

    // Hallamos el exponente de p en n!
    int sum = 0;
    int term = n / p; // Primer término de la suma
    while(term > 0){  // Iteramos mientras haya algo qué sumar
        sum += term;
        term /= p;    // Usamos la identidad floor(n / p^(i+1)) =
floor(floor(n / p^i) / p)
    }
    return sum;
}

// Antes de llamar a esta función, precalculamos los suficientes primos
map<int, int> factoriceFactorial(int n){
    map<int, int> fact;
    for(int p : primes){
        if(p > n) break;
        // Por cada primo p <= n, hallamos su exponente y lo guardamos
        fact[p] = legendre(n, p);
    }
    return fact;
}
```

Obtener primos de 1 a n

```
vector<int> getPrimes(int n){
    vector<int> primes;
    for(int i = 1; i <= n; i++){
        if(isPrime(i)){
            primes.push_back(i);
        }
    }
    return primes;
}
```

Atravez de la criba recomendada

```
vector<bool> sieve(int n){
    vector<bool> is(n+1, true); // arreglo de marcas inicializado en true,
es decir, todos los números son "primos" al inicio
    is[0] = is[1] = false;      // excepto el 0 y el 1
    for(int i = 4; i <= n; i += 2){
```

```
        is[i] = false; // marcamos todos los múltiplos mayores de 2 como
compuestos
    }
    for(int i = 3; i*i <= n; i += 2){ // comenzamos a iterar por los impares
desde el 3
        if(is[i]){ // si el número actual es primo, marcamos todos sus
múltiplos mayores
            for(int j = i*i; j <= n; j += 2*i){ // comenzamos en i*i y solo
iteramos por los impares
                is[j] = false;
            }
        }
    }
    return is;
}
```

```
vector<int> getPrimes(int n){
    vector<bool> is = sieve(n);
    vector<int> primes;
    for(int i = 1; i <= n; i++){
        if(is[i]){
            primes.push_back(i);
        }
    }
    return primes;
}
```

Obtener primos al mismo tiempo – mas recomendada

```
vector<int> getPrimes(int n){
    vector<bool> is(n+1, true);
    vector<int> primes = {2}; // introducimos el 2 a mano, pues solo
procesamos los primos impares por la optimización
    is[0] = is[1] = false;
    for(int i = 4; i <= n; i += 2){
        is[i] = false;
    }
    for(int i = 3; i <= n; i += 2){
        if(is[i]){
            primes.push_back(i);     // como i es primo, lo introducimos al
arreglo
            if((long long)i*i <= n){ // chequeo adicional para evitar
desbordar i*i
                for(int j = i*i; j <= n; j += 2*i){
```

```
                    is[j] = false;
                }
            }
        }
    }
    return primes; // ahora devolvemos el arreglo de primos
}
```

Obtener divisores de un numero

```
vector<int> getDivisors(int n){
    vector<int> divisors;
    for(int a = 1; a*a <= n; a++){ // Se recomienda usar a*a<=n en lugar de
a<=sqrt(n) por la precisión
        if(n % a == 0){
            int b = n / a;
            divisors.push_back(a);
            if(a != b) divisors.push_back(b);
        }
    }
    return divisors;
}
```

Test de primalidad

```
bool isPrime(int n){
    if(n == 1) return false; // El 1 sigue siendo un caso especial
    for(int d = 2; d*d <= n; d++){
        if(n % d == 0){
            return false; // Encontramos un divisor de n que no es ni 1 ni n
        }
    }
    return true; // Si llegamos hasta este punto, n tiene que ser primo
}
```

Criba de numeros divisors

```
vector<int> divsSieve(int n){
    vector<int> divs(n+1); // incializamos el arreglo en ceros
    for(int i = 1; i <= n; i++){
        for(int j = i; j <= n; j += i){ // iteramos por todos los múltiplos
de i
            divs[j]++;
        }
    }
```

```
    return divs;
}
```

Y la complejidad queda como

$$O(n + n/2 + n/3 + n/4 + \cdots) = O(n(1 + 1/2 + 1/3 + 1/4 + \cdots)) = O(n \log n).$$

Podemos modificar esta criba para obtener la suma de divisores en lugar del número de divisores, simplemente cambiando la línea 5 a `divs[j] += i`, o incluso obtener los divisores como tal para cada número en el rango, solo habría que cambiar la línea por `divs[j].push_back(i)` y que `divs` sea ahora un arreglo de arreglos.

Factor primo mas pequeño

```
vector<int> lpSieve(int n){
    vector<int> lp(n+1);           // arreglo del factor primo más pequeño
    iota(lp.begin(), lp.end(), 0); // usamos la STL para rellenar el arreglo
inicial
    for(int i = 4; i <= n; i += 2){
        lp[i] = 2; // 2 es el factor primo más pequeño de todos los números
pares
    }
    for(int i = 3; i*i <= n; i += 2){ // comenzamos a iterar por los impares
desde el 3
        if(lp[i] == i){ // si el número actual es primo, actualizamos todos
sus múltiplos mayores
            for(int j = i*i; j <= n; j += 2*i){ // comenzamos en i*i y solo
iteramos por los impares
                lp[j] = min(lp[j], i);
            }
        }
    }
    return lp;
}
```

Criba factor primo mas grande

```
vector<int> gpSieve(int n){
    vector<int> gp(n+1);           // arreglo del factor primo más grande
    iota(gp.begin(), gp.end(), 0); // usamos la STL para rellenar el arreglo
inicial
    for(int i = 2; i <= n; i++){
        if(gp[i] == i){ // si el número actual es primo, actualizamos todos
sus múltiplos mayores
            for(int j = 2*i; j <= n; j += i){
                gp[j] = i; // al siempre actualizar los múltiplos,
garantizamos siempre guardar el factor primo más grande
```

```
            }
        }
    }
    return gp;
}
```

Problemas resueltos

520B - Two Buttons

```
void solve(){
    int n, m, x = 0; cin>>n>>m;
    while (n != m)
    {
        if(m % 2 != 0 || n> m) m++;
        else m /= 2;
        x++;
    }
    cout<<x<< endl;
}
```

1971B - Different String

```
void solve(){
    string x; cin>>x;
    string y = x;
    set<char> z(all(x));
    if(z.l == 1) cout<<"NO"<<endl;
    else{
        while(y == x){
            next_permutation(all(x));
        }
        cout<<"YES"<<endl;
        cout<<x<<endl;
    }
}
```

Upper bound – bebida en tiendas

```
void solve(){
    int n = r();
    vector<int> tiendas(n);
    generate(all(tiendas), r);
    int dias = r();
    vector<int> p(dias);
    generate(all(p),r);
    sort(all(tiendas));
    for(int i = 0; i < p.l; i++){
        auto it = upper_bound(all(tiendas), p[i]);
        int pos = distance(tiendas.begin(), it);
        cout<<pos<<endl;
    }
}
```

Prefix sum

```
int main()
{
    fast
    int n,t; cin>>n>>t;
    vector<ll> dp(n+1, 0);
    for(int i = 1; i <= n; i++){
        int x; cin>>x;
        dp[i] = dp[i-1] + x;
        //cout<<dp[i]<<" ";
    }
    while(t--){
        int x,y; cin>>x>>y;
        cout<<dp[y] - dp[x-1]<<endl;
    }

    return 0;
}
```

AKBAR – soldados ciudad

```cpp
#include <bits/stdc++.h>
// Common file
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;
using namespace std;


using ll = long long;
using pi = pair<int, int>;
using ti = tuple<int, int, int>;
using tl = tuple<ll, ll, ll>;
using pil = pair<int, ll>;
using pli = pair<ll, int>;
using pl = pair<ll, ll>;
using vi = vector<int>;
using vl =  vector<ll>;
using vpi = vector<pi>;
using vpil = vector<pil>;
using vpli = vector<pli>;
using vpl = vector<pl>;
using vti = vector<ti>;
using vtl = vector<tl>;
using vvi = vector<vi>;
using vvl = vector<vl>;
using vvpi = vector<vpi>;
using vvpil = vector<vpil>;
using vvpli = vector<vpli>;
using vvpl = vector<vpl>;
using vvti = vector<vti>;
using vvtl = vector<vtl>;

using oset =  tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update>;

using omset =  tree<pair<int,int>, null_type, less<pair<int, int>>,
rb_tree_tag, tree_order_statistics_node_update>;

mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

const int INF = INT_MAX;
const ll LINF = LLONG_MAX;
```

```
const int MOD = 1000000000 + 7;

#define setbits(n) __builtin_popcount(n)

const int N = (int) 1e6;
vi g[N+1];

int visited[N+1];
vpi src;

bool bfs() {
    queue<pi> q;
    int sz = (int) src.size();
    for (int i = 0; i < sz; i++) {
        q.push({src[i].first, src[i].second});
        visited[src[i].first] = src[i].first;
    }
    while (!q.empty()) {
        int u = q.front().first;
        int strength = q.front().second;
        assert(visited[u]);
        q.pop();
        if (strength <= 0) {
            continue;
        }
        for (int v : g[u]) {
            if (!visited[v]) {
                visited[v] = visited[u];
                q.push({v, strength-1});
            } else {
                if (visited[v] == visited[u]) {
                    continue;
                }
                assert(visited[v] != visited[u]);
                return false;
            }
        }
    }
    return true;
}
void reset(int n) {
    for (int i = 1; i <= n; i++) {
        g[i].clear();
        visited[i] = 0;
    }
```

```
    src.clear();
}
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int T;
    cin >> T;
    while (T--) {
        int n, r, m;
        cin >> n >> r >> m;
        for (int i = 0; i < r; i++) {
            int u, v;
            cin >> u >> v;
            g[u].emplace_back(v);
            g[v].emplace_back(u);
        }

        for (int i = 0; i < m; i++) {
            int k, s;
            cin >> k >> s;
            src.emplace_back(k, s);
        }
        bool flag = bfs();
        if (!flag) {
            cout << "No\n";
        } else {
            bool ans = true;
            for (int i = 1; i <= n; i++) {
                if (!visited[i]) {
                    ans = false;
                    break;
                }
            }
            if (ans) {
                cout << "Yes\n";
            } else {
                cout << "No\n";
            }
        }
        reset(n);
    }
    return 0;
}
```

KQUERY - K-query – tripleta y numerios mayores a k,

```
#include<bits/stdc++.h>
using namespace std;
#define mx 30005
int tree[mx*4] = {0};
struct leaf
{
    int val;
    int ind;
}arr[mx];
bool compl1(leaf ob1, leaf ob2)
{
    return ob1.val>ob2.val;
}
struct query
{
    int k;
    int l;
    int r;
    int id;

}q[200005];
int ans[200005];
bool compq1(query ob1, query ob2)
{
    return ob1.k>ob2.k;
}
void update(int pos, int b, int e, int i, int j)
{
    if(b>e || b>j || e<i) return;
    if(b>=i && e<=j){
        tree[pos] = 1;
        return;
    }
    int l = pos*2+1;
    int r = l+1;
    int mid = (b+e)/2;
    update(l,b,mid,i,j);
    update(r, mid+1,e,i,j);
    tree[pos] = tree[l]+tree[r];
}
int Query(int pos, int b, int e, int i, int j)
{
    if(b>j || e<i || b>e) return 0;
    if(b>=i && e<=j) return tree[pos];
```

```
    int l = pos*2+1;
    int r = l+1;
    int mid = (b+e)/2;
    int r1 = Query(l,b,mid,i,j);
    int r2 = Query(r, mid+1,e,i,j);
    return r1 + r2;
}
int main()
{
    int n,qr,a;
    scanf("%d",&n);
    for(int i=0; i<n; i++){
        scanf("%d",&arr[i].val);
        arr[i].ind = i;
    }
    sort(arr, arr+n,compl1);
    int idl = 0;
    scanf("%d",&qr);
    for(int i=0; i<qr; i++){
        scanf("%d%d%d",&q[i].l,&q[i].r,&q[i].k);
        q[i].id=i;
        q[i].l--;
        q[i].r--;
    }
    sort(q,q+qr,compq1);
    for(int i=0; i<qr; i++){
        for(idl; idl<n && arr[idl].val> q[i].k; idl++){
            update(0,0,n-1,arr[idl].ind,arr[idl].ind);
        }
        ans[q[i].id] = Query(0,0,n-1,q[i].l, q[i].r);
    }
    for(int i=0; i<qr; i++){
        printf("%d\n",ans[i]);
    }
    return 0;
}
```

Xenia and bit operation

```cpp
#include <cstdio>

int num[20];
int tree[18][140000];

int main()
{
    num[0] = 1;
    for(int i=1; i<20; i++)
        num[i] = num[i-1] * 2;

    int n, m;
    scanf("%d%d", &n, &m);

    for(int i=0; i<num[n]; i++)
        scanf("%d", &tree[n][i]);

    int state = n % 2;

    for(int i=n-1; i>=0; i--)
        for(int j=0; j<num[i]; j++)
            if(i % 2 == state)
                tree[i][j] = tree[i+1][j*2] ^ tree[i+1][j*2+1];
            else
                tree[i][j] = tree[i+1][j*2] | tree[i+1][j*2+1];

    for(int i=0; i<m; i++)
    {
        int p, b;
        scanf("%d%d", &p, &b);

        tree[n][p-1] = b;
        int now = p-1;
        for(int j=n-1; j>=0; j--)
        {
            now = now/2;
            if(j % 2 == state)
                tree[j][now] = tree[j+1][now*2] ^ tree[j+1][now*2+1];
            else
                tree[j][now] = tree[j+1][now*2] | tree[j+1][now*2+1];
        }

        printf("%d\n", tree[0][0]);
    }
```

```
    return 0;
}
```

Espiral numérica

Una espiral numérica es una cuadrícula infinita cuyo cuadrado superior izquierdo tiene el número 1. Aquí están las primeras cinco capas de la espiral:

1	2	9	10	25
4	3	8	11	24
5	6	7	12	23
16	15	14	13	22
17	18	19	20	21

Tu tarea es descubrir el número en la fila.yy columna$X$.

```cpp
#include <iostream>
#include <bits/stdc++.h>
#include <math.h>
using namespace std;
#define fast ios_base::sync_with_stdio(false); cin.tie(NULL);

#define MAX_N 1005
#define INF 1e9
#define MOD 1000000007

#define endl "\n"
#define f first
#define s second
#define pb push_back
#define ll long long int
#define ull unsigned long long
#define l size()

#define all(x) x.begin(), x.end()

ll r(){
    ll x; cin>>x;
    return x;
}

void solve(){
    ll x = r(), y = r();
    ll mx = max(x,y);
```

```
    if(mx % 2 == 0){
        swap(x,y);
    }
    if(y == mx) cout<<mx*mx-x+1<<endl;
    else cout<<(mx-1)*(mx-1)+y<<endl;
}

int main() {
    fast
    int t = 1;
    cin>>t;
    while(t--)
    solve();
    return 0;
}
```

Military problema - Ejercito de berland entrega de comandos

```
#include<bits/stdc++.h>
using namespace std;
const int maxn = 2e5 + 10;
int n, m, a, b, c = 1, d;
bool vis[maxn] = { false };
vector<int> graph[maxn];
int level[maxn],pos[maxn],child[maxn];
int dfs(int src)
{
    vis[src] = true;
    pos[c] = src;
    level[src] = c++;
    int temp = 1;
    for (int i = 0; i<graph[src].size(); i++)
    {
        if (!vis[graph[src][i]]) temp += dfs(graph[src][i]);
    }
    return  child[src] = temp;
}
int main()
{
    cin >> n >> m;
    for (int i = 2; i <= n; i++)
    {
        cin >> a;
        graph[a].push_back(i);
```

```
    }
    dfs(1);
    while (m--)
    {
        cin >> a >> b;
        if (child[a]<b) cout << -1 << endl;
        else cout << pos[level[a] + b - 1] << endl;
    }
    return 0;
}
```

N segmentos de línea horizontales en un plano

```cpp
#include <bits/stdc++.h>
using namespace std;
#define INF 1000000000
#define endl '\n'
#define int long long
const int M = 800 + 2;
vector<pair<int,long long>> v[M];
map<int,int> mp;
vector<pair<int,int>> vv; int n, cnt = 1;
long long dis[M];
bool sol(long long mid) {
    for(int i = 0; i < cnt; i++) {
        v[i].clear();
        if (i > 1) v[i].push_back({i-1,0});
    }
    for(int i = 0; i < n; i++) {
        int a = vv[i].first, b = vv[i].second;
        v[a].push_back({b,mid});
        v[b].push_back({a,-1});
    }
    for(int i = 1; i < cnt; i++) {
        v[0].push_back({i,0});
    }
    for(int i = 1; i < cnt; i++) {
        dis[i] = INF;
    }
    dis[0] = 0;

    for(int i = 0; i < cnt; i++) {
        for(int j = 0; j < cnt; j++) {
            for(auto z : v[j]) {
                if(dis[j]+z.second < dis[z.first]) {

                    dis[z.first] = dis[j]+z.second;
                }
            }
        }
    }
    for(int j = 0; j < cnt; j++) {
            for(auto z : v[j]) {
                if(dis[j]+z.second < dis[z.first]) {
                    return false;
                }
            }
      }
     return true;
}
signed main()
```

```cpp
signed main()
{
    ios_base::sync_with_stdio(0);cout.tie(0);cin.tie(0);
    cin >> n; set<int> st;
    for(int i = 0; i < n; i++) {
        long long a, b, c; cin >> a >> b >> c;

        st.insert(4*a+1); st.insert(4*b-1);
        vv.push_back({4*a+1,4*b-1});
    }
    for(auto i : st) {

        mp[i] = cnt; cnt++;
    }
    for(int i = 0; i < n; i++) {
        vv[i].first = mp[vv[i].first];
        vv[i].second = mp[vv[i].second];
    }
    long long l = 1, r = 400, ans = 0;
    while(l <= r) {
        long long mid = (l+r)/2;
        if(sol(mid)) {
            ans = mid;
            r = mid-1;
        }
        else {
            l = mid+1;
        }
    }
    cout << ans<<endl;
     return 0;
}


```

Otro código del anterior

```cpp
#include <bits/stdc++.h>

using namespace std;

#pragma GCC optimize("Ofast,unroll-loops")

template<typename A, typename B> ostream& operator<<(ostream &os, const pair<A, B> &p) { return os << '(' << p.first << ", " << p.second << ')'; }
template<typename T_container, typename T = typename enable_if<!is_same<T_container, string>::value, typename T_container::value_type>::type> ostream& operator<<(ostream &os, const T_container &v) { os << '{'; string sep; for (const T &x : v) os << sep << x, sep = ", "; return os << '}'; }
void dbg_out() { cerr << endl; }
template<typename Head, typename... Tail> void dbg_out(Head H, Tail... T) { cerr << ' ' << H; dbg_out(T...); }
#ifdef LOCAL
#define dbg(...) cerr << "(" << #__VA_ARGS__ << "):", dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif

#define ar array
#define ll long long
#define ld long double
#define sza(x) ((int)x.size())
#define all(a) (a).begin(), (a).end()
#define pb push_back
#define f first
#define s second

typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
#define PI 3.14159265358979323846264338327950l
const int MAX_N = 1e5 + 5;
const ll MOD = 1e9 + 7;
const ll iINF = 1e9;
const ll lINF = 1e18;
const ld EPS = 1e-9;

```

```
34
35  ll n, t;
36
37  struct point {
38      ll x;
39      bool end;
40      ll id;
41  };
42  vector<point> points;
43
44  struct edge {
45      ll u, v, w;
46  };
47  vector<edge> edges;
48
49  vector<ll> dist;
50
51  bool relax() {
52      bool relaxed = false;
53      for (struct edge e : edges) {
54          if (dist[e.u] != iINF && dist[e.v] > dist[e.u] + e.w) {
55              relaxed = true;
56              dist[e.v] = dist[e.u] + e.w;
57          }
58      }
59      return relaxed;
60  }
61
62  bool bellman_ford() {
63      fill(dist.begin(), dist.end(), iINF);
64      dist[2*n] = 0;
65      for (ll i = 0; i < 2*n; i++) {
66          if (!relax()) break;
67      }
68      return relax();
69  }
70
71  bool test(ll r) {
72      // set interval lengths as r
73      for (edge &e : edges) {
74          if (e.w > 0) e.w = r;
75      }
76      return !bellman_ford();
77  }
78
```

```
int main() {
    cin.tie(0); ios::sync_with_stdio(0);
    cin >>  n;
    dist.resize(2*n+1);
    points.resize(2*n);
    for (ll i = 0; i < n; i++) {
        struct point p1 = {0, 0, i};
        struct point p2 = {0, 1, i+n};
        cin >> p1.x >> p2.x >> t;
        points[i] = p1;
        points[i+n] = p2;
        // f(n+i) - f(i) <= r
        edges.pb({i, n+i, iINF});
        // f(i) - f(n+i) <= -1
        edges.pb({n+i, i, -1});

        // source
        edges.pb({2*n, i, 0});
        edges.pb({2*n, i+n, 0});
    }
    // only make edges between consecutive/equal x values
    sort(points.begin(), points.end(), [](struct point a, struct point b){return a.x < b.x ;});
    // for (int i = 0; i < 2*n; i++) printf("%lld ", points[i].id);
    // printf("\n");
    for (ll i = 0; i < 2*n; i++) {
        struct point p1 = points[i];
        for (ll j = i+1; j < 2*n; j++) {
            // printf("j is %lld\n", j);
            struct point p2 = points[j];
            // printf("x values are %lld %lld\n", p1.x, p2.x);
            if (p1.x == p2.x) {
                // if equal and not end - start, add edge
                if (!p2.end || p1.end) edges.pb({p2.id, p1.id, 0});
                if (!p1.end || p2.end) edges.pb({p1.id, p2.id, 0});
            }
            if (p1.x < p2.x) {
                // printf("here %lld %lld\n", p1.id, p2.id);
                // if they are start/end of same interval ignore since we already dealt with
that
                if ((p1.id - p2.id) % n) edges.pb({p2.id, p1.id, 0});
                break;
            }
        }
    }
    // for (struct edge e : edges) printf("%lld %lld %lld\n", e.u, e.v, e.w);

    // bs, find min r for which bellman ford returns false
    ll lo = 1, hi = n, best_so_far = n+1;
    while (lo <= hi) {
        ll mid = (lo+hi)/2;
        if (test(mid)) {
            best_so_far = mid;
            hi = mid-1;
        } else {
            lo = mid+1;
        }
    }
    cout << best_so_far;
}
```

// no se cosas que estaban en ehatsap

```
RECORRIDO DE MATRIZ CON BFS

#include <iostream>
#include <vector>
#include <queue>
using namespace std;

const char START = 'S';
const char EXIT = 'E';
const char FREE = '.';
const char WALL = '#';

struct Point {
    int x, y;
    Point(int x, int y) : x(x), y(y) {}
};

void bfs(vector<vector<char>>& maze, Point start) {
    int rows = maze.size();
    int cols = maze[0].size();

    vector<vector<bool>> visited(rows, vector<bool>(cols, false));
    queue<Point> q;
    q.push(start);
    visited[start.x][start.y] = true;

    int dx[] = {0, 0, 1, -1}; // filas
    int dy[] = {1, -1, 0, 0}; // columnas

    while (!q.empty()) {
        Point current = q.front();
        q.pop();

        if (maze[current.x][current.y] == EXIT) {
            cout << "Se ha encontrado la salida. en : " << current.x << " "
<< current.y << endl;
            return;
        }

        for (int i = 0; i <4; i++) {
            int nx = current.x + dx[i];
            int ny = current.y + dy[i];
```

```
            if (nx >= 0 && nx < rows && ny >= 0 && ny < cols &&
!visited[nx][ny] &&
                (maze[nx][ny] == FREE || maze[nx][ny] == EXIT)) {
                q.push(Point(nx, ny));
                visited[nx][ny] = true;
            }
        }
    }
    cout << "No se ha encontrado la salida" << endl;
}

int main() {
     int M, N;
    cin >> M >> N; // Leer los valores de M, N y T desde la entrada estándar
    vector<vector<char>> maze(M, vector<char>(N)); //
    Point start(0, 0); // Crear un objeto Point para almacenar las
coordenadas del punto de inicio
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            cin >> maze[i][j]; // Leer los caracteres del laberinto desde la
entrada estándar y almacenarlos en la matriz maze

            if (maze[i][j] == START) {
                start.x = i;
                start.y = j;
            }
        }
    }
    bfs(maze, start);

    return 0;
}
```

BFS

```
BFS

#include <iostream>
#include <queue>
#include <vector>

using namespace std;

// Función para realizar el recorrido BFS
void BFS(vector<vector<int>>& grafo, int nodoInicial) {
```

```cpp
    // Crear una cola para almacenar los nodos que se van a visitar
    queue<int> cola;

    // Vector para marcar los nodos visitados
    vector<bool> visitado(grafo.size(), false);

    // Marcar el nodo inicial como visitado y agregarlo a la cola
    visitado[nodoInicial] = true;
    cola.push(nodoInicial);

    while (!cola.empty()) {
        // Obtener el siguiente nodo de la cola y mostrarlo
        int nodoActual = cola.front();
        cola.pop();
        cout << nodoActual << " ";

        // Recorrer todos los nodos adyacentes al nodo actual
        for (int i = 0; i < grafo[nodoActual].size(); ++i) {
            int nodoAdyacente = grafo[nodoActual][i];
            if (!visitado[nodoAdyacente]) {
                // Marcar el nodo adyacente como visitado y agregarlo a la
cola
                visitado[nodoAdyacente] = true;
                cola.push(nodoAdyacente);
            }
        }
    }
}

int main() {
    // Ejemplo de grafo representado mediante una lista de adyacencia
    vector<vector<int>> grafo = {
        {1, 2},     // Nodo 0: Conectado con nodos 1 y 2
        {0, 2, 3},  // Nodo 1: Conectado con nodos 0, 2 y 3
        {0, 1, 3},  // Nodo 2: Conectado con nodos 0, 1 y 3
        {1, 2, 4},  // Nodo 3: Conectado con nodos 1, 2 y 4
        {3}         // Nodo 4: Conectado con nodo 3
    };

    int nodoInicial = 0;
    cout << "Recorrido BFS comenzando desde el nodo " << nodoInicial << ":
";
    BFS(grafo, nodoInicial);
    return 0;
}
```

BFS DE LABERINTO

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <bits/stdc++.h>
using namespace std;
//librerias y macross
//CONSTANTES PARA OPTIMIZAR
const char START = 'S';
const char EXIT = 'E';
const char FREE = '.';
const char WALL = '#';

struct Point {
    int x, y;
    Point(int x, int y) : x(x), y(y) {}
};

//funcion BFS

void bfs(vector<vector<char>>& maze, Point start) {
    int rows = maze.size();
    int cols = maze[0].size();
        // Arreglo para marcar los nodos visitados
    vector<vector<bool>> visited(rows, vector<bool>(cols, false));

    queue<Point> q; //AQUI COMO YA NO USAMOS ENTEROS USAREMOS CORDENADAS
    q.push(start);
    visited[start.x][start.y] = true;
    /*
    Definimos dos arreglos dx y dy que representan las
    direcciones a las que podemos movernos desde un nodo:
    hacia arriba, abajo, izquierda y derecha. Estos
    arreglos nos permitirán explorar los vecinos del nodo
    actual.
    */
   int dx[] = {0, 0, 1, -1,+1,-1,1,-1};//filas
   int dy[] = {1, -1, 0, 0,-1,1,1,-1};//columnas
        // Explorar los vecinos del nodo actual
        while (!q.empty()) {
        Point current = q.front();
        q.pop();
        // Verificar si se ha llegado al punto de salida
        if (maze[current.x][current.y] == EXIT) {
```

```
            cout << "Se ha encontrado la salida. en : "<<current.x<<"
"<<current.y << endl;
            return;
        }
                // Explorar los vecinos del nodo actual
        for (int i = 0; i < 8; i++) {
                /*
                Dentro del bucle, exploramos los vecinos del nodo actual.
                Para cada dirección posible (arriba, abajo, izquierda,
derecha),
                calculamos las coordenadas del vecino (nx, ny) sumando
                los valores de dx[i] y dy[i] al valor de las coordenadas del
nodo actual.

                */

            int nx = current.x + dx[i];
            int ny = current.y + dy[i];

            // Verificar si la coordenada vecina está dentro del laberinto y
no ha sido visitada
            if (nx >= 0 && nx < rows && ny >= 0 && ny < cols &&
!visited[nx][ny] &&
                (maze[nx][ny] == FREE || maze[nx][ny] == EXIT )) {
                q.push(Point(nx, ny));
                visited[nx][ny] = true;
            }
        }
    }
    cout << "No se ha encontrado la salida." << endl;
}
int main() {
    vector<vector<char>> maze = {
        {'S', '#', '.', '#', '.'},
        {'#', 'E', '.', '.', '.'},
        {'.', '.', '.', '#', '.'},
        {'*', '#', '.', '.', '.'}
    };
    Point start(0, 0);
    bfs(maze, start);

}
```

```
BFS PRA EL I DEL LATINO
```

```
int solve(int nodoInicial) {

    int c=0;
    cout<<"SOLVE: NODO: ";
    // Crear una cola para almacenar los nodos que se van a visitar
    queue<int> cola;
    cout<<nodoInicial<<endl;
    // Vector para marcar los nodos visitados
    cout<<"TAM: "<<adj[1].size()<<endl;
    // Marcar el nodo inicial como visitado y agregarlo a la cola
    visited[nodoInicial] = true;
    cola.push(nodoInicial);

    while (!cola.empty()) {
        // Obtener el siguiente nodo de la cola y mostrarlo
        int nodoActual = cola.front();
        cola.pop();
        cout << nodoActual << " ";
        c++;
        // Recorrer todos los nodos adyacentes al nodo actual
        for (int i = 0; i < adj[nodoActual].size(); ++i) {
            int nodoAdyacente = adj[nodoActual][i];
            if (!visited[nodoAdyacente])
                // Marcar el nodo adyacente como visitado y agregarlo a la
cola
                visited[nodoAdyacente] = true;
                cola.push(nodoAdyacente);
            }
        }
    }
    if(c>maxn){
        maxn=c;
    }
}
```