// g++ -std=c++11 -O2 -fmax-errors=1

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

typedef long double dbl;

typedef pair<int,int> pii;

typedef vector<int> vi;

#define X first

#define Y second

#define mp make\_pair

#define pb push\_back

#define all(c) begin(c), end(c)

#define uni(c) c.resize(distance (c.begin(), unique(all(c))))

#define rep(i,n) for(ll i=0, \_n=(n); i<\_n; i++)

#define rep1(i,n) for(ll i=1, \_n=(n); i<=\_n; i++)

#define cout(d) cout << fixed << setprecision(d)

#define err(x) cerr << #x << " = " << x << "\n"

const dbl eps=1e-12, pi=acosl(-1);

const ll inf=1e16, mod=1e9+7;

const int N=2e6+10;

int main() {}

**SIEVE**

int np(0), prime[N], isprime[N];

void sieve(int N){

memset(isprime, 1, sizeof isprime);

for(int i=2; i<N; i++){

if(isprime[i]){

prime[++np] = i;

for(int j=2\*i; j<N; j+=i){

isprime[j] = false; }}}}

**BIG MULTIPLY**

inline ll mul(ll a, ll b, ll m){

ll q = ((dbl)a \* (dbl)b) / (dbl)m;

ll r = a\*b - q\*m;

return (r<0 ? r+m:r); }

**MILLER-RABIN**

bool ispmiller(ll p){ // O(30\*logp)

if(p<2) return false;

if(p==2) return true;

if(p%2==0) return false;

ll s=p-1; s>>=\_\_builtin\_ctzll(s);

rep(i,30) {

ll val=pwr(myrand(p-1)+1,s,p);

ll temp=s;

while(temp!=p-1 and 2<=val and val<=p-2){

val=mul(val,val,p);

temp<<=1; }

if(val!=p-1 && temp%2==0) return false; }

return true; }

**POLLARD-RHO**

ll pollardrho(ll n) { // O(n^0.25)

if(n==1) return 1;

if(n%2==0) return 2;

ll c=myrand(n-1)+1;

ll x=myrand(n-2)+2, y=x;

ll d=1; while(d==1) {

x=mul(x,x,n)+c; if(x>=n) x-=n;

y=mul(y,y,n)+c; if(y>=n) y-=n;

y=mul(y,y,n)+c; if(y>=n) y-=n;

d=\_\_gcd(abs(x-y),n);

if(d==n) return (ispmiller(n) ? n:pollardrho(n)); }

return d; }

**DSU**

struct dsu {

vi par, sz;

dsu(int n): par(n), sz(n), iota(all(par),0) {}

int root(int a){ return par[a]==a ? a:par[a]=root(par[a]); }

void merge(int a, int b){

a=root(a), b=root(b);

if(a==b)return;

if(sz[a]<sz[b]) swap(a,b);

sz[a]+=sz[b];

par[b]=a; }};

template<class T> struct dsu{

map<T,T> par,sz;

T root(T a){

if(par.find(a)==par.end() or par[a]==a) return par[a]=a;

return par[a]=root(par[a]); }

// same merge };

**MATRIX MULTIPLICATION**

const int M=2;

void mul(ll a[M][M], ll b[M][M]){

ll c[M][M];

rep(i,M)rep(j,M){

ll temp=0;

rep(k,M) temp += a[i][k]\*b[k][j]% mod;

c[i][j]=temp%mod; }

rep(i,M)rep(j,M) a[i][j]=c[i][j];}

void pwr(ll a[M][M], ll n){

assert(n>=0);

ll temp[M][M];

rep(i,M)rep(j,M) temp[i][j]=a[i][j], a[i][j]=i==j;

while(n){

if(n&1) mul(a,temp);

mul(temp,temp);

n>>=1; }}

**FAST INPUT**

template<class T> void in(T&p) {

p=0;char neg=0,ch=0;

while(ch<0x30 or ch>0x39){

if(ch=='-')neg=1; ch=getchar();}

while(0x30<=ch and ch<=0x39){

p=(p<<1)+(p<<3)+(ch&15);

ch=getchar();}

if(neg)p=-p; }

**FAST OUTPUT**

template<class T> void out(T n) {

char d[20],i=0;

if(n<0){putchar('-'); n=-n;}

do{d[i++]=(n%10)|0x30; n/=10;} while(n);

while(i) putchar(d[--i]);

putchar('\n'); }

**LENGTH OF INT**

int high(ll n){ return 63 - \_\_builtin\_clzll(n);} // location of highest set bit

**PRIORITY QUEUE**

#define pq(c) priority\_queue <c, vector<c>, compare<c>>

template<class T> struct compare{

bool operator()(const T& l, const T& r) const{ return l.Y > r.Y; }};

**HASHING**

#define umap(key, val) unordered\_map <key, val, hashed<key>>

template<class T> struct hashed{

size\_t operator()(T const& c)const{

return c.X+231LL\*c.Y; }};

**K’TH ORDER STATISTICS**

int kth(int i, int j){

if(i==j) return a[i];

int s(i), e(j), M(a[i+myrand(j-i+1)]);

while(s<=e){

while(s<=j and a[s]<M) s++;

while(e>=i and M<a[e]) e--;

if(s<=e) swap(a[s++], a[e--]);}

if(i<=e and k<=e) return kth(i,e);

if(s<=j and s<=k) return kth(s,j);

return M; }

**MO’S**

typedef pair<pii,int> node;

int K, a[N];

inline void add(int i) {}

inline void rem(int i) {}

int main(){

int q; cin >> q; // query count

K = q/sqrt(n);

vector<node> qu(q);

rep(i,q){

int x,y; cin>>x>>y;

qu[i]={{x-1,y-1},i};}

sort(all(qu),[](node l, node r){

if(l.X.X/K==r.X.X/K) return l.X.Y<r.X.Y;

return l.X.X<r.X.X;});

vi ans(q);

int curl(0), curr(-1), l, r;

rep(i,q){

tie(l,r)=qu[i].X;

while(curl>l) add(--curl);

while(curr<r) add(++curr);

while(curl<l) rem(curl++);

while(curr>r) rem(curr--);

ans[qu[i].Y]=getans(); }}

**COMBINATORICS**

ll fac[N], inv[N];

inline void init(int N){

fac[0]=1; rep1(i,N) fac[i]=fac[i-1]\*i%mod;

inv[N]=pwr(fac[N],mod-2);

for(ll i=N; i>=1; i--) inv[i-1]=inv[i]\*i%mod; }

inline ll ncr(ll n, ll r){

if(n<0 or r<0 or n<r) return 0;

return fac[n]\*inv[r]%mod\*inv[n-r]%mod; }

182537 -24976361-911046887 -996955601

257 577 1009 2213

**SEARCHING**

While(l<r) {

//minimum ok..

mid=l+r>>1;

if(ok(mid))r=mid;

else l=mid+1; }

while(l<r) {

//maximum ok..

mid=l+r+1>>1;

if(ok(mid)) l=mid;

else r=mid-1; }

while(l<r) {

int lt=(l+l+r)/3, rt=(l+r+r)/3;

if(f(lt)<f(rt)) l=lt;

else r=rt; }

**RMQ**

const int K=25;

int n, a[N], lg[N], m[N][K];

inline void init() {

lg[0]=-2;

rep(i,n+1) lg[i]=1+lg[i>>1];

rep(j,K){

if(!j) rep(i,n){ m[i][j]=a[i];}

else rep(i,n-(1<<j)+1){ m[i][j]=min(m[i][j], m[i+(1<<j-1)][j-1]);} }}

inline int rmq(int l, int r) {

int k=lg[r-l+1];

return min(m[l][k], m[r-(1<<k)+1] [k]); }

**SEGMENT TREE**

struct nod {} t[N<<1];

int n, a[N];

inline nod merge(nod a, nod b) {}

inline void build() {

for(int i=n;i--;) t[n+i]=nod(a[i]);

for(int i=n;--i;) t[i]=merge( t[i<<1], t[i<<1|1]); }

inline void upd(int p, int v){

t[n+p]=nod(v); // or merge

for(int i=n+p;i>>=1;) t[i]=merge( t[i<<1], t[i<<1|1]); }

inline nod get(int l, int r){

nod lt(), rt(); // identity

for(l+=n,r+=n+1; l<r; l>>=1,r>>=1){

if(l&1) lt=merge(lt, t[l++]);

if(r&1) rt=merge(t[--r], rt);}

return merge(lt,rt); }

**BIT**

**POINT UPDATE**

ll bt[N], a[N], nb;

inline void upd(int x, ll val) {

x++;

for(int i=x; i<=nb; i+=i&-i)

bt[i]+=val;}

inline ll sumh(int x) {

ll sum=0;

for(int i=x; i>0; i^=i&-i)

sum+=bt[i];

return sum; }

inline ll sum(int x, int X) {

X++;

return sumh(X) - sumh(x); }

**POINT QUERY**

ll bt[N], a[N], nb;

inline ll get(int x) {

ll sum=0; x++;

for(int i=x; i>0; i^=i&-i)

sum+=bt[i];

return sum; }

inline void updh(int x, ll val) {

x++;

for(int i=x; i<=nb; i+=i&-i)

bt[i]+=val; }

void upd(int x, int X, ll val) {

X++;

updh(X, -val);

updh(x, val); }

**COMPLETE BIT**

ll bt1[N], bt2[N], nb;

void updh(int x, ll val, ll \*bt) {

x++;

for(int i=x; i<=nb; i+=i&-i)

bt[i]+=val; }

void upd(int x, int X, ll val) {

X++;

updh(X, -val, bt1);

updh(X, -val\*X, bt2);

updh(x, val, bt1);

updh(x, val\*x, bt2); }

inline ll sumhh(int x, ll \*bt) {

ll sum=0;

for(int i=x; i>0; i^=i&-i)

sum+=bt[i];

return sum; }

inline ll sumh(int x) {

return sumhh(x, bt1)\*x - sumhh(x, bt2); }

inline ll sum(int x, int X) {

X++;

return sumh(X) - sumh(x); }

**ORDERED SET**

#include<ext/pb\_ds/assoc\_container.hpp>

#include <ext/pb\_ds/tree\_policy.hpp>

using namespace \_\_gnu\_pbds;

template<class T> using oset = tree<T,null\_type,less<T>,rb\_tree\_tag,tree\_order\_statistics\_node\_update>;

s.insert(x); s.erase(x); s.erase(s.lower\_bound(x));

s.order\_of\_key(x); // 0-based index of least a[i] >= x

\*s.find\_by\_order(i); // a[i]

isnan(d); // returns true if double 'd' is NaN

round(), floor(), ceil(), trunc(); // 5.5-> 6, 5, 6, 5 and -5.5-> -6, -6, -5, -5

sqrt(), cbrt(), exp(), log(), log2(), log10(); //

frac = modf(num, &inte); // double 'num' is divided into double 'inte' and double 'frac, all having same sign

b.count(), b.any(), b.none(), b.all(), b.flip(); // oeprations on bitset 'b'

cin >> ws;//eat whitespaces

c = cin.peek();//get next character, maybe EOF

cin.putback(c); //put c in stream

**GEOMETRY**

struct pt {

dbl x, y;

pt(dbl xx=0, dbl yy=0){x=xx;y=yy;}

dbl operator\*(pt p){ return x\*p.x + y\*p.y;}

dbl operator%(pt p){ return x\*p.y - y\*p.x;}

pt operator+(pt p){ return pt(x+ p.x, y+p.y);}

pt operator-(pt p){ return pt(x-p.x, y-p.y);}

pt operator\*(dbl k){ return pt(x\*k, y\*k);}

pt operator/(dbl k){ assert(k>eps); return pt(x/k, y/k);}

pt rot(dbl th=pi/2) { return rot( cos(th), sin(th));}

pt rot(dbl cosa, dbl sina) { return \*this\*cosa + pt(-y,x)\*sina;} };

typedef vector<pt> vp;

dbl len(pt p) { return sqrt(p\*p);}

pt dir(pt p) { return p/len(p);}

pt operator\*(dbl k, pt p) { return p\*k;}

bool operator==(pt p, pt q) { return fabs(len(p-q))<eps;}

ostream& operator<<(ostream& os, pt p){os<<"("<<p.x<<", "<<p.y<<")\n";

return os;}

istream& operator>>(istream& is, pt& p) {is>>p.x>>p.y; return is;}

bool operator<(const pt &p, const pt &q){

if(fabs(p.x-q.x)<eps) return p.y<q.y;

return p.x<q.x;}

pt foot(pt p, pt q, pt r){

return p + dir(q-p)\*((r-p)\*dir(q-p));}

bool lineline(pt p1, pt p2, pt p3, pt p4, pt &p){

dbl d = (p4-p3)%(p2-p1);

if(fabs(d)<eps) return false;

d=((p1-p3)%(p4-p3))/d; // sine rule

p=p1+(p2-p1)\*d;

return true; }

dbl angle(pt p, pt q, pt r){ // angle pq->pr in (-pi, pi]

pt base=dir(q-p);

return atan2((r-p)\*base.rot(), (r-p)\*base); }

int sense(pt p, pt q, pt r){

dbl d=(q-p)%(r-p);

if(d>eps) return 1;

if(d<-eps) return -1;

return 0; }

vp linecir(pt c, dbl r, pt p, pt q){

pt b=foot(p,q,c); dbl d=len(b-c);

if(d>r+eps) return {};

if(fabs(r-d)<eps) return {b};

d=sqrt(r\*r-d\*d);

return {b+d\*dir(p-q), b+d\*dir(q-p)}; }

**CONVEX HULL**

pt p0;

void hull(vp v, vp &w){

int n=v.size(), id(0);

rep(i,n){

if(fabs(v[i].y-v[id].y)<eps and v[i].x<v[id].x-eps) id=i;

else if(v[i].y<v[id].y-eps) id=i;}

p0=v[id]; v.erase(v.begin()+id);

sort(all(v), [](pt p, pt q){

int sg=sense(p0, p, q);

if(sg==0) return len(p-p0) < len(q-p0)-eps;

return sg>0;});

int m=1;

w.clear(); w.pb(p0);

for(pt p: v){

while(m>1 and sense(w[m-2],w[m-1],p)!=1){

w.pop\_back(); m--;}

if(len(p-w.back())>eps) w.pb(p), m++;} }

**CLOSEST POINT**

struct CP {

vp p;

CP(vp &a) {p=a;}

dbl getl(vp& strip, dbl d){

sort(all(strip), [](pt a, pt b){ return a.y < b.y;});

int n=strip.size();

dbl ans=d;

rep(i,n)for(int j=i+1; j<=i+4; j++) if(j<n) ans=min(ans, len( strip[i]-strip[j]));

return ans; }

dbl get(int l, int r){

if(l==r) return inf;

int m=(l+r)/2;

dbl d=min(get(l,m),get(m+1,r));

vp strip;

for(int i=l; i<=r; i++){

if(abs(p[i].x-p[mid].x) < d) strip.pb(p[i]);}

return getl(strip, d); }

dbl closest(){

sort(all(p));

return get(0, (int)p.size()-1);}};

**CONVEX OPT**

vector<ll> M, B;

int sz;

inline void add(ll m, ll b){

// in strictly monotonic order of m

while(sz>1 and (dbl)(b-B[sz-2])/(M[sz-2]-m) < (dbl)(B[sz-1]-B[sz-2])/(M[sz-2]-M[sz-1])){

M.pop\_back(); B.pop\_back(); sz—-;}

M.pb(m); B.pb(b); sz++; }

// queries in increasing order

int pt;

inline ll query(ll x){

while(pt<sz-1 and x>(dbl)(B[pt]-B[pt+1])/(M[pt+1]-M[pt])) pt++;

return M[pt]\*x+B[pt]; }

// random queries

inline ll query(int x) {

int l=0, r=sz-1;

while(l<r){

int m1=(l+l+r)/3, m2=(l+r+r)/3;

if(x\*M[m1]+B[m1]<x\*M[m2]+B[m2]) l=m1; // use > if added in dec. of m

else r=m2;

return x\*M[l]+B[l]; }

**KMP**

int pref[N];

template <typename T>

void prefixarr(T& s) {

int i=1, j=0, n=s.size();

rep(k,n) pref[k] = 0;

while(i < n){

if(s[i]==s[j]) pref[i++]=++j;

else if(j) j=pref[j-1];

else i++; }

template <typename T>

vi kmp(T& s, T t) {

int m=s.size(), n=t.size();

prefixarr(t);

vi pos;

int i=0, j=0;

while(i < m){

if(s[i]==t[j]) i++, j++;

else if(j>0) j=pref[j-1];

else i++;

if(j==n) pos.pb(i-n),j=pref[j-1];}

return pos; }

**BIPARTITE MATCHING**

vi t[N];

int mr[N], mc[N], vis[N], nl, nr;

bool match(int i) {

for(int j: t[i]){

if(!vis[j]){

vis[j]=1;

if(mc[j]<0 or match(mc[j])){

mr[i]=j;

mc[j]=i;

return true;}}}

return false; }

int matching() {

memset(mr, -1, sizeof mr);

memset(mc, -1, sizeof mc);

int ans=0;

rep(i,nl){

memset(vis, 0, sizeof vis);

if(match(i)) ans++;}

return ans; }

**PHI**

catalan numbers:

**GAUSS ELIMINATION**

int DiaReduce(dbl A[N][N\*2], int n){

rep(i,n)rep(j,n) A[i][j+n]=i==j;

rep(k,n){

int i\_max = k;

rep(i,n) if(i>k and abs(A[i][k])> abs(A[i\_max][k])) i\_max = i;

if(abs(A[i\_max][k])<eps) return k;

if(i\_max != k) rep(j,n\*2) swap( A[k][j], A[i\_max][j]);

rep(i,n) if(i!=k){

dbl f=A[i][k]/A[k][k];

rep(j,n\*2) A[i][j]-=A[k][j]\*f;}}

return -1; }

bool Inv(dbl A[N][N\*2], int n) {

if(DiaReduce(A, n)+1) return false;

rep(i,n)rep(j,n)A[i][j+n]/=A[i][i];

return true; }

dbl Det(dbl A[N][N\*2], int n){

if(DiaReduce(A, n)+1) return 0;

dbl ans(1); rep(i,n) ans\*=A[i][i];

return ans; }

bool solve(dbl A[N][N\*2], dbl x[N], dbl B[N], int n) {

// solve Ax = B for unique x only

if(!Inv(A, n)) return false;

rep(i,n){

x[i] = 0;

rep(j,n) x[i]+=A[i][n+j]\*B[j];}}

**FFT**

typedef complex<dbl> cpx;

void fft(vector<cpx>& a, int s=1) {

int n=a.size();

for(int i=1,j=0; i<n; i++){

int bit=n>>1;

for(; j>=bit; bit>>=1) j-=bit;

j+=bit;

if(i<j) swap(a[i], a[j]);}

for(int len=2; len<=n; len<<=1){

dbl ang=2\*s\*pi/len;

cpx wlen(cos(ang), sin(ang));

for(int i=0; i<n; i+=len){

cpx w(1);

rep(j,len/2){

cpx u=a[i+j], v=a[i+j+len/2]\*w;

a[i+j]=u+v;

a[i+j+len/2]=u-v;

w\*=wlen;}}}

if(s==-1) rep(i,n) a[i]/=n; }

template<typename T>

T multiply(T& a, T& b) {

vector<cpx> fa(all(a)), fb(all(b));

int n=1;

while(n<max(a.size(),b.size())) n<<=1; n<<=1;

fa.resize(n), fb.resize(n);

fft(fa), fft(fb);

rep(i,n) fa[i]\*=fb[i];

fft(fa, -1);

T res(n);

rep(i,n) res[i]=(ll)(fa[i].real() +0.5);

return res; }