Notebook

UTN FRRO - HOLA

2024

Contents

1 Basics

1.1 Template

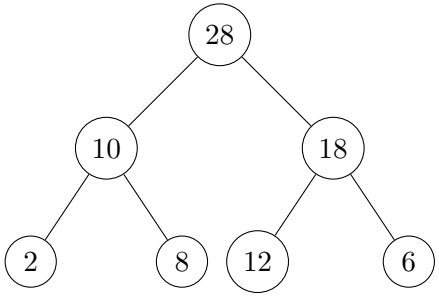
```cpp
#include <bits/stdc++.h>
#define forr(i, a, b) for (int i = (a); i < (b); i++)
#define forn(i, n) forr(i, 0, n)
#define dforn(i, n) for (int i = (n) - 1; i >= 0; i--)
#define forall(it, v) for (auto it = v.begin(); it != v.end(); it++)

#ifdef EBUG
// local
#else
// judge
#endif

using namespace std;

int main() {
#ifdef EBUG
    freopen("input.txt", "r", stdin);
#endif

    ios::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    return 0;
}
```

1.2 Compilation

```
g++ -DEBUG <ej>.cpp -o a && time ./a
```

2 Estructuras

2.1 Segment Tree




```cpp
#include <bits/stdc++.h>
#define INF 100000000
using namespace std;

int n, t[4*10000];

void buildST(int a[], int v, int tl, int tr) { // este te hace las sumas
    if (tl == tr) {
        t[v] = a[tl];
    } else {
        int tm = (tl + tr) / 2;
        buildST(a, v * 2, tl, tm);
        buildST(a, v * 2 + 1, tm + 1, tr);
        t[v] = t[v * 2] + t[v * 2 + 1]; //aca esta la parte en que las suma
    }
}

int queryST(int v, int tl, int tr, int l, int r){
    if( l > r){
        return 0;
    }
    if( l == tl && r == tr){
        return t[v];
    }
    int tm = (tl + tr)/2;
    return queryST(v*2, tl, tm, l, min(r, tm)) +
    + queryST(v*2+1, tm+1, tr, max(l, tm+1), r); // esto lo hace para sumar

}

void updateST(int v, int tl, int tr, int pos, int new_val){
    if(tl == tr){
        t[v] = new_val;
    } else {
        int tm = (tl + tr) / 2;
        if(pos <= tm){
            updateST(v*2, tl, tm, pos, new_val);
        }else{
            updateST(v*2 + 1, tm+1, tr, pos, new_val);
        }
        t[v] = t[v*2] + t[v*2+1];
    }
}
```

3 Grafos

3.1 Recorrer Grafos

Dado un Grafo como lista de adjacencias

```cpp
#include <bits/stdc++.h>
#define MAXN 100000
using namespace std;

vector<int> G[MAXN];
bool visited[MAXN];
```

Podemos recorrerlo con DFS o con BFS.

3.1.1 DFS

```cpp
void dfs(int nodo){
    if(visited[nodo]) return;
    visited[nodo] = true;

    for(auto it : G[nodo]){
        if(!visited[it]){
            // logic
        }
    }
}
```

3.1.2 BFS

```cpp
void bfs(int nodo){
    cola.push(nodo);

    while(!cola.empty()){
        Nodo actual = cola.front();
        cola.pop();

        for (int vecino : G[actual]) {
            if (!visited[vecino]) {
                visited[vecino] = true;
                cola.push(vecino);
            }
        }
    }
}
```

3.2 Camino Mínimo

3.2.1 Bellman-Ford

El algoritmo de Bellman-Ford encuentra el camino desde un nodo de origen a todos los nodos del grafo.
Complejidad = O(nm)

```cpp
#include <bits/stdc++.h>
#define INF 100000000
using namespace std;

vector<tuple<int, int, int>> edges;
int n;

void bellman_ford(int x){
    vector<int> distance(n, INF);
    distance[x] = 0;

    for (int i = 1; i <= n-1; i++) {
        for (auto e : edges) {
            int a, b, w;
            tie(a, b, w) = e;
            distance[b] = min(distance[b], distance[a]+w);
        }
    }
}
```

3.2.2 Ciclos negativos

El algoritmo es capaz de detectar ciclos negativos. Para eso Se debe correr una vez m'as

3.2.3 Dijkstra

El algorimo necesita que todos los pesos sean ¿ 0
Complejidad = O(n + m log m)

```cpp
#include <bits/stdc++.h>
#define MAXN 100000
using namespace std;
typedef pair<int, int> ii;

vector<pair<int, int>> G[MAXN]; //Lista de pares, dest, peso
bool visited[MAXN];
int n;

void dijkstra(int x){
    // La PQ esta ordenada de menor a mayor
    priority_queue<ii, vector<ii>, greater<ii>> q;
    vector<int> distance(n, MAXN);
    distance[x] = 0;
    q.push({0,x});

    while (!q.empty()) {
        int a = q.top().second; q.pop();
        if (visited[a]) continue;
        visited[a] = true;

        for (auto u : G[a]) {
            int b = u.first, w = u.second;
            if (distance[a]+w < distance[b]) {
                distance[b] = distance[a]+w;
                q.push({distance[b], b});
            }
        }
    }
}
```

4 Geometria

4.1 Punto

```cpp
#include <bits/stdc++.h>
#define ll long long
#define ld double

struct pto {
    ll x, y;
    pto() : x(0), y(0) {} //Constructor pto a = pto(); ==> ) a.x = 0, a.y = 0
    pto(ll _x, ll _y) : x(_x), y(_y) {}
    pto operator+(pto b) { return pto(x+b.x, y+b.y); }
    pto operator-(pto b) { return pto(x-b.x, y-b.y); }
    pto operator+(ll k) { return pto(x+k, y+k); }
    pto operator*(ll k) { return pto(x*k, y*k); }
    pto operator/(ll k) { return pto(x/k, y/k); }
    ll operator*(pto b) { return x*b.x + y*b.y; }
    pto proj(pto b) { return b*((*this)*b) / (b*b); }
    ll operator^(pto b) { return x*b.y - y*b.x; }
    ld norm() { return sqrt(x*x + y*y); }
    ld dist(pto b) { return (b - (*this)).norm(); }
};
```

4.2 Line

```cpp
#include "pto.cpp"

int sgn(T x) { return x < 0 ? -1 : !!x; }

struct line {
    T a, b, c;  // Ax+By=C
    line() {}
    line(T a_, T b_, T c_) : a(a_), b(b_), c(c_) {}

    // TO DO: check negative C (multiply everything by -1)
    line(pto u, pto v) : a(v.y - u.y), b(u.x - v.x), c(a * u.x + b * u.y) {}
    int side(pto v) { return sgn(a * v.x + b * v.y - c); }
    bool inside(pto v) { return abs(a * v.x + b * v.y - c) <= EPS; }
    bool parallel(line v) { return abs(a * v.b - v.a * b) <= EPS; }
    pto inter(line v) {
        T det = a * v.b - v.a * b;
        if (abs(det) <= EPS) return pto(INF, INF);
        return pto(v.b * c - b * v.c, a * v.c - v.a * c) / det;
    }
};
```

4.3 Segment

```cpp
#include "pto.cpp"
#include "line.cpp"

struct segment {
    pto s, e;
    segment(pto s_, pto e_) : s(s_), e(e_) {}

    pto closest(pto b) {
        pto bs = b - s, es = e - s;
        ld l = es * es;
        if (abs(l) <= EPS) return s;
        ld t = (bs * es) / l;
        if (t < 0.) return s;       // comment for lines
        else if (t > 1.) return e;  // comment for lines
        return s + (es * t);
    }
    bool inside(pto b) { //Return true if pto b is inside the segment
        return abs(s.dist(b) + e.dist(b) - s.dist(e)) < EPS;
    }

    pto inter(segment b) {  // if a and b are collinear, returns one point
        if ((*this).inside(b.s)) return b.s;
        if ((*this).inside(b.e)) return b.e;
        pto in = line(s, e).inter(line(b.s, b.e));
        if ((*this).inside(in) && b.inside(in)) return in;
        return pto(INF, INF);
    }
};
```

4.4 Circle

```cpp
#define sqr(a) ((a)*(a))
pto perp(pto a){return pto(-a.y, a.x);}
line bisector(pto a, pto b){
    line l = line(a, b); pto m = (a+b)/2;
    return line(-l.b, l.a, -l.b*m.x+l.a*m.y);
}

struct circle{
    pto o; T r;
    circle(){}
    circle(pto a, pto b, pto c) {
        o = bisector(a, b).inter(bisector(b, c));
        r = o.dist(a);
    }
    bool inside(pto p) { return (o-p).norm_sq() <= r*r+EPS; }
    bool inside(circle c) { // this inside of c
        T d = (o - c.o).norm_sq();
        return d <= (c.r-r) * (c.r-r) + EPS;
    }
    // circle containing p1 and p2 with radius r
    // swap p1, p2 to get snd solution
    circle* circle2PtoR(pto a, pto b, T r_) {
        ld d2 = (a-b).norm_sq(), det = r_*r_/d2 - ld(0.25);
        if(det < 0) return nullptr;
        circle *ret = new circle();
        ret->o = (a+b)/ld(2) + perp(b-a)*sqrt(det);
        ret->r = r_;
        return ret;
    }
    pair<pto, pto> tang(pto p) {
        pto m = (p+o)/2;
        ld d = o.dist(m);
        ld a = r*r/(2*d);
        ld h = sqrtl(r*r - a*a);
        pto m2 = o + (m-o)*a/d;
        pto per = perp(m-o)/d;
        return make_pair(m2 - per*h, m2 + per*h);
    }
    vector<pto> inter(line l) {
        ld a = l.a, b = l.b, c = l.c - l.a*o.x - l.b*o.y;
        pto xy0 = pto(a*c/(a*a + b*b), b*c/(a*a + b*b));
        if(c*c > r*r*(a*a + b*b) + EPS) {
            return {};
        }else if(abs(c*c - r*r*(a*a + b*b)) < EPS) {
            return { xy0 + o };
        }else{
            ld m = sqrtl((r*r - c*c/(a*a + b*b))/(a*a + b*b));
            pto p1 = xy0 + (pto(-b,a)*m);
            pto p2 = xy0 + (pto(b,-a)*m);
            return { p1 + o, p2 + o };
        }
    }
    vector<pto> inter(circle c) {
        line l;
        l.a = o.x - c.o.x;
        l.b = o.y - c.o.y;
        l.c = (sqr(c.r)-sqr(r)+sqr(o.x)-sqr(c.o.x)+sqr(o.y)-sqr(c.o.y))/2.0;
        return (*this).inter(l);
    }
    ld inter_triangle(pto a, pto b) { // area of intersection with oab
        if(abs((o-a)^(o-b)) <= EPS) return 0.;
        vector<pto> q = {a}, w = inter(line(a,b));
        if(sz(w) == 2) forn(i,sz(w)) if((a-w[i])*(b-w[i]) < -EPS) q.pb(w[i]);
        q.pb(b);
        if(sz(q) == 4 && (q[0]-q[1])*(q[2]-q[1]) > EPS) swap(q[1], q[2]);
        ld s = 0;
        forn(i, sz(q)-1){
            if(!inside(q[i]) || !inside(q[i+1])) {
                s += r*r*angle((q[i]-o),q[i+1]-o)/T(2);
            }
            else s += abs((q[i]-o)^(q[i+1]-o)/2);
        }
        return s;
    }
};
vector<ld> inter_circles(vector<circle> c){
    vector<ld> r(sz(c)+1); // r[k]: area covered by at least k circles
    forn(i, sz(c)) {      // O(n^2 log n) (high constant)
```

```
        int k = 1;
        cmp s(c[i].o, pto(1,0));
        vector<pair<pto,int>> p = {
            {c[i].o + pto(1,0)*c[i].r, 0},
            {c[i].o - pto(1,0)*c[i].r, 0}};
        forn(j, sz(c)) if(j != i) {
            bool b0 = c[i].inside(c[j]), b1 = c[j].inside(c[i]);
            if(b0 && (!b1 || i<j)) k++;
            else if(!b0 && !b1) {
                vector<pto> v = c[i].inter(c[j]);
                if(sz(v) == 2) {
                    p.pb({v[0], 1}); p.pb({v[1], -1});
                    if(s(v[1], v[0])) k++;
                }
            }
        }
        sort(p.begin(), p.end(), [&](pair<pto,int> a, pair<pto,int> b) {
                return s(a.fst,b.fst); });
        forn(j,sz(p)) {
            pto p0 = p[j? j-1: sz(p)-1].fst, p1 = p[j].fst;
            ld a = angle(p0 - c[i].o, p1 - c[i].o);
            r[k]+=(p0.x-p1.x)*(p0.y+p1.y)/ld(2)+c[i].r*c[i].r*(a-sinl(a))/ld
                (2);
            k += p[j].snd;
        }
    }
    return r;
}
```

5 Utils

5.1 Binary Search

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

int bs(vector<int> &v, int val){
    int l = 0, r = v.size() - 1, mid = (l+r)/2;
    while(l <= r){
        if(val < v[mid]){
            r = mid - 1;
        }else{
            l = mid + 1;
        }
        mid = (l+r)/2;
    }
    if(val < v[mid]){
        mid --;
    }

    return mid;
}
```

5.2 Sort

Ordenar un vector de pair por su segunda componente

```
vector<pair<int, int>> v;

bool sortbysec(const pair<int,int> &a, const pair<int,int> &b){
  return (a.second < b.second);
}

sort(v.begin(), v.end(), sortbysec);
```