Ayudamemoria

My room is random Sorted

October 11, 2024

Contents

1 Template

```cpp
#include <bits/stdc++.h>
using namespace std;
#define dprint(v) cout << #v "=" << v << endl
#define forr(i, a, b) for (int i = (a); i < (b); i++)
#define forn(i, n) forr(i, 0, n)
#define dforr(i, a, b) for (int i = (b - 1); i >= (a); i--)
#define dforn(i, n) dforr(i,0,n)
#define forall(it, v) for (auto it = v.begin(); it != v.end();
    ++it)
#define sz(c) ((int)c.size())
#define pb push_back
#define fst first
#define snd second
#define mp make_pair
#define all(v) begin(v),end(v)
#define endl '\n'
typedef long long ll;
typedef pair<int, int> pii;

int main(int argc, char **argv){
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    if(argc == 2) freopen(argv[1], "r", stdin);
}
```

1.1 run.sh

```
clear
make -s $1 && ./$1 $2
```

1.2 comp.sh

```
clear
make -s $1 2>&1 | head -$2
```

1.3 Makefile

```
CC = g++
CXXFLAGS = -Wall -g \
-fsanitize=undefined -fsanitize=bounds \
-std=c++2a -O2
```

2 Estructuras de datos

2.1 Sparse Table

```
#define oper min
int st[K][1<<K];int n; // K such that 2^K>n
void st_init(vector<int> a){
    forn(i,n)st[0][i]=a[i];
    forr(k,1,K)forn(i,n-(1<<k)+1)
        st[k][i]=oper(st[k-1][i],st[k-1][i+(1<<(k-1))]);
}
int st_query(int s, int e){
    int k=31-__builtin_clz(e-s);
    return oper(st[k][s],st[k][e-(1<<k)]);
}
```

2.2 Segment Tree

```
//Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera sobre el rango [i, j).
#define MAXN 100000
#define oper(x, y) max(x, y)
const int neutro=0;
struct RMQ{
    int sz;
    tipo t[4*MAXN];
    tipo &operator[](int p){return t[sz+p];}
    void init(int n){//O(nlgn)
        sz = 1 << (32-__builtin_clz(n));
        forn(i, 2*sz) t[i]=neutro;
    }
    void updall(){//O(n)
        dforn(i, sz) t[i]=oper(t[2*i], t[2*i+1]);}
    tipo get(int i, int j){return get(i,j,1,0,sz);}
    tipo get(int i, int j, int n, int a, int b){//O(lgn)
        if(j<=a || i>=b) return neutro;
        if(i<=a && b<=j) return t[n];
        int c=(a+b)/2;
        return oper(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
    }
    void set(int p, tipo val){//O(lgn)
        for(p+=sz; p>0 && t[p]!=val;){
            t[p]=val;
            p/=2;
            val=oper(t[p*2], t[p*2+1]);
        }
    }
}rmq;
//Usage:
cin >> n; rmq.init(n); forn(i, n) cin >> rmq[i]; rmq.updall();
```

2.3 Segment Tree Lazy

```
//Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera sobre el rango [i, j).
typedef int Elem;//Elem de los elementos del arreglo
typedef int Alt;//Elem de la alteracion
#define operacion(x,y) x+y
const Elem neutro=0; const Alt neutro2=0;
#define MAXN 100000
struct RMQ{
    int sz;
    Elem t[4*MAXN];
    Alt dirty[4*MAXN];//las alteraciones pueden ser de distinto Elem
    Elem &operator[](int p){return t[sz+p];}
    void init(int n){//O(nlgn)
        sz = 1 << (32-__builtin_clz(n));
        forn(i, 2*sz) t[i]=neutro;
        forn(i, 2*sz) dirty[i]=neutro2;
    }
    void push(int n, int a, int b){//propaga el dirty a sus hijos
        if(dirty[n]!=0){
            t[n]+=dirty[n]*(b-a);//altera el nodo
            if(n<sz){
                dirty[2*n]+=dirty[n];
                dirty[2*n+1]+=dirty[n];
            }
            dirty[n]=0;
        }
    }
    Elem get(int i, int j, int n, int a, int b){//O(lgn)
        if(j<=a || i>=b) return neutro;
        push(n, a, b);//corrige el valor antes de usarlo
        if(i<=a && b<=j) return t[n];
        int c=(a+b)/2;
        return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
    }
    Elem get(int i, int j){return get(i,j,1,0,sz);}
    //altera los valores en [i, j) con una alteracion de val
    void alterar(Alt val, int i, int j, int n, int a, int b){//O(lgn)
        push(n, a, b);
        if(j<=a || i>=b) return;
        if(i<=a && b<=j){
            dirty[n]+=val;
            push(n, a, b);
            return;
        }
        int c=(a+b)/2;
        alterar(val, i, j, 2*n, a, c), alterar(val, i, j, 2*n+1, c, b);
        t[n]=operacion(t[2*n], t[2*n+1]);//por esto es el push de arriba
    }
    void alterar(Alt val, int i, int j){alterar(val,i,j,1,0,sz);}
}rmq;
```

2.4 Fenwick Tree

```
struct Fenwick{
    static const int sz=1<<K;
    ll t[sz]={};
    void adjust(int p, ll v){
        for(int i=p+1;i<sz;i+=(i&-i)) t[i]+=v;
    }
    ll sum(int p){ // suma [0,p)
        ll s = 0;
        for(int i=p;i;i-=(i&-i)) s+=t[i];
        return s;
    }
    ll sum(int a, int b){return sum(b)-sum(a);} // suma [a,b)

    //funciona solo con valores no negativos en el fenwick
    //longitud del minimo prefijo t.q. suma <= x
    //para el maximo v+1 y restar 1 al resultado
    int pref(ll v){
        int x = 0;
        for(int d = 1<<(K-1); d; d>>=1){
            if( t[x|d] < v ){
                x |= d;
                v -= t[x];
            }
        }
        return x+1;
    }
};

struct RangeFT { // 0-indexed, query [0, i), update [l, r)
    Fenwick rate, err;
    void adjust(int l, int r, int x) { // range update
        rate.adjust(l, x); rate.adjust(r, -x); err.adjust(l, -x*l); err.adjust(r, x*r); }
    ll sum(int i) { return rate.sum(i) * i + err.sum(i); } }; // prefix query


struct Fenwick2D{
    ll t[N][M]={};
    void adjust(int p, int q, ll v){
        for(int i=p+1;i<N;i+=(i&-i))
            for(int j= q+1; j<M; j+=(j&-j))
                t[i][j]+=v;
    }
    ll sum(int p,int q){ // suma [0,p)
        ll s = 0;
        for(int i=p;i;i-=(i&-i))
            for(int j=q; j; j-=(j&-j))
                s+=t[i][j];
        return s;
    }
    ll sum(int x1, int y1, int x2, int y2){return sum(x2,y2)-sum(x1,y2)-sum(x2,y1)+sum(x1,y1);} // suma [a,b)
};
```

2.5 Tabla Aditiva

```
// Tablita aditiva 2D
forn (dim, 2) {
    forn (i, N) {
        forn (j, M) {
            int pi = i-(dim==0), pj = j-(dim==1);
            if (pi >= 0 && pj >= 0) {
                dp[i][j] += dp[pi][pj];
            }
        }
    }
}
// Generalizacion a 32 dimensiones para mascaras de bits
forn (i, 32) {
    forn (mask, 1<<32) {
        if ((mask>>i)&1) {
            dp[mask] += dp[mask - (1<<i)];
        }
    }
}
```

2.6 Union Find

```
vector<int> uf(MAXN, -1);
int uf_find(int x) { return uf[x]<0 ? x : uf[x] = uf_find(uf[x]); }
bool uf_join(int x, int y){ // True sii x e y estan en != componentes
    x = uf_find(x); y = uf_find(y);
    if(x == y) return false;
    if(uf[x] > uf[y]) swap(x, y);
    uf[x] += uf[y]; uf[y] = x; return true;
}
```

3 Matemática

3.1 Criba Lineal

```
const int N = 10000000;
vector<int> lp(N+1);
vector<int> pr;

for (int i=2; i <= N; ++i) {
    if (lp[i] == 0) {
        lp[i] = i;
        pr.push_back(i);
    }
    for (int j = 0; i * pr[j] <= N; ++j) {
        lp[i * pr[j]] = pr[j];
        if (pr[j] == lp[i]) {
            break;
        }
    }
}
```

3.2 Phollard's Rho

```
ll gcd(ll a, ll b){return a?gcd(b %a, a):b;}


ll mulmod(ll a, ll b, ll m) {
 return ll(__int128(a) * b % m);
}

ll expmod (ll b, ll e, ll m){//O(log b)
    if(!e) return 1;
    ll q= expmod(b,e/2,m); q=mulmod(q,q,m);
    return e%2? mulmod(b,q,m) : q;
}

```

```
bool es_primo_prob (ll n, int a)
{
    if (n == a) return true;
    ll s = 0,d = n-1;
    while (d % 2 == 0) s++,d/=2;

    ll x = expmod(a,d,n);
    if ((x == 1) || (x+1 == n)) return true;

    forn (i, s-1){
        x = mulmod(x, x, n);
        if (x == 1) return false;
        if (x+1 == n) return true;
    }
    return false;
}

bool rabin (ll n){ //devuelve true si n es primo
    if (n == 1) return false;
    const int ar[] = {2,3,5,7,11,13,17,19,23};
    forn (j,9)
        if (!es_primo_prob(n,ar[j]))
            return false;
    return true;
}

ll rho(ll n){
    if( (n & 1) == 0 ) return 2;
    ll x = 2 , y = 2 , d = 1;
    ll c = rand() % n + 1;
    while( d == 1 ){
        x = (mulmod( x , x , n ) + c)%n;
        y = (mulmod( y , y , n ) + c)%n;
        y = (mulmod( y , y , n ) + c)%n;
        if( x - y >= 0 ) d = gcd( x - y , n );
        else d = gcd( y - x , n );
    }
    return d==n? rho(n):d;
}

map<ll,ll> prim;
void factRho (ll n){ //O (lg n)^3. un solo numero
    if (n == 1) return;
    if (rabin(n)){
        prim[n]++;
        return;
    }
    ll factor = rho(n);
    factRho(factor);
    factRho(n/factor);
}
```

4 Geometria

```
struct Point
{
    double x, y;
    double Point::operator*(const Point &o) const {
        return x * o.x + y * o.y; }
    double Point::operator^(const Point &o) const {
        return x * o.y - y * o.x; }
    Point Point::operator-(const Point &o) const {
        return {x - o.x, y - o.y}; }
    Point Point::operator+(const Point &o) const {
        return {x + o.x, y + o.y}; }
    Point Point::operator*(const double &u) const {
        return {x * u, y * u}; }
    Point Point::operator/(const double &u) const {
        return {x / u, y / u}; }
    double Point::norm_sq() const {
        return x * x + y * y; }
    double Point::norm() const {
        return sqrt(x * x + y * y); }
};

struct Comp {
    Vector o, v;
    Comp(Vector _o, Vector _v) : o(_o), v(_v) {}
    bool half(Vector p) {
        assert(!(p.x == 0 && p.y == 0));
        return (v ^ p) < 0 || ((v ^ p) == 0 && (v * p) < 0);
    }
    bool operator()(Vector a, Vector b) {
        return mp(half(a - o), 0ll) < mp(half(b - o), ((a - o) ^ (b - o)));
    }
};

struct Segment {
    Vector a, b;
    long double eval() const
    { // funcion auxiliar para ordenar segmentos
        assert(a.x != b.x || a.y != b.y);
        Vector a1 = a, b1 = b;
        if (a1.x > b1.x)
            swap(a1, b1);
        assert(x >= a1.x && x <= b1.x);
        if (x == a1.x)
            return a1.y;
        if (x == b1.x)
            return b1.y;
        Vector ab = b1 - a1;
        return a1.y + (x - a1.x) * (ab.y / ab.x);
    }
    bool operator<(Segment o) const
    { // orden de segmentos en un punto (x=cte)
        return (eval() - o.eval()) < -1e-13;
    }
};

bool ccw(const Point &a, const Point &m, const Point &b) {
    return ((a - m) ^ (b - m)) > EPS; }

bool collinear(const Point &a, const Point &b, const Point &c) {
    return fabs((b - a) ^ (c - a)) < EPS; }

double dist_sq(const Point &a, const Point &b) {
    return (a - b).norm_sq(); }

double dist(const Point &a, const Point &b) {
    return (a - b).norm(); }

bool in_segment(const Point &p, const Point &b, const Point &c) {
    return fabs(dist_sq(p, b) + dist_sq(p, c) - dist_sq(b, c)) < EPS; }

double angle(const Point &a, const Point &m, const Point &b) {
    Point ma = a - m, mb = b - m;
    return atan2(ma ^ mb, ma * mb);} //atan2l

void sweep_space() {
    vector<Event> eventos; // puntos, segmentos, ...
    sort(eventos);        // sort por x, y, ...
    set<Info> estado;     // mantener la informacion ordenada
    forn(i, sz(eventos)) {
        Event &e = eventos[i];
        process(e, estado); // procesar un evento cambia el estado
        ans = actualizar(ans);
} }

vector<pt> minkowski_sum(vector<pt> p, vector<pt> q){
    int n=SZ(p),m=SZ(q),x=0,y=0;
    fore(i,0,n) if(p[i]<p[x]) x=i;
    fore(i,0,m) if(q[i]<q[y]) y=i;
    vector<pt> ans={p[x]+q[y]};
    fore(it,1,n+m){
        pt a=p[(x+1)%n]+q[y];
        pt b=p[x]+q[(y+1)%m];
        if(b.left(ans.back(),a)) ans.pb(b), y=(y+1)%m;
        else ans.pb(a), x=(x+1)%n;
    }
    return ans; }
```

4.1 Lower Envelope

```
const ll is_query = -(1LL<<62);
struct Line {
    ll m, b;
    mutable multiset<Line>::iterator it;
    const Line *succ(multiset<Line>::iterator it) const;
    bool operator<(const Line & rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line *s = succ(it);
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b > (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line> {
    bool bad(iterator y) {
        iterator z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b >= z->b;
        }
        iterator x = prev(y);
        if (z == end()) return y->m == x->m && y->b >= x->b;
        return (x->m-z->m)*(z->b-y->b) >= (z->b-x->b)*(y->m-z->m);
    }
    iterator next(iterator y) {return ++y;}
    iterator prev(iterator y) {return --y;}
    void insert_line(ll m, ll b) {
        iterator y = insert((Line) {m, b});
        y->it = y;
        if (bad(y)) {erase(y); return;}
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        Line l = *lower_bound((Line) {x, is_query});
        return l.m * x + l.b;
    }
} h;
const Line *Line::succ(multiset<Line>::iterator it) const {
    return (++it==h.end() ? NULL : &*it); }
```

5 Strings

5.1 Hashing

```
struct StrHash { // Hash polinomial con exponentes decrecientes.
    static constexpr ll ms[] = {1'000'000'007, 1'000'000'403};
    static constexpr ll b = 500'000'000;
    vector<ll> hs[2], bs[2];
    StrHash(string const& s) {
        int n = sz(s);
        forn(k, 2) {
            hs[k].resize(n+1), bs[k].resize(n+1, 1);
            forn(i, n) {
                hs[k][i+1] = (hs[k][i] * b + s[i]) % ms[k];
                bs[k][i+1] = bs[k][i] * b       % ms[k];
            }
        }
    }
    ll get(int idx, int len) const { // Hashes en `s[idx, idx+len)`.
        ll h[2];
        forn(k, 2) {
            h[k] = hs[k][idx+len] - hs[k][idx] * bs[k][len] % ms[k];
            if (h[k] < 0) h[k] += ms[k];
        }
        return (h[0] << 32) | h[1];
    }
};
```

5.2 Suffix Array

```
#define RB(x) ((x) < n ? r[x] : 0)
void csort(vector<int>& sa, vector<int>& r, int k) {
    int n = sz(sa);
    vector<int> f(max(255, n)), t(n);
    forn(i, n) ++f[RB(i+k)];
    int sum = 0;
    forn(i, max(255, n)) f[i] = (sum += f[i]) - f[i];
    forn(i, n) t[f[RB(sa[i]+k)]++] = sa[i];
    sa = t;
}
vector<int> compute_sa(string& s){ // O(n*log2(n))
    int n = sz(s) + 1, rank;
    vector<int> sa(n), r(n), t(n);
    iota(all(sa), 0);
    forn(i, n) r[i] = s[i];
    for (int k = 1; k < n; k *= 2) {
        csort(sa, r, k), csort(sa, r, 0);
        t[sa[0]] = rank = 0;
        forr(i, 1, n) {
            if(r[sa[i]] != r[sa[i-1]] || RB(sa[i]+k) !=
                RB(sa[i-1]+k)) ++rank;
            t[sa[i]] = rank;
        }
        r = t;
        if (r[sa[n-1]] == n-1) break;
    }
    return sa; // sa[i] = i-th suffix of s in lexicographical order
}
vector<int> compute_lcp(string& s, vector<int>& sa){
    int n = sz(s) + 1, L = 0;
    vector<int> lcp(n), plcp(n), phi(n);
    phi[sa[0]] = -1;
    forr(i, 1, n) phi[sa[i]] = sa[i-1];
    forn(i,n) {
        if (phi[i] < 0) { plcp[i] = 0; continue; }
        while(s[i+L] == s[phi[i]+L]) ++L;
        plcp[i] = L;
        L = max(L - 1, 0);
    }
    forn(i, n) lcp[i] = plcp[sa[i]];
    return lcp; // lcp[i] = longest common prefix between sa[i-1]
        and sa[i]
}
```

5.3 Kmp

```
template<class Char=char>struct Kmp {
    using str = basic_string<Char>;
    vector<int> pi; str pat;
    Kmp(str const& _pat): pi(move(pfun(_pat))), pat(_pat) {}
    vector<int> matches(str const& txt) const {
        if (sz(pat) > sz(txt)) {return {};}
        vector<int> occs; int m = sz(pat), n = sz(txt);
        if (m == 0) {occs.push_back(0);}
        int j = 0;
        forn(i, n) {
            while (j != 0 && txt[i] != pat[j]) {j = pi[j-1];}
            if (txt[i] == pat[j]) {++j;}
            if (j == m) {occs.push_back(i - j + 1);}
        }
        return occs;
    }
};
```

5.4 Manacher

```
struct Manacher {
    vector<int> p;
    Manacher(string const& s) {
        int n = sz(s), m = 2*n+1, l = -1, r = 1;
        vector<char> t(m); forn(i, n) t[2*i+1] = s[i];
        p.resize(m); forr(i, 1, m) {
            if (i < r) p[i] = min(r-i, p[l+r-i]);
            while (p[i] <= i && i < m-p[i] && t[i-p[i]] ==
                t[i+p[i]]) ++p[i];
            if (i+p[i] > r) l = i-p[i], r = i+p[i];
        }
    } // Retorna palindromos de la forma {comienzo, largo}.
    pii at(int i) const {int k = p[i]-1; return pair{i/2-k/2, k};}
    pii odd(int i) const {return at(2*i+1);} // Mayor centrado en
        s[i].
    pii even(int i) const {return at(2*i);} // Mayor centrado en
        s[i-1,i].
};
```

5.5 String Functions

```
template<class Char=char>vector<int> pfun(basic_string<Char>const&
    w) {
    int n = sz(w), j = 0; vector<int> pi(n);
    forr(i, 1, n) {
        while (j != 0 && w[i] != w[j]) {j = pi[j - 1];}
        if (w[i] == w[j]) {++j;}
        pi[i] = j;
    } // pi[i] = lengh of longest proper suffix of w[0..i] that is also prefix
    return pi;
}
template<class Char=char>vector<int> zfun(const basic_string<Char>& w) {
    int n = sz(w), l = 0, r = 0; vector<int> z(n);
    forr(i, 1, n) {
        if (i <= r) {z[i] = min(r - i + 1, z[i - l]);}
        while (i + z[i] < n && w[z[i]] == w[i + z[i]]) {++z[i];}
        if (i + z[i] - 1 > r) {l = i, r = i + z[i] - 1;}
    } // z[i] = lengh of longest prefix of w that also begins at index i
    return z;
}
```

6 Flujo

6.1 Dinic

```
struct Dinic{
    int nodes,src,dst;
    vector<int> dist,q,work;
    struct edge {int to,rev;ll f,cap;};
    vector<vector<edge>> g;
    Dinic(int x):nodes(x),g(x),dist(x),q(x),work(x){}
    void add_edge(int s, int t, ll cap){
        //dprint(s); dprint(t); dprint(cap);
        g[s].pb((edge){t,sz(g[t]),0,cap});
        g[t].pb((edge){s,sz(g[s])-1,0,0});
    }
    bool dinic_bfs(){
        fill(all(dist),-1);dist[src]=0;
        int qt=0;q[qt++]=src;
        for(int qh=0;qh<qt;qh++){
            int u=q[qh];
            forn(i,sz(g[u])){
                edge &e=g[u][i];int v=g[u][i].to;
                if(dist[v]<0&&e.f<e.cap)dist[v]=dist[u]+1,q[qt++]=v;
            }
        }
        return dist[dst]>=0;
    }
    ll dinic_dfs(int u, ll f){
        if(u==dst)return f;
        for(int &i=work[u];i<sz(g[u]);i++){
            edge &e=g[u][i];
            if(e.cap<=e.f)continue;
            int v=e.to;
            if(dist[v]==dist[u]+1){
                ll df=dinic_dfs(v,min(f,e.cap-e.f));
                if(df>0){e.f+=df;g[v][e.rev].f-=df;return df;}
            }
        }
        return 0;
    }
    ll max_flow(int _src, int _dst){
        src=_src;dst=_dst;
        ll result=0;
        while(dinic_bfs()){
            fill(all(work),0);
            while(ll delta=dinic_dfs(src,INF))result+=delta;
        }
        return result;
    }
};
```

7 Otros

7.1 Fijar el numero de decimales

```
cout.precision(7); fixed(cout);
cout << x << " " << y;
// otra forma
cout.precision(7);
cout << fixed << x << " " << fixed << y;
```