Competitive Programming Notebook

Programadores Roblox

Contents

1 String

1.1 Countpermutations

```
// Returns the number of distinct permutations
// that are lexicographically less than the string t
// using the provided frequency (freq) of the characters
// O(n*freq.size())
int countPermLess(vector<int> freq, const string &t) {
    int n = t.size();
    int ans = 0;

    vector<int> fact(n + 1, 1), invfact(n + 1, 1);
    for (int i = 1; i <= n; i++)
        fact[i] = (fact[i - 1] * i) % MOD;
    invfact[n] = fexp(fact[n], MOD - 2, MOD);
    for (int i = n - 1; i >= 0; i--)
        invfact[i] = (invfact[i + 1] * (i + 1)) % MOD;

    // For each position in t, try placing a letter smaller than t[i] that is in freq
    for (int i = 0; i < n; i++) {
        for (char c = ’a’; c < t[i]; c++) {
            if (freq[c - ’a’] > 0) {
                freq[c - ’a’]--;
                int ways = fact[n - i - 1];
                for (int f : freq)
                    ways = (ways * invfact[f]) % MOD;
                ans = (ans + ways) % MOD;
                freq[c - ’a’]++;
            }
        }
        if (freq[t[i] - ’a’] == 0) break;
        freq[t[i] - ’a’]--;
    }
    return ans;
}
```

1.2 Trie Ponteiros

```
// Trie por ponteiros
// InserÃğÃčo, busca e consulta de prefixo em O(N)

struct Node {
    Node *filhos[26] = {};
    bool acaba = false;
    int contador = 0;
};

void insere(string s, Node *raiz) {
    Node *cur = raiz;
    for(auto &c : s) {
        cur->contador++;
        if(cur->filhos[c - ’a’] != NULL) {
            cur = cur->filhos[c - ’a’];
            continue;
        }
        cur->filhos[c - ’a’] = new Node();
        cur = cur->filhos[c - ’a’];
    }
    cur->contador++;
    cur->acaba = true;
}

bool busca(string s, Node *raiz) {
    Node *cur = raiz;
    for(auto &c : s) {
        if (cur->filhos[c - ’a’] != NULL) {
            cur = cur->filhos[c - ’a’];
            continue;
        }
        return false;
    }
    return cur->acaba;
}

// Retorna se Ãľ prefixo e quantas strings tem s como prefixo
int isPref(string s, Node *raiz) {
    Node *cur = raiz;
    for(auto &c : s) {
        if (cur->filhos[c - ’a’] != NULL) {
            cur = cur->filhos[c - ’a’];
            continue;
        }
        return -1;
    }
    return cur->contador;
}
```

1.3 Z Function

```
vector<int> z_function(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for(int i = 1; i < n; i++) {
        if(i < r) {
            z[i] = min(r - i, z[i - l]);
        }
        while(i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if(i + z[i] > r) {
            l = i;
            r = i + z[i];
        }
    }
    return z;
}
```

1.4 Hashing

```
// String Hash template
// constructor(s) - O(|s|)
// query(l, r) - returns the hash of the range [l,r] from left to right - O(1)
// query_inv(l, r) from right to left - O(1)
// patrocinado por tiagodfs

struct Hash {
    const int X = 2147483647;
    const int MOD = 1e9+7;
    int n; string s;
    vector<int> h, hi, p;
    Hash() {}
    Hash(string s): s(s), n(s.size()), h(n), hi(n), p(n) {
        for (int i=0;i<n;i++) p[i] = (i ? X*p[i-1]:1) % MOD;
        for (int i=0;i<n;i++)
            h[i] = (s[i] + (i ? h[i-1]:0) * X) % MOD;
        for (int i=n-1;i>=0;i--)
            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * X) % MOD;
    }
    int query(int l, int r) {
        int hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
    int query_inv(int l, int r) {
        int hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l+1] % MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
};
```

1.5 Kmp

```
vector<int> kmp(string s) {
    int n = (int)s.length();
    vector<int> p(n+1);
    p[0] = -1;
    for (int i = 1; i < n; i++) {
        int j = p[i-1];
        while (j >= 0 && s[j] != s[i-1])
            j = p[j-1];
        p[i] = j+1;
    }
    return p;
}
```

1.6 Lcs

```
int lcs(string &s1, string &s2) {
    int m = s1.size();
    int n = s2.size();

    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));

    for (int i = 1; i <= m; ++i) {
        for (int j = 1; j <= n; ++j) {
            if (s1[i - 1] == s2[j - 1])
                dp[i][j] = dp[i - 1][j - 1] + 1;
            else
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
        }
    }

    return dp[m][n];
}
```

2 DS

2.1 Segtree Gcd

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

class SegmentTreeGCD {
private:
    vector<int> tree;
    int n;

    void build(const vector<int>& arr, int node, int start, int end) {
        if (start == end) {
            tree[node] = arr[start];
        } else {
            int mid = (start + end) / 2;
            build(arr, 2 * node + 1, start, mid);
            build(arr, 2 * node + 2, mid + 1, end);
            tree[node] = gcd(tree[2 * node + 1], tree[2 * node + 2]);
        }
    }

    void update(int node, int start, int end, int idx, int value) {
        if (start == end) {
            tree[node] = value;
        } else {
            int mid = (start + end) / 2;
            if (idx <= mid) {
                update(2 * node + 1, start, mid, idx, value);
            } else {
                update(2 * node + 2, mid + 1, end, idx, value);
            }
            tree[node] = gcd(tree[2 * node + 1], tree[2 * node + 2]);
        }
    }

    int query(int node, int start, int end, int l, int r) {
        if (r < start || l > end) {
            return 0;
        }
        if (l <= start && end <= r) {
            return tree[node];
        }
        int mid = (start + end) / 2;
        int left_gcd = query(2 * node + 1, start, mid, l, r);
        int right_gcd = query(2 * node + 2, mid + 1, end, l, r);
        return gcd(left_gcd, right_gcd);
    }

public:
    SegmentTreeGCD(const vector<int>& arr) {
        n = arr.size();
        tree.resize(4 * n);
        build(arr, 0, 0, n - 1);
    }
    void update(int idx, int value) {
        update(0, 0, n - 1, idx, value);
    }
    int query(int l, int r) {
        return query(0, 0, n - 1, l, r);
    }
};
```

2.2 Bit

```
class BIT {
    vector<int> bit;
    int n;
    int sum(int idx) {
        int result = 0;
        while (idx > 0) {
            result += bit[idx];
            idx -= idx & -idx;
        }
        return result;
    }

public:
    BIT(int size) {
        n = size;
        bit.assign(n + 1, 0);  // BIT indexada em 1
    }
    void update(int idx, int delta) {
        while (idx <= n) {
            bit[idx] += delta;
            idx += idx & -idx;
        }
    }
    int query(int idx) {
        return sum(idx);
    }
    int range_query(int l, int r) {
        return sum(r) - sum(l - 1);
    }
};

BIT fenwick(n);
for(int i = 1; i <= n; i++) {
    fenwick.update(i, arr[i]);
}
```

2.3 Psum 2d

```
// retangulo retorna a psum2d do intervalo inclusivo
vector<vector<int>> psum(n+1, vector<int>(m+1, 0));

for (int i=1; i<n+1; i++){
    for (int j=1; j<m+1; j++){
        cin >> psum[i][j];
        psum[i][j] += psum[i-1][j]+psum[i][j-1]-psum[i-1][j-1];
    }
}

// y1 eh variavel reservada
int retangulo(int x1, int yy1, int x2, int yy2){
    x2 = min(x2, n), yy2 = min(yy2, m);
    x1 = max(0LL, x1-1), yy1 = max(0LL, yy1-1);

    return psum[x2][yy2]-psum[x1][yy2]-psum[x2][yy1]+psum[x1][yy1];
}
```

2.4 Ordered Set E Map

```

#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;

template<typename T> using ordered_multiset = tree<T, null_type, less_equal<T>, rb_tree_tag, tree_order_statistics_node_update>;
template <typename T> using o_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
template <typename T, typename R> using o_map = tree<T, R, less<T>, rb_tree_tag, tree_order_statistics_node_update>;

int main() {
  int i, j, k, n, m;
  o_set<int>st;
  st.insert(1);
  st.insert(2);
  cout << *st.find_by_order(0) << endl; /// k-esimo elemento
  cout << st.order_of_key(2) << endl; ///numero de elementos menores que k
  o_map<int, int>mp;
  mp.insert({1, 10});
  mp.insert({2, 20});
  cout << mp.find_by_order(0)->second << endl; /// k-esimo elemento
  cout << mp.order_of_key(2) << endl; /// numero de elementos (chave) menores que k
  return 0;
}
```

2.5 Dsu

```
struct DSU {
    vector<int> par, rank, sz;
    int c;
    DSU(int n) : par(n + 1), rank(n + 1, 0), sz(n + 1, 1), c(n) {
        for (int i = 1; i <= n; ++i) par[i] = i;
    }
    int find(int i) {
        return (par[i] == i ? i : (par[i] = find(par[i])));
    }
    bool same(int i, int j) {
        return find(i) == find(j);
    }
    int get_size(int i) {
        return sz[find(i)];
    }
    int count() {
        return c;  // quantos componentes conexos
    }
    int merge(int i, int j) {
        if ((i = find(i)) == (j = find(j))) return -1;
        else --c;
        if (rank[i] > rank[j]) swap(i, j);
        par[i] = j;
        sz[j] += sz[i];
        if (rank[i] == rank[j]) rank[j]++;
        return j;
    }
};
```

2.6 Segtree Iterativa

```
// Exemplo de uso:
// SegTree<int> st(vetor);
// range query e point update

template <typename T>
struct SegTree {
    int n;
    vector<T> tree;
    T neutral_value = 0;
    T combine(T a, T b) {
        return a + b;
    }

    SegTree(const vector<T>& data) {
        n = data.size();
        tree.resize(2 * n, neutral_value);

        for (int i = 0; i < n; i++)
            tree[n + i] = data[i];

        for (int i = n - 1; i > 0; --i)
            tree[i] = combine(tree[i * 2], tree[i * 2 + 1]);
    }

    T range_query(int l, int r) {
        T result = neutral_value;

        for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
            if (l & 1) result = combine(result, tree[l++]);
            if (r & 1) result = combine(result, tree[--r]);
```

```cpp
        }

        return result;
    }

    void update(int pos, T new_val) {
        tree[pos += n] = new_val;

        for (pos >>= 1; pos > 0; pos >>= 1)
            tree[pos] = combine(tree[2 * pos], tree[2 * pos + 1]);
    }
};
```

2.7 Segtree Sum

```cpp
struct SegTree {
    ll merge(ll a, ll b) { return a + b; }
    const ll neutral = 0;
    int n;
    vector<ll> t, lazy;
    vector<bool> replace;
    inline int lc(int p) { return p * 2; }
    inline int rc(int p) { return p * 2 + 1; }
    void push(int p, int l, int r) {
        if (replace[p]) {
            t[p] = lazy[p] * (r - l + 1);
            if (l != r) {
                lazy[lc(p)] = lazy[p];
                lazy[rc(p)] = lazy[p];
                replace[lc(p)] = true;
                replace[rc(p)] = true;
            }
        } else if (lazy[p] != 0) {
            t[p] += lazy[p] * (r - l + 1);
            if (l != r) {
                lazy[lc(p)] += lazy[p];
                lazy[rc(p)] += lazy[p];
            }
        }
        replace[p] = false;
        lazy[p] = 0;
    }
    void build(int p, int l, int r, const vector<ll> &v) {
        if (l == r) {
            t[p] = v[l];
        } else {
            int mid = (l + r) / 2;
            build(lc(p), l, mid, v);
            build(rc(p), mid + 1, r, v);
            t[p] = merge(t[lc(p)], t[rc(p)]);
        }
    }
    void build(int _n) {
        n = _n;
        t.assign(n * 4, neutral);
        lazy.assign(n * 4, 0);
        replace.assign(n * 4, false);
    }
    void build(const vector<ll> &v) {
        n = (int)v.size();
        t.assign(n * 4, neutral);
        lazy.assign(n * 4, 0);
        replace.assign(n * 4, false);
        build(1, 0, n - 1, v);
    }
    void build(ll *bg, ll *en) {
        build(vector<ll>(bg, en));
    }
    ll query(int p, int l, int r, int L, int R) {
        push(p, l, r);
        if (l > R || r < L) return neutral;
        if (l >= L && r <= R) return t[p];
        int mid = (l + r) / 2;
        auto ql = query(lc(p), l, mid, L, R);
        auto qr = query(rc(p), mid + 1, r, L, R);
        return merge(ql, qr);
    }
    ll query(int l, int r) { return query(1, 0, n - 1, l, r); }
    void update(int p, int l, int r, int L, int R, ll val, bool repl = 0) {
        push(p, l, r);
        if (l > R || r < L) return;
        if (l >= L && r <= R) {
            lazy[p] = val;
            replace[p] = repl;
            push(p, l, r);
        } else {
            int mid = (l + r) / 2;
            update(lc(p), l, mid, L, R, val, repl);
            update(rc(p), mid + 1, r, L, R, val, repl);
            t[p] = merge(t[lc(p)], t[rc(p)]);
        }
    }
    void sumUpdate(int l, int r, ll val) { update(1, 0, n - 1, l, r, val, 0); }
    void assignUpdate(int l, int r, ll val) { update(1, 0, n - 1, l, r, val, 1); }
} segsum;
```

3 Primitives

4 Geometry

4.1 Point Location

```cpp

int32_t main(){
    sws;

    int t; cin >> t;

    while(t--){

        int x1, y1, x2, y2, x3, y3; cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3;

        int deltax1 = (x1-x2), deltay1 = (y1-y2);

        int compx = (x1-x3), compy = (y1-y3);

        int ans = (deltax1*compy) - (compx*deltay1);

        if(ans == 0){cout << "TOUCH\n"; continue;}
        if(ans < 0){cout << "RIGHT\n"; continue;}
        if(ans > 0){cout << "LEFT\n"; continue;}
    }
    return 0;
}
```

4.2 Convex Hull

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

using namespace std;
#define int long long
typedef int cod;

struct point
{
    cod x,y;
    point(cod x = 0, cod y = 0): x(x), y(y)
    {}

    double modulo()
    {
        return sqrt(x*x + y*y);
    }

    point operator+(point o)
    {
        return point(x+o.x, y+o.y);
    }
    point operator-(point o)
    {
        return point(x - o.x , y - o.y);
    }
    point operator*(cod t)
    {
        return point(x*t, y*t);
    }
    point operator/(cod t)
    {
        return point(x/t, y/t);
    }

    cod operator*(point o)
    {
        return x*o.x + y*o.y;
    }
    cod operator^(point o)
    {
        return x*o.y - y * o.x;
    }
    bool operator<(point o)
    {
        if( x != o.x) return x < o.x;
        return y < o.y;
    }

};

int ccw(point p1, point p2, point p3)
{
    cod cross = (p2-p1) ^ (p3-p1);
    if(cross == 0) return 0;
    else if(cross < 0) return -1;
    else return 1;
}

vector <point> convex_hull(vector<point> p)
{
    sort(p.begin(), p.end());
    vector<point> L,U;

    //Lower
    for(auto pp : p)
    {
        while(L.size() >= 2 and ccw(L[L.size() - 2], L.back(), pp) == -1)
        {
            // Ãľ -1 pq eu nÃčo quero excluir os colineares
            L.pop_back();
        }
        L.push_back(pp);
    }

    reverse(p.begin(), p.end());

    //Upper
    for(auto pp : p)
    {
        while(U.size() >= 2 and ccw(U[U.size()-2], U.back(), pp) == -1)
        {
            U.pop_back();
        }
        U.push_back(pp);
    }

    L.pop_back();
    L.insert(L.end(), U.begin(), U.end()-1);
    return L;
}

cod area(vector<point> v)
{
    int ans = 0;
    int aux = (int)v.size();
    for(int i = 2; i < aux; i++)
    {
        ans += ((v[i] - v[0])^(v[i-1] - v[0]))/2;
    }
    ans = abs(ans);
    return ans;
}

int bound(point p1 , point p2)
{
    return __gcd(abs(p1.x-p2.x), abs(p1.y-p2.y));
}
//teorema de pick [pontos = A - (bound+points)/2 + 1]

int32_t main()
{

    int n;
    cin >> n;

    vector<point> v(n);
    for(int i = 0; i < n; i++)
    {
        cin >> v[i].x >> v[i].y;
    }

    vector <point> ch = convex_hull(v);

    cout << ch.size() << '\n';
    for(auto p : ch) cout << p.x << " " << p.y << "\n";

    return 0;
}
```

4.3 Lattice Points

```cpp
ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a % b);
}
ll area_triangulo(ll x1, ll y1, ll x2, ll y2, ll x3, ll y3) {
    return abs(x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2));
}
ll pontos_borda(ll x1, ll y1, ll x2, ll y2) {
    return gcd(abs(x2 - x1), abs(y2 - y1));
}

int32_t main() {
    ll x1, y1, x2, y2, x3, y3;
    cin >> x1 >> y1;
    cin >> x2 >> y2;
    cin >> x3 >> y3;
    ll area = area_triangulo(x1, y1, x2, y2, x3, y3);
    ll tot_borda = pontos_borda(x1, y1, x2, y2) + pontos_borda(x2, y2, x3, y3) + pontos_borda(x3, y3, x1, y1);

    ll ans = (area - tot_borda) / 2 + 1;
    cout << ans << endl;

    return 0;
}
```

4.4 Inside Polygon

```
// Convex O(logn)

bool insideT(point a, point b, point c, point e){
    int x = ccw(a, b, e);
    int y = ccw(b, c, e);
    int z = ccw(c, a, e);
    return !((x==1 or y==1 or z==1) and (x==-1 or y==-1 or z==-1));
}

bool inside(vp &p, point e){ // ccw
    int l=2, r=(int)p.size()-1;
    while(l<r){
        int mid = (l+r)/2;
        if(ccw(p[0], p[mid], e) == 1)
            l=mid+1;
        else{
            r=mid;
        }
    }
    // bordo
    // if(r==(int)p.size()-1 and ccw(p[0], p[r], e)==0) return false;
    // if(r==2 and ccw(p[0], p[1], e)==0) return false;
    // if(ccw(p[r], p[r-1], e)==0) return false;
    return insideT(p[0], p[r-1], p[r], e);
}


// Any O(n)

int inside(vp &p, point pp){
    // 1 - inside / 0 - boundary / -1 - outside
    int n = p.size();
    for(int i=0;i<n;i++){
        int j = (i+1)%n;
        if(line({p[i], p[j]}).inside_seg(pp))
            return 0;
    }
    int inter = 0;
    for(int i=0;i<n;i++){
        int j = (i+1)%n;
        if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p[i], p[j], pp)==1)
            inter++; // up
        else if(p[j].x <= pp.x and pp.x < p[i].x and ccw(p[i], p[j], pp)==-1)
            inter++; // down
    }

    if(inter%2==0) return -1; // outside
    else return 1; // inside
}
```

5 DP

5.1 Lis

5.2 Knapsack

```
// dp[i][j] => i-esimo item com j-carga sobrando na mochila
// O(N * W)

for(int j = 0; j < MAXN; j++) {
    dp[0][j] = 0;
}
for(int i = 1; i <= N; i++) {
    for(int j = 0; j <= W; j++) {
        if(items[i].first > j) {
            dp[i][j] = dp[i-1][j];
        }
        else {
            dp[i][j] = max(dp[i-1][j], dp[i-1][j-items[i].first] + items[i].second);
        }
    }
}
```

5.3 Lcs

6 General

6.1 Struct

```
struct Pessoa{
    // Atributos
    string nome;
    int idade;

    // Comparador
    bool operator<(const Pessoa& other) const{
        if(idade != other.idade) return idade > other.idade;
        else return nome > other.nome;
    }
}
```

6.2 Bitwise

```
int check_kth_bit(int x, int k) {
  return (x >> k) & 1;
}

void print_on_bits(int x) {
  for (int k = 0; k < 32; k++) {
    if (check_kth_bit(x, k)) {
      cout << k << ' ';
    }
  }
  cout << '\n';
}

int count_on_bits(int x) {
  int ans = 0;
  for (int k = 0; k < 32; k++) {
    if (check_kth_bit(x, k)) {
      ans++;
    }
  }
  return ans;
}

bool is_even(int x) {
  return ((x & 1) == 0);
}

int set_kth_bit(int x, int k) {
  return x | (1 << k);
```

```cpp
}

int unset_kth_bit(int x, int k) {
  return x & (~(1 << k));
}

int toggle_kth_bit(int x, int k) {
  return x ^ (1 << k);
}

bool check_power_of_2(int x) {
  return count_on_bits(x) == 1;
}
```

7 Graph

7.1 Bellman Ford

```cpp
struct Edge {
    int u, v, w;
};

// se x = -1, nÃčo tem ciclo
// se x != -1, pegar pais de x pra formar o ciclo

int n, m;
vector<Edge> edges;
vector<int> dist(n);
vector<int> pai(n, -1);

    for (int i = 0; i < n; i++) {
        x = -1;
        for (Edge &e : edges) {
            if (dist[e.u] + e.w < dist[e.v]) {
                dist[e.v] = max(-INF, dist[e.u] + e.w);
                pai[e.v] = e.u;
                x = e.v;
            }
        }
    }

// achando caminho (se precisar)
for (int i = 0; i < n; i++) x = pai[x];

vector<int> ciclo;
for (int v = x;; v = pai[v]) {
    cycle.push_back(v);
    if (v == x && ciclo.size() > 1) break;
}
reverse(ciclo.begin(), ciclo.end());
```

7.2 Lca

```cpp
// LCA - CP algorithm
// preprocessing O(NlogN)
// lca O(logN)
// Uso: criar LCA com a quantidade de vÃĺrtices (n) e lista de adjacÃłncia (adj)
// chamar a funÃğÃčo preprocess com a raiz da Ãąrvore

struct LCA {
    int n, l, timer;
    vector<vector<int>> adj;
    vector<int> tin, tout;
    vector<vector<int>> up;

    LCA(int n, const vector<vector<int>>& adj) : n(n), adj(adj) {}

    void dfs(int v, int p) {
        tin[v] = ++timer;
        up[v][0] = p;
        for (int i = 1; i <= l; ++i)
            up[v][i] = up[up[v][i-1]][i-1];

        for (int u : adj[v]) {
            if (u != p)
                dfs(u, v);
        }

        tout[v] = ++timer;
    }

    bool is_ancestor(int u, int v) {
        return tin[u] <= tin[v] && tout[u] >= tout[v];
    }

    int lca(int u, int v) {
        if (is_ancestor(u, v))
            return u;
        if (is_ancestor(v, u))
            return v;
        for (int i = l; i >= 0; --i) {
            if (!is_ancestor(up[u][i], v))
                u = up[u][i];
        }
        return up[u][0];
    }

    void preprocess(int root) {
        tin.resize(n);
        tout.resize(n);
        timer = 0;
        l = ceil(log2(n));
        up.assign(n, vector<int>(l + 1));
        dfs(root, root);
    }
};
```

7.3 Dfs

```cpp
int dfs(int x, int p) {
    for (auto e : adj[x]) {
        if (e != p) {
            dfs(e, x);
        }
    }
}
```

7.4 Topological Sort

```cpp
vector<int> adj[MAXN];
vector<int> estado(MAXN); // 0: nao visitado 1: processamento 2: processado
vector<int> ordem;
bool temCiclo = false;

void dfs(int v) {
    if(estado[v] == 1) {
        temCiclo = true;
        return;
    }
    if(estado[v] == 2) return;
    estado[v] = 1;
    for(auto &nei : adj[v]) {
        if(estado[v] != 2) dfs(nei);
    }
    estado[v] = 2;
    ordem.push_back(v);
    return;
```

7.5 Dijkstra

```
// SSP com pesos positivos.
// O((V + E) log V).

vector<int> dijkstra(int S) {
    vector<bool> vis(MAXN, 0);
    vector<ll> dist(MAXN, LLONG_MAX);
    dist[S] = 0;
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    pq.push({0, S});
    while(pq.size()) {
        ll v = pq.top().second;
        pq.pop();
        if(vis[v]) continue;
        vis[v] = 1;
        for(auto &[peso, vizinho] : adj[v]) {
            if(dist[vizinho] > dist[v] + peso) {
                dist[vizinho] = dist[v] + peso;
                pq.push({dist[vizinho], vizinho});
            }
        }
    }
    return dist;
}
```

7.6 Lca Jc

```
int LOG;

int get_lca(int a, int b) {
    if(profundidade[b] > profundidade[a]) {
        swap(a, b);
    }
    int k = profundidade[a] - profundidade[b]; // tanto que tenho que subir
    for(int j = LOG-1; j >= 0; j--) {
        if((1 << j) & k) {
            a = cima[a][j];
        }
    }
    if(a == b) return a; // ja to no lca

    for(int j = LOG-1; j >= 0; j--) { // subo com os dois atÃľ chegar no lca fazendo binary lifting
        if(cima[a][j] != cima[b][j]) {
            a = cima[a][j];
            b = cima[b][j];
        }
    }
    return cima[a][0];
}

void dfs(int v, int p) {
    if(v != 1) profundidade[v] = profundidade[p] + 1;
    cima[v][0] = p;
    for(int j = 1; j < LOG; j++) {
        if (cima[v][j-1] != -1) {
            cima[v][j] = cima[cima[v][j-1]][j-1];
        } else {
            cima[v][j] = -1;
        }
    }
    for(auto &nei : adj[v]) {
        if(nei != p) {
            dfs(nei, v);
        }
    }
}

while((1 << LOG) <= n) LOG++;
```

7.7 Kruskal

```
// Ordena as arestas por peso, insere se ja nao estiver no mesmo componente
// O(E log E)

struct DSU {
    vector<int> par, rank, sz;
    int c;
    DSU(int n) : par(n + 1), rank(n + 1, 0), sz(n + 1, 1), c(n) {
        for (int i = 1; i <= n; ++i) par[i] = i;
    }
    int find(int i) {
        return (par[i] == i ? i : (par[i] = find(par[i])));
    }
    bool same(int i, int j) {
        return find(i) == find(j);
    }
    int get_size(int i) {
        return sz[find(i)];
    }
    int count() {
        return c;  // quantos componentes conexos
    }
    int merge(int i, int j) {
        if ((i = find(i)) == (j = find(j))) return -1;
        else --c;
        if (rank[i] > rank[j]) swap(i, j);
        par[i] = j;
        sz[j] += sz[i];
        if (rank[i] == rank[j]) rank[j]++;
        return j;
    }
};

struct Edge {
    int u, v, w;
    bool operator <(Edge const & other) {
        return weight <other.weight;
    }
}

vector<Edge> kruskal(int n, vector<Edge> edges) {
    vector<Edge> mst;
    DSU dsu = DSU(n + 1);
    sort(edges.begin(), edges.end());
    for (Edge e : edges) {
        if (dsu.find(e.u) != dsu.find(e.v)) {
            mst.push_back(e);
            dsu.join(e.u, e.v);
        }
    }
    return mst;
}
```

7.8 Floyd Warshall

```
// SSP e acha ciclos.
// Bom com constraints menores.
// O(n^3)

int dist[501][501];

void floydWarshall() {
    for(int k = 0; k < n; k++) {
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++) {
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }
}
void solve() {
    int m, q;
    cin >> n >> m >> q;
    for(int i = 0; i < n; i++) {
        for(int j = i; j < n; j++) {
            if(i == j) {
                dist[i][j] = dist[j][i] = 0;
            } else {
                dist[i][j] = dist[j][i] = linf;
            }
        }
    }
    for(int i = 0; i < m; i++) {
        int u, v, w;
        cin >> u >> v >> w; u--; v--;
        dist[u][v] = min(dist[u][v], w);
        dist[v][u] = min(dist[v][u], w);
    }
    floydWarshall();
    while(q--) {
        int u, v;
        cin >> u >> v; u--; v--;
        if(dist[u][v] == linf) cout << -1 << '\n';
        else cout << dist[u][v] << '\n';
    }
}
```

8 Search and sort

8.1 Mergeandcount

```

// Realiza a mesclagem de dois subarrays e conta o nÃžmero de trocas necessÃąrias.
int mergeAndCount(vector<int>& v, int l, int m, int r) {
    int x = m - l + 1;  // Tamanho do subarray esquerdo.
    int y = r - m;  // Tamanho do subarray direito.

    // Vetores temporarios para os subarray esquerdo e direito.
    vector<int> left(x), right(y);

    for (int i = 0; i < x; i++) left[i] = v[l + i];
    for (int j = 0; j < y; j++) right[j] = v[m + 1 + j];

    int i = 0, j = 0, k = l;
    int swaps = 0;

    while (i < x && j < y) {
        if (left[i] <= right[j]) {
            // Se o elemento da esquerda for menor ou igual, coloca no vetor original.
            v[k++] = left[i++];
        } else {
            // Caso contrario, coloca o elemento da direita e conta as trocas.
            v[k++] = right[j++];
            swaps += (x - i);
        }
    }

    // Adiciona os elementos restantes do subarray esquerdo (se houver).
    while (i < x) v[k++] = left[i++];

    // Adiciona os elementos restantes do subarray direito (se houver).
    while (j < y) v[k++] = right[j++];

    return swaps;  // Retorna o numero total de trocas realizadas.
}

int mergeSort(vector<int>& v, int l, int r) {
    int swaps = 0;

    if (l < r) {
        // Encontra o ponto medio para dividir o vetor.
        int m = l + (r - l) / 2;

        // Chama merge sort para a metade esquerda.
        swaps += mergeSort(v, l, m);
        // Chama merge sort para a metade direita.
        swaps += mergeSort(v, m + 1, r);

        // Mescla as duas metades e conta as trocas.
        swaps += mergeAndCount(v, l, m, r);
    }

    return swaps;  // Retorna o numero total de trocas no vetor.
}
```

8.2 Bfs

```
// Printa os nos na ordem em que sÃčo visitados
// Explora em largura (camadas)
// Complexidade: O(V+A) V = vertices e A = arestas
// Espaco: O(V)
// Uso: busca pelo caminho mais curto

void bfs(vector<vector<int>>&grafo, int inicio){
    set<int> visited;
    queue<int> fila;

    fila.push(inicio);
    visited.insert(inicio);

    while(!fila.empty()){
        int cur = fila.front();
        fila.pop();

        cout << cur << " "; // printa o nÃş atual

        for(int vizinho: grafo[cur]){
            if(visited.find(vizinho) == visited.end()){
                fila.push(vizinho);
                visited.insert(vizinho)
            }
        }
    }
}
```

8.3 Dfs

```
// Printa os nos na ordem em que sÃčo visitados
// Explora em profundidade
// Complexidade: O(V+A) V = vertices e A = arestas
// Espaco: O(V)
// Uso: explorar caminhos e backtracking

void dfs(vector<vector<int>>& grafo, int inicio){
    set<int> visited;
    stack<int> pilha;

    pilha.push(inicio);

    while(!pilha.empty()){
        int cur = pilha.top();
        pilha.pop();

        if(visited.find(cur) == visited.end()){
            cout << cur << " ";
            visited.insert(cur);

            for(int vizinho: grafo[cur]){
                if(visited.find(vizinho) == visited.
    end()){
                    pilha.push(vizinho);
                }
            }
        }
    }
}
```

9 Math

9.1 Equacao Diofantina

```
// resolve equacao ax + by = c
// retorno {existe sol., x, y, g}
array<ll, 4> find_any_solution(ll a, ll b, ll c) {
    auto[x, y, g] = exgcd(a, b);
    if (c % g) return {false, 0, 0, 0};
    x *= c / g;
    y *= c / g;
    return {true, x, y, g};
}
```

9.2 Crivo

```
// O(n*log(log(n)))
bool composto[MAX]
for(int i = 1; i <= n; i++) {
    if(composto[i]) continue;
    for(int j = 2*i; j <= n; j += i)
        composto[j] = 1;
}
```

9.3 Fexp

```
// a^e mod m
// O(log n)

int fexp(int a, int e, int m) {
    a %= m;
    int ans = 1;
    while (e > 0){
        if (e & 1) ans = ans*a % m;
        a = a*a % m;
        e /= 2;
    }
    return ans%m;
}
```

9.4 Exgcd

```
// O retorno da funcao eh {n, m, g}
// e significa que gcd(a, b) = g e
// n e m sao inteiros tais que an + bm = g
array<ll, 3> exgcd(int a, int b) {
    if(b == 0) return {1, 0, a};
    auto [m, n, g] = exgcd(b, a % b);
    return {n, m - a / b * n, g};
}
```

9.5 Mod Inverse

```
array<int, 2> extended_gcd(int a, int b) {
    if (b == 0) return {1, 0};
    auto [x, y] = extended_gcd(b, a % b);
    return {y, x - (a / b) * y};
}

int mod_inverse(int a, int m) {
    auto [x, y] = extended_gcd(a, m);
    return (x % m + m) % m;
}
```