Ayudamemoria

My room is random Sorted

28 de octubre de 2024

Índice

1. Template

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

```

```cpp
#define forr(i, a, b) for (int i = int(a); i < int(b); i++)
#define forn(i, n) forr(i,0,n)
#define dforr(i, a, b) for (int i = int(b)-1; i >= int(a); i--)
#define dforn(i, n) dforr(i,0,n)
#define all(v) begin(v),end(v)
#define sz(v) (int(size(v)))
#define pb push_back
#define fst first
#define snd second
#define mp make_pair
#define endl '\n'
#define dprint(v) cout << #v " = " << v << endl

typedef long long ll;
typedef pair<int, int> pii;

int main() {
    ios::sync_with_stdio(0); cin.tie(0);
}
```

1.1. run.sh

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

1.2. comp.sh

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

1.3. Makefile

```make
CXXFLAGS = -std=gnu++2a -O2 -g -Wall -Wextra -Wshadow -Wconversion \
-fsanitize=address -fsanitize=undefined
```

2. Estructuras de datos

2.1. Sparse Table

```cpp
#define oper min
int st[K][1<<K]; // K tal que (1<<K) > n
void st_init(vector<int>& a) {
    int n = sz(a); // assert(K >= 31-__builtin_clz(2*n));
    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 l, int r) { // assert(l<r);
    int k = 31-__builtin_clz(r-l);
    return oper(st[k][l], st[k][r-(1<<k)]);
}
```

2.2. Segment Tree

```cpp
// Dado un array y una operacion asociativa con neutro, get(i,j) opera en [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(){dforn(i, sz) t[i]=oper(t[2*i], t[2*i+1]);} // O(N)
    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 = 10'000'000;
```

```
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 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 sii 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;
        d=gcd(x-y,n);
    }
    return d==n ? rho(n) : d;
}

void factRho(map<ll,ll>&prim, 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);
}
auto factRho(ll n){
    map<ll,ll>prim;
    factRho(prim,n);
    return prim;
}
```

3.3. Divisores

```
// Usar asi: divisores(fac, divs, fac.begin()); NO ESTA ORDENADO
void divisores(const map<ll,ll> &f, vector<ll> &divs, auto it, ll n=1){
    if (it==f.begin()) divs.clear();
    if (it==f.end()) { divs.pb(n); return; }
    ll p=it->fst, k=it->snd; ++it;
    forn(_, k+1) divisores(f,divs,it,n), n*=p;
}
```

3.4. Inversos Modulares

```
pair<ll,ll> extended_euclid(ll a, ll b) {
    if (b == 0) return {1, 0};
    auto [y, x] = extended_euclid(b, a%b);
    y -= (a/b)*x;
    if (a*x + b*y < 0) x = -x, y = -y;
    return {x, y}; // a*x + b*y = gcd(a,b)
}
```

```
constexpr ll MOD = 1000000007; // tmb es comun 998'244'353
ll invmod[MAXN]; // inversos módulo MOD hasta MAXN
void invmods() { // todo entero en [2,MAXN] debe ser coprimo con MOD
    inv[1] = 1;
    forr(i, 2, MAXN) inv[i] = MOD - MOD/i*inv[MOD%i]%MOD;
}

// si MAXN es demasiado grande o MOD no es fijo:
// versión corta, m debe ser primo. O(log(m))
ll invmod(ll a, ll m) { return expmod(a,m-2,m); }
// versión larga, a y m deben ser coprimos. O(log(a)), en general más rápido
ll invmod(ll a, ll m) { return (extended_euclid(a,m).fst % m + m) % m; }
```

4. Geometria

4.1. Formulas

- **Ley de cosenos**: sea un triangulo con lados A, B, C y angulos α, β, γ entre A, B y C, respectivamente.

$$A^2 = B^2 + C^2 - 2 * cos(\alpha)$$

$$B^2 = A^2 + C^2 - 2 * cos(\beta)$$

$$C^2 = A^2 + B^2 - 2 * cos(\gamma)$$

- **Ley de senos**: idem

$$\frac{sin(\alpha)}{A} = \frac{sin(\beta)}{B} = \frac{sin(\gamma)}{C}$$

- **Valor de PI**: $\pi = acos(-1{,}0)$ o $\pi = 4 * atan(1{,}0)$

- **Longitud de una cuerda**: sea α el angulo descripto por una cuerda de longitud l.

$$l = \sqrt{2 * r^2 * (1 - cos(\alpha))}$$

- **Formula de Heron**: sea un triangulo con lados a, b, c y semiperimetro s. El area del triangulo es

$$A = \sqrt{s * (s-a) * (s-b) * (s-c)}$$

- **Teorema de Pick**: sean A, I y B el area de un poligono, la cantidad de puntos con coordenadas enteras dentro del mismo y la cantidad de puntos con coordenadas enteras en el borde del mismo.

$$A = I + \frac{B}{2} - 1$$

4.2. Punto

```
struct pt {
    tipo x, y;
    // tipo x, y, z; // only for 3d
    pt() {}
    pt(tipo _x, tipo _y) : x(_x), y(_y) {}
    // pt(tipo _x, tipo _y, tipo _z) : x(_x), y(_y), z(_z) {} // for 3d
    tipo norm2(){return *this**this;}
    tipo norm(){return sqrt(norm2());}
    pt operator+(pt o){return pt(x+o.x,y+o.y);}
    pt operator-(pt o){return pt(x-o.x,y-o.y);}
    pt operator*(tipo u){return pt(x*u,y*u);}
    pt operator/(tipo u) {
        if (u == 0) return pt(INF,INF);
        return pt(x/u,y/u);
    }
    tipo operator*(pt o){return x*o.x+y*o.y;}
// pt operator^(pt p){ // only for 3D
//      return pt(y*p.z-z*p.y,z*p.x-x*p.z,x*p.y-y*p.x);}
    tipo operator^(pt o){return x*o.y-y*o.x;}
    tipo angle(pt o){return atan2(*this^o,*this*o);}
    pt unit(){return *this/norm();}
    bool left(pt p, pt q){ // is it to the left of directed line pq?
        return ((q-p)^(*this-p))>EPS;}
    bool operator<(pt p)const{ // for convex hull
        return x<p.x-EPS||(abs(x-p.x)<=EPS&&y<p.y-EPS);}
    bool collinear(pt p, pt q){return fabs((p-*this)^(q-*this))<EPS;}
    pt rot(pt r){return pt(*this^r,*this*r);}
    pt rot(tipo a){return rot(pt(sin(a),cos(a)));}
};
pt ccw90(1,0);
pt cw90(-1,0);
```

4.3. Linea

```
int sgn2(tipo x){return x<0?-1:1;}
struct ln {
    pt p,pq;
    ln(pt p, pt q):p(p),pq(q-p){}
    ln(){}
    bool has(pt r){return dist(r)<=EPS;}
    bool seghas(pt r){return has(r)&&(r-p)*(r-(p+pq))<=EPS;}
//  bool operator /(ln l){return (pq.unit()^l.pq.unit()).norm()<=EPS;} // 3D
    bool operator/(ln l){return abs(pq.unit()^l.pq.unit())<=EPS;} // 2D
    bool operator==(ln l){return *this/l&&has(l.p);}
    pt operator^(ln l){ // intersection
        if(*this/l)return pt(INF,INF);
        tipo a=-pq.y, b=pq.x, c=p.x*a+p.y*b;
        tipo la=-l.pq.y, lb=l.pq.x, lc=l.p.x*la+l.p.y*lb;
        tipo det = a * lb - b * la;
        pt r((lb*c-b*lc)/det, (a*lc-c*la)/det);
        return r;
//      pt r=l.p+l.pq*(((p-l.p)^pq)/(l.pq^pq));
//      if(!has(r)){return pt(NAN,NAN,NAN);} // check only for 3D
    }
    tipo angle(ln l){return pq.angle(l.pq);}
    int side(pt r){return has(r)?0:sgn2(pq^(r-p));} // 2D
    pt proj(pt r){return p+pq*((r-p)*pq/pq.norm2());}
    pt segclosest(pt r) {
        double l2 = pq.norm2();
        if(l2==0.) return p;
        double t =((r-p)*pq)/l2;
        return p+(pq*min(1,max(0,t)));
    }
    pt ref(pt r){return proj(r)*2-r;}
    tipo dist(pt r){return (r-proj(r)).norm();}
//  tipo dist(ln l){ // only 3D
//      if(*this/l)return dist(l.p);
//      return abs((l.p-p)*(pq^l.pq))/(pq^l.pq).norm();
//  }
    ln rot(auto a){return ln(p,p+pq.rot(a));} // 2D
};
ln bisector(ln l, ln m){ // angle bisector
    pt p=l^m;
    return ln(p,p+l.pq.unit()+m.pq.unit());
}
ln bisector(pt p, pt q){ // segment bisector (2D)
    return ln((p+q)*.5,p).rot(ccw90);
}
```

4.4. Poligono

```
struct pol {
    int n;vector<pt> p;
    pol(){}
    pol(vector<pt> _p){p=_p;n=p.size();}
    double area() {
        ll a = 0;
        forr (i, 1, sz(p)-1) {
            a += (p[i]-p[0])^(p[i+1]-p[0]);
        }
        return abs(a)/2;
    }
    bool has(pt q){ // O(n), winding number
        forr(i,0,n)if(ln(p[i],p[(i+1)%n]).seghas(q))return true;
        int cnt=0;
        forr(i,0,n){
            int j=(i+1)%n;
            int k=sgn((q-p[j])%(p[i]-p[j]));
            int u=sgn(p[i].y-q.y),v=sgn(p[j].y-q.y);
            if(k>0&&u<0&&v>=0)cnt++;
            if(k<0&&v<0&&u>=0)cnt--;
        }
        return cnt!=0;
    }
    void normalize(){ // (call before haslog, remove collinear first)
        if(p[2].left(p[0],p[1]))reverse(p.begin(),p.end());
        int pi=min_element(p.begin(),p.end())-p.begin();
        vector<pt> s(n);
        forr(i,0,n)s[i]=p[(pi+i)%n];
        p.swap(s);
    }
    bool haslog(pt q){ // O(log(n)) only CONVEX. Call normalize first
        if(q.left(p[0],p[1])||q.left(p.back(),p[0]))return false;
        int a=1,b=p.size()-1; // returns true if point on boundary
        while(b-a>1){       // (change sign of EPS in left
            int c=(a+b)/2;    //  to return false in such case)
            if(!q.left(p[0],p[c]))a=c;
            else b=c;
        }
        return !q.left(p[a],p[a+1]);
    }
    bool isconvex(){//O(N), delete collinear points!
        int N=sz(p);
        if(N<3) return false;
        bool isLeft=p[0].left(p[1], p[2]);
        forr(i, 1, N)
            if(p[i].left(p[(i+1)%N], p[(i+2)%N])!=isLeft)
                return false;
        return true;
    }
    pt farthest(pt v){ // O(log(n)) only CONVEX
        if(n<10){
            int k=0;
            forr(i,1,n)if(v*(p[i]-p[k])>EPS)k=i;
            return p[k];
        }
        if(n==sz(p))p.pb(p[0]);
        pt a=p[1]-p[0];
        int s=0,e=n,ua=v*a>EPS;
        if(!ua&&v*(p[n-1]-p[0])<=EPS)return p[0];
        while(1){
            int m=(s+e)/2;pt c=p[m+1]-p[m];
            int uc=v*c>EPS;
            if(!uc&&v*(p[m-1]-p[m])<=EPS)return p[m];
            if(ua&&(!uc||v*(p[s]-p[m])>EPS))e=m;
            else if(ua||uc||v*(p[s]-p[m])>=-EPS)s=m,a=c,ua=uc;
            else e=m;
            assert(e>s+1);
        }
    }
    pol cut(ln l){ // cut CONVEX polygon by line l
        vector<pt> q; // returns part at left of l.pq
        forr(i,0,n){
            int d0=sgn(l.pq%(p[i]-l.p)),d1=sgn(l.pq%(p[(i+1)%n]-l.p));
            if(d0>=0)q.pb(p[i]);
            ln m(p[i],p[(i+1)%n]);
            if(d0*d1<0&&!(l/m))q.pb(l^m);
        }
        return pol(q);
    }
    double intercircle(circle c){ // area of intersection with circle
        double r=0.;
        forr(i,0,n){
            int j=(i+1)%n;double w=c.intertriangle(p[i],p[j]);
            if((p[j]-c.o)%(p[i]-c.o)>0)r+=w;
            else r-=w;
        }
        return abs(r);
    }
    double callipers(){ // square distance of most distant points
        double r=0;   // prereq: convex, ccw, NO COLLINEAR POINTS
        for(int i=0,j=n<2?0:1;i<j;++i){
            for(;;j=(j+1)%n){
                r=max(r,(p[i]-p[j]).norm2());
                if((p[(i+1)%n]-p[i])%(p[(j+1)%n]-p[j])<=EPS)break;
            }
        }
        return r;
    }
};
// Dynamic convex hull trick
vector<pol> w;
void add(pt q){ // add(q), O(log^2(n))
    vector<pt> p={q};
    while(!w.empty()&&sz(w.back().p)<2*sz(p)){
        for(pt v:w.back().p)p.pb(v);
        w.pop_back();
    }
    w.pb(pol(chull(p)));
}
ll query(pt v){ // max(q*v:q in w), O(log^2(n))
    ll r=-INF;
    for(auto& p:w)r=max(r,p.farthest(v)*v);
    return r;
}
```

4.5. Circulo

```
struct circle {
    pt o;double r;
    circle(pt o, double r):o(o),r(r){}
    circle(pt x, pt y, pt z){o=bisector(x,y)^bisector(x,z);r=(o-x).norm();}
    bool has(pt p){return (o-p).norm()<=r+EPS;}
    vector<pt> operator^(circle c){ // ccw
        vector<pt> s;
        double d=(o-c.o).norm();
        if(d>r+c.r+EPS||d+min(r,c.r)+EPS<max(r,c.r))return s;
        double x=(d*d-c.r*c.r+r*r)/(2*d);
        double y=sqrt(r*r-x*x);
        pt v=(c.o-o)/d;
        s.pb(o+v*x-v.rot(ccw90)*y);
        if(y>EPS)s.pb(o+v*x+v.rot(ccw90)*y);
        return s;
    }
    vector<pt> operator^(ln l){
        vector<pt> s;
        pt p=l.proj(o);
        double d=(p-o).norm();
        if(d-EPS>r)return s;
        if(abs(d-r)<=EPS){s.pb(p);return s;}
        d=sqrt(r*r-d*d);
        s.pb(p+l.pq.unit()*d);
        s.pb(p-l.pq.unit()*d);
        return s;
    }
    vector<pt> tang(pt p){
        double d=sqrt((p-o).norm2()-r*r);
        return *this^circle(p,d);
    }
    bool in(circle c){ // non strict
        double d=(o-c.o).norm();
        return d+r<=c.r+EPS;
    }
    double intertriangle(pt a, pt b){ // area of intersection with oab
        if(abs((o-a)%(o-b))<=EPS)return 0.;
        vector<pt> q={a},w=*this^ln(a,b);
        if(w.size()==2)for(auto p:w)if((a-p)*(b-p)<-EPS)q.pb(p);
        q.pb(b);
        if(q.size()==4&&(q[0]-q[1])*(q[2]-q[1])>EPS)swap(q[1],q[2]);
        double s=0;
        fore(i,0,q.size()-1){
            if(!has(q[i])||!has(q[i+1]))s+=r*r*(q[i]-o).angle(q[i+1]-o)/2;
            else s+=abs((q[i]-o)%(q[i+1]-o)/2);
        }
        return s;
    }
};
```

4.6. Convex Hull

```
// CCW order
// Includes collinear points (change sign of EPS in left to exclude)
vector<pt> chull(vector<pt> p){
    if(sz(p)<3)return p;
    vector<pt> r;
    sort(p.begin(),p.end()); // first x, then y
    forr(i,0,p.size()){ // lower hull
        while(r.size()>=2&&r.back().left(r[r.size()-2],p[i]))r.pop_back();
        r.pb(p[i]);
    }
    r.pop_back();
    int k=r.size();
    for(int i=p.size()-1;i>=0;--i){ // upper hull
        while(r.size()>=k+2&&r.back().left(r[r.size()-2],p[i]))r.pop_back();
        r.pb(p[i]);
    }
    r.pop_back();
    return r;
}
```

4.7. Orden Radial

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

// no debe haber un punto igual al pivot en el rango [b, e]
// en general usar la direccion (1,0)
void radial_sort(vector<pt>::iterator b,
    vector<pt>::iterator e, pt pivot, pt dir) {
    sort(b, e, Comp(pivot, dir)); }
```

4.8. Par de puntos más cercano

```
double INF=8e18+1;
#define dist(a, b) ((a-b).norm_sq())
bool compy(pt a, pt b) {
    return mp(a.y,a.x)<mp(b.y,b.x); }
bool compx(pt a, pt b) {
    return mp(a.x,a.y)<mp(b.x,b.y); }
// los puntos deben estar ordenados por x
// inicialmente: l=0, r=sz(ps)
ll closest(vector<pt> &ps, int l, int r) {
    if (l == r-1) return INF;
    if (l == r-2) {
        sort(&ps[l], &ps[r], compy);
        return dist(ps[l], ps[l+1]); }
    int m = (l+r)/2, xm = ps[m].x;
    ll min_dist = min(closest(ps, l, m),closest(ps, m, r));
    double delta = sqrt(min_dist);
    vector<pt> sorted(r-l);
    merge(&ps[l], &ps[m], &ps[m], &ps[r], &sorted[0], compy);
    copy(all(sorted), &ps[l]);
    vector<pt> strip;
    forr (i, l, r) {
        if (ps[i].x > int(xm-delta)
        && ps[i].x <= int(xm+delta)) {
            strip.pb(ps[i]);
        }
    }
    forn (i, sz(strip)) {
        forr (j, 1, 8) {
            if (i+j >= sz(strip)) break;
            if (dist(strip[i], strip[i+j]) < min_dist)
                min_dist = dist(strip[i], strip[i+j]);
        }
    }
    return min_dist;
}
```

4.9. Arbol KD

```
// given a set of points, answer queries of nearest point in
```

```
    O(log(n))
bool onx(pt a, pt b){return a.x<b.x;}
bool ony(pt a, pt b){return a.y<b.y;}
struct Node {
    pt pp;
    ll x0=INF, x1=-INF, y0=INF, y1=-INF;
    Node *first=0, *second=0;
    ll distance(pt p){
        ll x=min(max(x0,p.x),x1);
        ll y=min(max(y0,p.y),y1);
        return (pt(x,y)-p).norm2();
    }
    Node(vector<pt>&& vp):pp(vp[0]){
        for(pt p:vp){
            x0=min(x0,p.x); x1=max(x1,p.x);
            y0=min(y0,p.y); y1=max(y1,p.y);
        }
        if(sz(vp)>1){
            sort(all(vp),x1-x0>=y1-y0?onx:ony);
            int m=sz(vp)/2;
            first=new Node({vp.begin(),vp.begin()+m});
            second=new Node({vp.begin()+m,vp.end()});
        }
    }
};
struct KDTree {
    Node* root;
    KDTree(const vector<pt>& vp):root(new Node({all(vp)})) {}
    pair<ll,pt> search(pt p, Node *node){
        if(!node->first){
            //avoid query point as answer
            //if(p==node->pp) {INF,pt()};
            return {(p-node->pp).norm2(),node->pp};
        }
        Node *f=node->first, *s=node->second;
        ll bf=f->distance(p), bs=s->distance(p);
        if(bf>bs)swap(bf,bs),swap(f,s);
        auto best=search(p,f);
        if(bs<best.fst) best=min(best,search(p,s));
        return best;
    }
    pair<ll,pt> nearest(pt p){return search(p,root);}
};
```

4.10. Suma de Minkowski

```
// normalizar los poligonos antes de hacer la suma
// si son poligonos concavos llamar a chull luego y normalizar
// si son convexos eliminar puntos colineales y normalizar
vector<pt> minkowski_sum(vector<pt> p, vector<pt> q){
    int n=sz(p),m=sz(q),x=0,y=0;
    forr(i,0,n) if(p[i]<p[x]) x=i;
    forr(i,0,m) if(q[i]<q[y]) y=i;
    vector<pt> ans={p[x]+q[y]};
    forr(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; }
```

4.11. Sweep Space

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

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);
```

```cpp
        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

```cpp
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

```cpp
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. Grafos

6.1. Dikjstra

```cpp
vector<pair<int,int>> g[MAXN]; // u->[(v,cost)]
ll dist[MAXN];
void dijkstra(int x){
    memset(dist,-1,sizeof(dist));
    priority_queue<pair<ll,int> > q;
    dist[x]=0;q.push({0,x});
    while(!q.empty()){
        x=q.top().snd;ll c=-q.top().fst;q.pop();
        if(dist[x]!=c)continue;
        forn(i,g[x].size()){
            int y=g[x][i].fst; ll c=g[x][i].snd;
            if(dist[y]<0||dist[x]+c<dist[y])
                dist[y]=dist[x]+c,q.push({-dist[y],y});
        }
    }
}
```

6.2. LCA

```cpp
int n;
```

```
vector<int> g[MAXN];

vector<int> depth, etour, vtime;

// operación de la sparse table, escribir `#define oper lca_oper`
int lca_oper(int u, int v) { return depth[u]<depth[v] ? u : v; };

void lca_dfs(int u) {
    vtime[u] = sz(etour), etour.push_back(u);
    for (auto v : g[u]) {
        if (vtime[v] >= 0) continue;
        depth[v] = depth[u]+1; lca_dfs(v); etour.push_back(u);
    }
}
auto lca_init(int root) {
    depth.assign(n,0), etour.clear(), vtime.assign(n,-1);
    lca_dfs(root); st_init(etour);
}

auto lca(int u, int v) {
    int l = min(vtime[u],vtime[v]);
    int r = max(vtime[u],vtime[v])+1;
    return st_query(l,r);
}
int dist(int u, int v) { return depth[u]+depth[v]-2*depth[lca(u,v)]; }
```

6.3. Toposort

```
vector<int> g[MAXN];int n;
vector<int> tsort(){ // lexicographically smallest topological sort
    vector<int> r;priority_queue<int> q;
    vector<int> d(2*n,0);
    forn(i,n)forn(j,g[i].size())d[g[i][j]]++;
    forn(i,n)if(!d[i])q.push(-i);
    while(!q.empty()){
        int x=-q.top();q.pop();r.pb(x);
        forn(i,sz(g[x])){
            d[g[x][i]]--;
            if(!d[g[x][i]])q.push(-g[x][i]);
        }
    }
    return r; // if not DAG it will have less than n elements
}
```

7. Flujo

7.1. 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){
        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;
    }
};
```

7.2. Kuhn

```
vector<int> g[MAXN];
vector<bool> vis;
vector<int> match;

bool kuhn_dfs(int u){
    if (vis[u]) return false;
    vis[u] = true;
    for (int v : g[u]) if (match[v] == -1 || kuhn_dfs(match[v])) {
        match[v] = u;
        return true;
    } return false;
}

vector<int> kuhn(int n) {
    match.resize(n, -1);
    vis.resize(n);
    forn(u, n) {
        vis.assign(n, false);
        kuhn_dfs(u);
    }
    return match;
} //n: cant de nodos devuelve un vector con -1 si no matchea y sino su match
```

8. Optimización

8.1. Ternary Search

```
// mínimo entero de f en (l,r)
ll ternary(auto f, ll l, ll r) {
    for (ll d = r-l; d > 2; d = r-l) {
        ll a = l+d/3, b = r-d/3;
        if (f(a) > f(b)) l = a; else r = b;
    }
    return l+1; // retorna un punto, no un resultado de evaluar f
}

// mínimo real de f en (l,r)
// para error < EPS, usar iters = log((r-l)/EPS)/log(1.618)
double golden(auto f, double l, double r, int iters) {
    constexpr double ratio = (3-sqrt(5))/2;
    double x1 = l+(r-l)*ratio, f1 = f(x1);
    double x2 = r-(r-l)*ratio, f2 = f(x2);
    while (iters--) {
        if (f1 > f2) l=x1, x1=x2, f1=f2, x2=r-(r-l)*ratio, f2=f(x2);
        else       r=x2, x2=x1, f2=f1, x1=l+(r-l)*ratio, f1=f(x1);
    }
    return (l+r)/2; // retorna un punto, no un resultado de evaluar f
}
```

8.2. Longest Increasing Subsequence

```
int lis(vector<int> const& a) {
    int n = a.size();
    const int INF = 1e9;
    vector<int> d(n+1, INF);
    d[0] = -INF;
    forn(i,n){
        int l = upper_bound(all(d), a[i]) - d.begin();
        if (d[l-1] < a[i] && a[i] < d[l])
            d[l] = a[i];
    }
    int ans = 0;
    for (int l = 0; l <= n; l++) {
        if (d[l] < INF)
            ans = l;
    }
    return ans;
}
```

9. Otros

9.1. Mo

```
int n,sq,nq; // array size, sqrt(array size), #queries
struct qu{int l,r,id;};
qu qs[MAXN];
ll ans[MAXN]; // ans[i] = answer to ith query
bool qcomp(const qu &a, const qu &b){
    if(a.l/sq!=b.l/sq) return a.l<b.l;
    return (a.l/sq)&1?a.r<b.r:a.r>b.r;
}
void mos(){
    forn(i,nq)qs[i].id=i;
    sq=sqrt(n)+.5;
    sort(qs,qs+nq,qcomp);
    int l=0,r=0;
    init();
    forn(i,nq){
        qu q=qs[i];
        while(l>q.l)add(--l);
        while(r<q.r)add(r++);
        while(l<q.l)remove(l++);
        while(r>q.r)remove(--r);
        ans[q.id]=get_ans();
    }
}
```

9.2. Fijar el numero de decimales

```
// antes de imprimir decimales, con una sola vez basta
cout << fixed << setprecision(DECIMAL_DIG);
```

9.3. Hash Table (Unordered Map/ Unordered Set)

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
template<class Key,class Val=null_type>using htable=gp_hash_table<Key,Val>;
// como unordered_map (o unordered_set si Val es vacio), pero sin metodo count
```

9.4. Indexed Set

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
template<class Key, class Val=null_type>
using indexed_set = tree<Key, Val, less<Key>, rb_tree_tag,
                        tree_order_statistics_node_update>;
// indexed_set<char> s;
// char val = *s.find_by_order(0); // acceso por indice
// int idx = s.order_of_key('a'); // busca indice del valor
```

9.5. Iterar subconjuntos

- Iterar por todos los subconjuntos de n elementos $O(2^n)$.

```
    for(int bm=0; bm<(1<<n); bm++)
```

- Iterar por cada superconjunto de un subconjunto de n elementos $O(2^n)$.

```
    for(int sbm=~bm; sbm; sbm=(sbm-1)&(~bm)) // super=bm&sbm
```

- Iterar por cada subconjunto de un subconjunto de n elementos $O(2^n)$.

```
    for(int sbm=bm; sbm; sbm=(sbm-1)&bm) // sub=sbm
```

- Para cada subconjunto de n elementos, iterar por cada superconjunto $O(3^n)$.

```
    for(int bm=0; bm<(1<<n); bm++)
      for(int sbm=~bm; sbm; sbm=(sbm-1)&(~bm)) // super=bm&sbm
```

- Para cada subconjunto de n elementos, iterar por cada subsubconjunto $O(3^n)$.

```
for(int bm=0; bm<(1<<n); bm++)
  for(int sbm=bm; sbm; sbm=(sbm-1)&(bm)) // sub=sbm
```