Competitive Programming Notebook

Programadores Roblox

Contents

1 DP

1.1 Lis

1.2 Lcs

1.3 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);
        }
    }
}
```

2 String

3 Geometry

3.1 Point Location

```

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;
}
```

3.2 Convex Hull

```
#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)
        {
            // Ãŕŕ -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;
}
```

3.3 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;
        }
    }
    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
}
```

4 Graph

4.1 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;
}
```

4.2 Topological Sort

```
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;
```

4.3 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';
    }
}
```

4.4 Bellman Ford

```
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());
```

4.5 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;
}
```

5 Math

5.1 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;
}
```

5.2 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};
}
```

5.3 Fexp

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

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

5.4 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};
}
```

6 DS

6.1 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;
}
```

6.2 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;
    }
};
```

7 Primitives

8 General

8.1 Bitwise

```cpp
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);
}

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;
}
```