mt19937 rnd(chrono::system\_clock::now().time\_since\_epoch().count()); //randomize

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

#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>;

oset<pair<int, int> > s;

s.insert({p[i], i});

///s.order\_of\_key(x): count number of s less than x

///\*s.find\_by\_order(k): giá trị k-th sau khi sort lại (bắt đầu từ 0)

F(mask) = sum(A(i)) all i is subset mask -> i & mask = i, maybe change min, max

// iterate over all the masks 3^N

for (int mask = 0; mask < (1 << n); mask++){

F[mask] = A[0];

// iterate over all the subsets of the mask

for(int i = mask; i > 0; i = (i-1) & mask) F[mask] += A[i];

}

//memory optimized, super easy to code. N \* 2^N

for(int i = 0; i < (1 << N); ++i) F[i] = A[i];

for(int i = 0; i < N; ++i)

for(int mask = 0; mask < (1 << N); ++mask){

if(mask & (1 << i)) F[mask] += F[mask ^ (1 << i)];}

void HLD(int u) {

if (chainHead[nChain] == 0) { ///chainHead[x] : đỉnh đầu tiên của đoạn thứ x

chainHead[nChain] = u;

}

chainInd[u] = nChain; ///đoạn mà chứa đỉnh u

Pos[u] = ++Base; ////Thứ tự của đỉnh u khi trải mảng

int dinh = 0;

for (int v : A[u]) {

if (dinh == 0 || con[v] > con[dinh]) dinh = v; ////lấy đỉnh v có số nút con là max

}

if (dinh > 0) HLD(dinh);

for (int v : A[u]) {

if (v != dinh) {nChain++; HLD(v);

}

}

}

**void** **update**(**int** u, **int** a) { /// Tách đường đi từ u đến a: đỉnh u thuộc cây con gốc a

*// uchain chuỗi hiện tại của u*

*// achain chuỗi của a*

**int** uchain **=** chainInd[u], achain **=** chainInd[a];

**while** (1) {

*// Nếu u và a cùng nằm trên một chuỗi thì update đoạn từ u đến a và kết thúc.*

**if** (uchain **==** achain) {

updateIntervalTree(..., posInBase[a], posInBase[u], ...);

**break**;

}

*// Nếu u và a không nằm trên cùng một chuỗi thì update đoạn từ u đến đỉnh đầu của chuỗi hiện tại.*

updateIntervalTree(..., posInBase[chainHead[uchain]], posInBase[u], ...);

*// Nhảy lên đỉnh cha của đỉnh đầu hiện tại.*

u **=** parent[chainHead[uchain]];

uchain **=** chainInd[u];

}

}

void Tach(int u) { /// Tach duong di tu 1 den u

if (u == 0) return;

int pos = chainInd[u];

int root = chainHead[pos];

top++;

L[top] = Pos[root];

R[top] = Pos[u];

Tach(parent[root]);

}

struct CentroidDecomposition { //base 0th, 0 -> n – 1

//Centroid Tree, LCA(A, B) in normal tree = LCA(A, B) in centroid tree

//max height: (logN)

vector<set<int>> tree; // it's not vector<vector<int>>!

vector<int> dad; ///dad[i] is parent node i in centroid tree, dad[root] = root

vector<int> sub;

int n;

CentroidDecomposition(int \_n = 0) {

n = \_n;

tree.resize(n);

dad.resize(n);

sub.resize(n);

}

void addEdge(int u, int v) {tree[u].insert(v); tree[v].insert(u); }

CentroidDecomposition(vector<set<int>> &tree) : tree(tree) {

int n = tree.size();

dad.resize(n);

sub.resize(n);

build(0, -1);

}

void build(int u, int p) { ///build centroid tree

int n = dfs\_subtree(u, p); // find the size of each subtree

int centroid = dfs(u, p, n); // find the centroid

if (p == -1) p = centroid; // dad of root is the root itself

dad[centroid] = p;

// for each tree resulting from the removal of the centroid

for (auto v : tree[centroid]) {

tree[v].erase(centroid), // the component from the tree

build(v, centroid);

}

tree[centroid].clear();

}

int dfs\_subtree(int u, int p) {

sub[u] = 1;

for (auto v : tree[u])

if (v != p) sub[u] += dfs\_subtree(v, u);

return sub[u];

}

int dfs(int u, int p, int n) {

for (auto v : tree[u])

if (v != p && sub[v] > n/2) return dfs(v, u, n);

return u;

}

int operator[](int i) {

return dad[i]; }};

// SPOJ 7826: Tree Isomorphism

#define fst first

#define snd second

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

struct tree {

int n;

vector<vector<int>> adj;

tree(int n) : n(n), adj(n) { }

void add\_edge(int src, int dst) {

adj[src].push\_back(dst);

adj[dst].push\_back(src);

}

vector<int> centers() {

vector<int> prev;

int u = 0;

for (int k = 0; k < 2; ++k) { // double sweep

queue<int> que;

prev.assign(n, -1);

que.push(prev[u] = u);

while (!que.empty()) {

u = que.front(); que.pop();

for (auto v: adj[u]) {

if (prev[v] >= 0) continue;

que.push(v);

prev[v] = u;

}

}

}

vector<int> path = {u}; // median on a path

while (u != prev[u])

path.push\_back(u = prev[u]);

int m = path.size();

if (m % 2 == 0) return {path[m/2-1], path[m/2]};

else return {path[m/2]};

}

vector<vector<int>> layer;

vector<int> prev;

int levelize(int r) { // split vertices into levels

prev.assign(n,-1); prev[r] = n;

layer = {{r}};

while (1) {

vector<int> next;

for (int u: layer.back()) {

for (int v: adj[u]) {

if (prev[v] >= 0) continue;

prev[v] = u;

next.push\_back(v);

}

}

if (next.empty()) break;

layer.push\_back(next);

}

return layer.size();

}

};

bool isomorphic(tree S, int s, tree T, int t) {

if (S.n != T.n) return false;

if (S.levelize(s) != T.levelize(t)) return false;

vector<vector<int>> longcodeS(S.n+1), longcodeT(T.n+1);

vector<int> codeS(S.n), codeT(T.n);

for (int h = S.layer.size()-1; h >= 0; --h) {

map<vector<int>, int> bucket;

for (int u: S.layer[h]) {

sort(all(longcodeS[u]));

bucket[longcodeS[u]] = 0;

}

for (int u: T.layer[h]) {

sort(all(longcodeT[u]));

bucket[longcodeT[u]] = 0;

}

int id = 0;

for (auto &p: bucket) p.snd = id++;

for (int u: S.layer[h]) {

codeS[u] = bucket[longcodeS[u]];

longcodeS[S.prev[u]].push\_back(codeS[u]);

}

for (int u: T.layer[h]) {

codeT[u] = bucket[longcodeT[u]];

longcodeT[T.prev[u]].push\_back(codeT[u]);

}

}

return codeS[s] == codeT[t];

}

bool isomorphic(tree S, tree T) {

auto x = S.centers(), y = T.centers();

if (x.size() != y.size()) return false;

if (isomorphic(S, x[0], T, y[0])) return true;

return x.size() > 1 && isomorphic(S, x[1], T, y[0]);

}

void doit() {

int n; scanf("%d", &n);

tree S(n), T(n);

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

int u, v; scanf("%d %d", &u, &v);

S.add\_edge(u-1, v-1);

}

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

int u, v; scanf("%d %d", &u, &v);

T.add\_edge(u-1, v-1);

}

if (isomorphic(S, T)) printf("YES\n");

else printf("NO\n");}

#define MCMF MincostMaxflow

#define flow\_t int

#define cost\_t int

const flow\_t foo = (flow\_t) 1e9;

const cost\_t coo = (cost\_t) 1e9;

namespace MincostMaxflow {

const int maxv = 1e5 + 5;

const int maxe = 1e6 + 5;

int n, s, t, E;

int adj[maxe], nxt[maxe], lst[maxv], frm[maxv], vis[maxv];

flow\_t cap[maxe], flw[maxe], totalFlow;

cost\_t cst[maxe], dst[maxv], totalCost;

void init(int nn, int ss, int tt) {

n = nn, s = ss, t = tt;

fill\_n(lst, n + 7, -1), E = 0;

}

void add(int u, int v, flow\_t ca, cost\_t co) {

adj[E] = v, cap[E] = ca, flw[E] = 0, cst[E] = +co, nxt[E] = lst[u], lst[u] = E++;

adj[E] = u, cap[E] = 0, flw[E] = 0, cst[E] = -co, nxt[E] = lst[v], lst[v] = E++;

}

int spfa() {

fill\_n(dst, n + 7, coo), dst[s] = 0;

queue<int> que; que.push(s);

while (que.size()) {

int u = que.front(); que.pop();

for (int e = lst[u]; e != -1; e = nxt[e]) if (flw[e] < cap[e]) {

int v = adj[e];

if (dst[v] > dst[u] + cst[e]) {

dst[v] = dst[u] + cst[e]; frm[v] = e;

if (!vis[v]) {vis[v] = 1; que.push(v);}

}

}

vis[u] = 0;

}

return dst[t] < coo;

}

pair<flow\_t, cost\_t> mincost() {

totalCost = 0, totalFlow = 0;

while (1) {

if (!spfa()) break;

flow\_t mn = foo;

for (int v = t, e = frm[v]; v != s; v = adj[e ^ 1], e = frm[v]) mn = min(mn, cap[e] - flw[e]);

for (int v = t, e = frm[v]; v != s; v = adj[e ^ 1], e = frm[v]) {flw[e] += mn; flw[e ^ 1] -= mn;}

totalFlow += mn;

totalCost += mn \* dst[t];

}

return {totalFlow, totalCost};

}

}

//Cặp ghép cực đại có cost cực tiểu

template<class T, T oo> struct Hungary { //index from 1

static const int MAXN = 1000 + 5;

static const int MAXM = 1000 + 5;

int nx, ny, maty[MAXM], frm[MAXM], used[MAXM];

T cst[MAXN][MAXM], fx[MAXN], fy[MAXM], dst[MAXM];

void init(int nx, int ny) {

this->nx = nx, this->ny = ny;

fill\_n(fx, nx + 1, 0), fill\_n(fy, ny + 1, 0);

fill\_n(maty, nx + 1, 0);

for (int i = 0; i <= nx; i++) fill\_n(cst[i], ny + 1, oo);

}

void add(int x, int y, int c) {cst[x][y] = c; } ///may be min(cst[x][y], c);

T mincost() {

for (int x = 1; x <= nx; x++) {

int y0 = 0;

maty[0] = x;

for (int y = 0; y <= ny; y++) {

dst[y] = oo + 1;

used[y] = 0;

}

do {

used[y0] = 1;

int x0 = maty[y0], y1;

T delta = oo + 1;

for (int y = 1; y <= ny; y++) if (!used[y]) {

T curdst = cst[x0][y] - fx[x0] - fy[y];

if (dst[y] > curdst) {

dst[y] = curdst;

frm[y] = y0;

}

if (delta > dst[y]) {

delta = dst[y];

y1 = y;

}

}

for (int y = 0; y <= ny; y++) if (used[y]) {

fx[maty[y]] += delta;

fy[y] -= delta;

}

else {

dst[y] -= delta;

}

y0 = y1;

}

while (maty[y0] != 0);

do {

int y1 = frm[y0];

maty[y0] = maty[y1];

y0 = y1;

}

while (y0);

}

T res = 0;

for (int y = 1; y <= ny; y++) {

int x = maty[y];

if (cst[x][y] < oo) res += cst[x][y];

}

return res;

}

};

Hungary<int, (int) 1e9> hungary;

hungary.init(3, 3);

hungary.add(1, 2, 3);

hungary.add(1, 2, 1);

cout << hungary.mincost() << "\n";

\* Cặp ghép cực đại trên đồ thị vô hướng

\* Complexity: O(E\*sqrt(V))

\* Indexing from 1

#define BS Blossom

namespace Blossom {

const int maxv = 1e3 + 5;

const int maxe = 1e6 + 5;

int n, E, lst[maxv], next[maxe], adj[maxe];

int nxt[maxv], mat[maxv], dad[maxv], col[maxv];

int que[maxv], qh, qt;

int vis[maxv], act[maxv];

int tag, total;

void init(int \_n) {

n = \_n;

for (int i = 0; i <= n; i++) lst[i] = nxt[i] = mat[i] = vis[i] = 0;

E = 1, tag = total = 0;

}

void add(int u,int v) {

if (!mat[u] && !mat[v]) mat[u] = v, mat[v] = u, total++;

E++, adj[E] = v, next[E] = lst[u], lst[u] = E;

E++, adj[E] = u, next[E] = lst[v], lst[v] = E;

}

int lca(int u, int v) {

tag++;

for(; ; swap(u, v)) {

if (u) {

if (vis[u = dad[u]] == tag) return u;

vis[u] = tag;

u = nxt[mat[u]];

}

}

}

void blossom(int u, int v, int g) {

while (dad[u] != g) {

nxt[u] = v;

if (col[mat[u]] == 2) {

col[mat[u]] = 1;

que[++qt] = mat[u];

}

if (u == dad[u]) dad[u] = g;

if (mat[u] == dad[mat[u]]) dad[mat[u]] = g;

v = mat[u];

u = nxt[v];

}

}

int augument(int s) {

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

col[i] = 0;

dad[i] = i;

}

qh = 0; que[qt = 1] = s; col[s] = 1;

for (int u, v, i; qh < qt; ) {

act[u = que[++qh]] = 1;

for (i = lst[u]; i ; i = next[i]) {

v = adj[i];

if (col[v] == 0) {

nxt[v] = u;

col[v] = 2;

if (!mat[v]) {

for (; v; v = u) {

u = mat[nxt[v]];

mat[v] = nxt[v];

mat[nxt[v]] = v;

}

return 1;

}

col[mat[v]] = 1;

que[++qt] = mat[v];

}

else if (dad[u] != dad[v] && col[v] == 1) {

int g = lca(u, v);

blossom(u, v, g);

blossom(v, u, g);

for (int j = 1; j <= n; j++) dad[j] = dad[dad[j]];

}

}

}

return 0;

}

int maxmat() {

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

if (!mat[i]) total += augument(i);

}

return total;

}

}

BS::init(10);

for (int i = 1; i <= 10; i++) {

int j = rnd() % 10 + 1;

BS::add(i, j);

cout << i << " " << j << '\n';

}

cout << BS::maxmat() << '\n';

for (int i = 1; i <= BS::n; i++) {

BS::act[i] = 0;

}

BS::maxmat();

//BS::act[i] = 1 iff the number of matching doesn't change if remove vertex i

for (int i = 1; i <= BS::n; i++) {

cout << i << " " << BS::act[i] << "\n";

}

/\* Numbered from 0

\* For man i, L[i] = list of women in order of decreasing preference

\* For women j, R[j][i] = index of man i in j-th women's list of preference

\* OUTPUTS:

\* - L2R[]: the mate of man i (always between 0 and n-1)

\* - R2L[]: the mate of woman j (or -1 if single)

\* COMPLEXITY: M^2

\*/

#define MAXM 1024

#define MAXW 1024

int m;

int L[MAXM][MAXW], R[MAXW][MAXM];

int L2R[MAXM], R2L[MAXW];

int p[MAXM]; //degree in traverse

void stableMarriage(){

static int p[128];

memset(R2L, -1, sizeof R2L);

memset(p, 0, sizeof p);

// Each man proposes...

for(int i = 0; i < m; i++) {

int man = i;

while (man >= 0) { // propose until success

int wom;

while (1) {

wom = L[man][p[man]++];

if (R2L[wom] < 0 || R[wom][man] > R[wom][R2L[wom]]) break;

}

int hubby = R2L[wom];

R2L[L2R[man] = wom] = man;

man = hubby; // remarry the dumped guy

}

}

}

Tìm tập phủ đỉnh cực tiểu trên đồ thị hai phía.

Có thể hiểu đề bài rằng: Trong một tập đỉnh, một đỉnh nằm trong tập thì phủ tất cả các cạnh kề với nó. Cần chọn tập có ít đỉnh nhất sao cho tất cả các cạnh đều bị phủ.

#define N 2003

int m, n, E;

vector<int> a[N]; //

int Assigned[N], Visited[N]; //

bool Choosed[N]; //

bool visit(int u, int Key) {

if (Visited[u] == Key)

return false;

Visited[u] = Key;

for (int v : a[u])

if (!Assigned[v] || visit(Assigned[v], Key)) {

Assigned[u] = v; Assigned[v] = u;

return true;

}

return false;

}

void konig() {

queue<int> qu;

for (int i = 1; i <= m; ++i) if (!Assigned[i]) qu.push(i);

for (int i = 1; i <= n; ++i) if (!Assigned[N - i]) qu.push(N - i);

while (qu.size()) {

int u = qu.front();

qu.pop();

for (int v : a[u])

if (!Choosed[v]) {

Choosed[v] = true;

qu.push(Assigned[v]);

}

}

for (int i = 1; i <= m; ++i) if (Assigned[i] && !Choosed[i] && !Choosed[Assigned[i]])

Choosed[i] = true;

}

int main() {

scanf("%d%d%d", &m, &n, &E);

for (int i = 1; i <= E; ++i) {

int x, y;

scanf("%d%d", &x, &y);

a[x].push\_back(N - y);

a[N - y].push\_back(x);

}

static int cnt = 0;

int Count = 0;

for (int i = 1; i <= m; ++i) if (!Assigned[i]) visit(i, ++cnt);

for (int i = 1; i <= m; ++i) if (Assigned[i]) Count++;

cout << Count;

konig();

for (int i = 1; i <= m; ++i) if (Choosed[i]) printf("%d ", i);

for (int i = 1; i <= n; ++i) if (Choosed[N - i]) printf("%d ", i);

printf("\n");

}

Sau khi chạy bộ ghép cực đại, ta thực hiện thuật toán Konig như sau: Đưa và queue tất cả các đỉnh chưa được ghép. Với mỗi phần tử u trong queue, ta xét tất cả cạnh (u, v), chọn v vào tập kết quả và đưa Assigned[v] vào queue. Với tất cả các cạnh nằm trong bộ ghép mà hai mút của cạnh chưa được đưa vào tập thì chọn tất cả các đỉnh bên trái (hoặc bên phải) vào tập.

// n = 2 \* (number of boolean variables)

// NOTE: if we need to fix some variable, e.g. set i = 0 --> addEdge(2\*i+1, 2\*i)

// var i --> 2 nodes: 2\*i, 2\*i+1.

//addEdge(u, v): g[u].push\_back(v), gt[v].push\_back(u)

int n;

vector<int> g[maxN], gt[maxN];

vector<bool> used;

vector<int> order, comp;

vector<bool> assignment;

void dfs1(int v) {

used[v] = true;

for (int u : g[v]) {

if (!used[u])

dfs1(u);

}

order.push\_back(v); //topo sort

}

void dfs2(int v, int cl) {

comp[v] = cl;

for (int u : gt[v]) {

if (comp[u] == -1)

dfs2(u, cl);

}

}

bool solve\_2SAT() {

used.assign(n, false);

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

if (!used[i])

dfs1(i);

}

comp.assign(n, -1);

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

int v = order[n - i - 1];

if (comp[v] == -1)

dfs2(v, j++);

}

assignment.assign(n / 2, false);

for (int i = 0; i < n; i += 2) {

if (comp[i] == comp[i + 1])

return false;

assignment[i / 2] = comp[i] > comp[i + 1];

}

return true;

}

struct Line { ///IT Line: Max y = ax + b;

///y = a \* x + b;

long long a, b;

Line(){};

Line(long long \_a, long long \_b) {a = \_a;b = \_b;}

};

Line IT[maxN \* 4];

vector<long long> realX; ///maybe not used

long long get(Line L, int x) {

if (L.a == 0 && L.b == 0) return -INF\_LL;

return L.a \* realX[x] + L.b;

}

void update(int i, int L, int R, int u, int v, Line val) {

if (L > v || R < u) return;

int mid = (L + R) >> 1;

if (L >= u && R <= v) {

if (get(IT[i], L) >= get(val, L) && get(IT[i], R) >= get(val, R)) return;

if (get(IT[i], L) <= get(val, L) && get(IT[i], R) <= get(val, R)) {IT[i] = val;return;}

if (get(IT[i], L) >= get(val, L) && get(IT[i], mid) >= get(val, mid)) {update(i << 1 | 1, mid + 1, R, u, v, val);return;}

if (get(IT[i], L) <= get(val, L) && get(IT[i], mid) <= get(val, mid)) {

update(i << 1 | 1, mid + 1, R, u, v, IT[i]); IT[i] = val; return; }

if (get(IT[i], mid + 1) >= get(val, mid + 1) && get(IT[i], R) >= get(val, R)) {

update(i << 1, L, mid, u, v, val); return; }

if (get(IT[i], mid + 1) <= get(val, mid + 1) && get(IT[i], R) <= get(val, R)) {

update(i << 1, L, mid, u, v, IT[i]); IT[i] = val; return; }

}

update(i << 1, L, mid, u, v, val);

update(i << 1 | 1, mid + 1, R, u, v, val);

}

long long get(int i, int L, int R, int x) {

if (L > x || R < x) return -INF\_LL;

if (L >= x && R <= x) return get(IT[i], x);

int mid = (L + R) >> 1;

long long left = get(i << 1, L, mid, x);

long long right = get(i << 1 | 1, mid + 1, R, x);

return max(max(left,right), get(IT[i], x)); }

const int BLOCK = 300; //sqrt(N); //Mo’s Algorithm

struct query {

int L, R, id;

query(){};

bool operator < (const query &A) {

if (L / BLOCK != A.L / BLOCK) {

return L / BLOCK < A.L / BLOCK;

}

return R < A.R;

}

};

void add(int x) {}//do something

void del(int x) {}//do something

query q[maxN];

int res[maxN];

void sol() {

int n, m;

cin >> n >> m;

for (int i = 1; i <= n; i++) cin >> a[i];

for (int i = 1; i <= m; i++) {

cin >> q[i].L >> q[i].R;

q[i].id = i;

}

sort(q + 1, q + 1 + m);

int L = 1, R = 0;

for (int i = 1; i <= m; i++) {

while (R < q[i].R) add(a[++R]);

while (R > q[i].R) del(a[R--]);

while (L < q[i].L) del(a[L++]);

while (L > q[i].L) add(a[--L]);

res[q[i].id] = ?; //get result query q[i].L -> q[i].R;

}

for (int i = 1; i <= m; i++) cout << res[i] << '\n';

}

template<typename num\_t>

struct RangeFenwick2D {

int n, m;

vector<vector<num\_t> > fen[4];

RangeFenwick2D(int n, int m) {

this->n = n, this->m = m;

for (int i = 0; i < 4; i++) fen[i].resize(n + 1, vector<num\_t>(m + 1));

}

void upd(int x, int y, num\_t val) {

//x++, y++; //if base 0th -> increase

assert(0 < x && 0 < y);

for (int xx = x; xx <= n; xx += xx & -xx) {

for (int yy = y; yy <= m; yy += yy & -yy) {

fen[0][xx][yy] += val;

fen[1][xx][yy] += val \* x;

fen[2][xx][yy] += val \* y;

fen[3][xx][yy] += val \* x \* y;

}

}

}

void upd(int x, int y, int z, int t, num\_t val) {

upd(x, y, +val);

upd(x, t + 1, -val);

upd(z + 1, y, -val);

upd(z + 1, t + 1, +val);

}

num\_t query(int x, int y) {

//x++, y++; //if base 0th -> increase

assert(x <= n && y <= m);

num\_t res = 0;

for (int xx = x; xx > 0; xx -= xx & -xx) {

for (int yy = y; yy > 0; yy -= yy & -yy) {

res += fen[0][xx][yy] \* (x + 1) \* (y + 1);

res -= fen[1][xx][yy] \* (y + 1);

res -= fen[2][xx][yy] \* (x + 1);

res += fen[3][xx][yy];

}

}

return res;

}

num\_t query(int x, int y, int z, int t) {

return query(z, t) - query(x - 1, t) - query(z, y - 1) + query(x - 1, y - 1);

}

};

const int MAXN = 3e6 + 5; //Persitent Segment Tree

int ptr;

struct Node {

Node \*l, \*r;

int L, R, cnt;

Node();

} mem[MAXN], \*nil = mem + MAXN - 1;

Node::Node() {

l = r = nil;

L = R = -1;

cnt = 0;

}

Node\* newNode() {

return mem + (ptr++);

}

Node\* build(int L, int R) {

Node\* node = newNode();

node->L = L; node->R = R;

if (L == R) return node;

node->l = build(L, (L + R) >> 1);

node->r = build(((L + R) >> 1) + 1, R);

return node;

}

Node\* update(Node\* node, int i, int val) { //a(i) += val

if (node->L > i || node->R < i) return node;

Node\* x = newNode();

x->L = node->L; x->R = node->R;

x->l = node->l; x->r = node->r;

if (node->L == node->R) {

x->cnt = node->cnt + val;

return x;

}

x->l = update(x->l, i, val);

x->r = update(x->r, i, val);

x->cnt = x->l->cnt + x->r->cnt;

return x;

}

int query(Node\* node, int L, int R) {

if (node->L > R || node->R < L) return 0;

if (node->L >= L && node->R <= R) return node->cnt;

return query(node->l, L, R) + query(node->r, L, R);

}

Node\* root[MAXN];

int n = 10;

root[0] = build(0, n - 1);

root[1] = update(root[0], 2, 4);

root[2] = update(root[1], 3, 1);

cout << query(root[2], 0, 3) << "\n"; //Expected 5

struct Node { ///IT 2D

int res;

Node \*Left,\*Right;

Node (int x) {

res = 0;

Left = NULL;

Right = NULL;

}

}\*IT[maxN \* 4];

void update\_y(Node \*&IT, int L, int R, int y, int val) {

if (L > y || R < y) return;

if (IT == NULL) IT = new Node(0);

IT->res += val;

if (L == R) return;

int mid = (L + R) >> 1;

update\_y(IT->Left, L, mid, y, val);

update\_y(IT->Right, mid + 1, R, y, val);

}

void update(int i, int L, int R, int x, int y, int val) { ///cập nhật điểm (x,y)

if (L > x || R < x) return;

update\_y(IT[i], 1, 10000, y, val);

if (L == R) return;

int mid = (L + R) >> 1;

update(i << 1, L, mid, x, y, val);

update(i << 1 | 1, mid + 1, R, x, y, val);

}

int Get\_y(Node \*IT, int L, int R, int yL, int yR) {

if (L > yR || R < yL) return 0;

if (IT == NULL) return 0;

if (R >= yL && R <= yR) return IT->res;

int mid = (L + R) >> 1;

int left = Get\_y(IT->Left, L, mid, yL, yR);

int right = Get\_y(IT->Right, mid + 1, R, yL, yR);

return left + right;

}

int Get(int i, int L, int R, int xL, int xR, int yL, int yR) { /// Lấy bảng (xL, yL) -> (xR, yR)

if (L > xR || R < xL) return 0;

if (L >= xL && R <= xR) return Get\_y(IT[i], 1, 10000, yL, yR);

int mid = (L + R) >> 1;

int left = Get(i << 1, L, mid, xL, xR, yL, yR);

int right = Get(i << 1 | 1, mid + 1, R, xL, xR, yL, yR);

return left + right;

}

int min\_rotation(string s) {

int a = 0, N = (int)s.size();

for (int b = 0; b < N; b++) {

for (int i = 0; i < N; i++) {

if (a + i == b || s[(a + i) % N] < s[(b + i) % N]) {b += max(0, i - 1); break;}

if (s[(a + i) % N] > s[(b + i) % N]) {a = b; break;}

}

}

return a;

};

struct SuffixArray {

const int L;

string s;

vector<vector<int> > P;

vector<pair<pair<int,int>,int> > M;

///P[k][i] = P[k][j] neu xau bat dau tu i co do dai 2^k = xau bat dau tu j co do dai 2^k

SuffixArray(const string &s) : L(s.length()), s(s), P(1, vector<int>(L, 0)), M(L) {

for (int i = 0; i < L; i++) P[0][i] = int(s[i]);

for (int skip = 1, level = 1; skip < L; skip \*= 2, level++) {

P.push\_back(vector<int>(L, 0));

for (int i = 0; i < L; i++) {

M[i] = make\_pair(make\_pair(P[level-1][i], i + skip < L ? P[level-1][i + skip] : -1000), i);}

sort(M.begin(), M.end());

for (int i = 0; i < L; i++) {

P[level][M[i].second] = (i > 0 && M[i].first == M[i-1].first) ? P[level][M[i-1].second] : i;}

}

}

vector<int> GetSuffixArray() { return P.back(); }

// returns the length of the longest common prefix of s[i...L-1] and s[j...L-1]

int LongestCommonPrefix(int i, int j) {

int len = 0;

if (i == j) return L - i;

for (int k = P.size() - 1; k >= 0 && i < L && j < L; k--) {

if (P[k][i] == P[k][j]) {

i += 1 << k;

j += 1 << k;

len += 1 << k;

}

}

return len;

}

};

struct Manacher {

string s;

int n;

vector<int> D1, D2;

Manacher(){};

Manacher(string \_s) {

s = \_s;

n = (int)s.size();

D1 = vector<int>(n, 0); //i-D1[i]+1 -> i+D1[i]-1

D2 = vector<int>(n, 0); //i-D2[i]->i+D2[i]-1

}

void calc() {

for (int i = 0, L = 0, R = -1; i < n; i++) {

int k = (i > R) ? 1 : min(D1[L + R - i], R - i + 1);

while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) k++;

D1[i] = k--;

if (i + k > R) {L = i - k; R = i + k;}

}

for (int i = 0, L = 0, R = -1; i < n; i++) {

int k = (i > R) ? 0 : min(D2[L + R - i + 1], R - i + 1);

while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {

k++;

}

D2[i] = k--;

if (i + k > R) {L = i - k - 1; R = i + k;}

}

}

int getLenPalindrome(int root) {

return max(D1[root] \* 2 - 1, D2[root] \* 2);

}

};

vector<bool> isPrime(R - L + 1, true); // x là số nguyên tố khi và chỉ khi isPrime[x - l] == true

for (long long i = 2; i \* i <= R; ++i) {

for (long long j = max(i \* i, (L + i - 1) / i \* i); j <= R; j += i) {

isPrime[j - L] = false;

}

}

if (1 >= L) { // Xét riêng trường hợp số 1

isPrime[1 - L] = false;

}

for (long long x = L; x <= R; ++x) {

if (isPrime[x - L]) {

// i là số nguyên tố

}

}

O(sqrt(R) \* k), k = const

using u64 = uint64\_t;

using u128 = \_\_uint128\_t;

bool check\_composite(u64 n, u64 a, u64 d, int s) {

u64 x = power\_mod(a, d, n); //calc a^d % n

if (x == 1 || x == n - 1) return false;

for (int r = 1; r < s; r++) {

x = (u128)x \* x % n;

if (x == n - 1) return false;

}

return true;

};

bool MillerRabin(u64 n, int iter=5) { // returns true if n is probably prime, else returns false.

if (n < 4)

return n == 2 || n == 3;

int s = 0;

u64 d = n - 1;

while ((d & 1) == 0) {

d >>= 1;

s++;

}

for (int i = 0; i < iter; i++) {

u64 a = 2 + rand() % (n - 3);

if (check\_composite(n, a, d, s))

return false;

}

return true;

}

long long F(int p, int q, int n) { ///Tinh tổng floor(p \* i / q) với i từ 1 đến n và p < q

if (p == 0) return 0;

long long res = n / (q / \_\_gcd(p, q));

long long val = 1LL \* n \* p / q;

res += val \* n;

res -= val \* (val + 1) / 2 \* (q / p);

return res - F(q % p, p, val);

}

// sinh bo 3 pytago nguyen thuy voi x, y, z <= n

vector< vector<int> > genPrimitivePytTriples(int n) {

vector< vector<int> > ret;

for (int r=1; r\*r<=n; ++r) for (int s=(r%2==0)?1:2; s<r; s+=2) if (\_\_gcd(r,s)==1) {

vector<int> t;

t.push\_back(r\*r+s\*s); //z

t.push\_back(2\*r\*s); // y

t.push\_back(r\*r-s\*s); // x

if (t[0]<=n) ret.push\_back(t);

}

sort(ret.begin(), ret.end());

return ret;

}

// a^2 + b^2 == c^2

// To generate all primitive triples:

// a = m^2 - n^2, b = 2mn, c = m^2 + n^2 (m > n)

// Primitive triples iff gcd(m, n) == 1 && (m - n) % 2 == 1

double max\_f(double left, double right) {/// Chặt tam phân

int N\_ITER = 100;

for (int i = 0; i < N\_ITER; i++) {

double x1 = left + (right - left) / 3.0;

double x2 = right - (right - left) / 3.0;

if (f(x1) > f(x2)) right = x2;

else left = x1;

}

return f(left);

}

Phi hàm euler (n) : số số không âm nguyên tố cùng nhau với n: Ký hiệu: phi(n)

**int** **eulerPhi**(**int** n) { *// = n (1-1/p1) ... (1-1/pn) = tích: ({p\_i} – 1) \* {p\_i}^({k\_i} – 1)*

*n = p1^k1 \* p2^k2 \* … \* pr^kr*

**if** (n **==** 0) **return** 0;

**int** ans **=** n;

**for** (**int** x **=** 2; x**\***x **<=** n; **++**x) {

**if** (n **%** x **==** 0) {

ans **-=** ans **/** x;

**while** (n **%** x **==** 0) n **/=** x;

}}

**int extgcd(int a, int b, int &x, int &y) {** => trả về cặp (x,y) sao cho thỏa Ax + By = GCD(A,B)

**int g = a; x = 1; y = 0;**

**if (b != 0) g = extgcd(b, a % b, y, x), y -= 1LL \* (a / b) \* x;**

**return g; //gcd(a, b)**

**}**

Để giải Ax + By = C (Điều kiện C % GCD(A,B) = 0)

Tìm (x,y) sao cho Ax + By = GCD(A,B)

x = x \* C / GCD(A,B), y = y \* C / GCD(A,B)

Vậy ta có 1 nghiệm Ax + By = C

Gọi P / Q = B / A

Nghiệm dạng tổng quát là (x + k \* P, y – k \* Q), k thuộc Z

// Returns minimum x for which a ^ x % m = b % m.

// a and m maybe no co-prime -> gcd(a, m) >= 1

// b % gcd(a, m) != 0 -> no solution

// Complexity O(sqrt(m))

int solve(int a, int b, int m) {

a %= m, b %= m;

int k = 1, add = 0, g;

while ((g = \_\_gcd(a, m)) > 1) {

if (b == k)

return add;

if (b % g)

return -1;

b /= g, m /= g, ++add;

k = (k \* 1ll \* a / g) % m;

}

int n = sqrt(m) + 1;

int an = 1;

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

an = (an \* 1ll \* a) % m;

unordered\_map<int, int> vals;

for (int q = 0, cur = b; q <= n; ++q) {

vals[cur] = q;

cur = (cur \* 1ll \* a) % m;

}

for (int p = 1, cur = k; p <= n; ++p) {

cur = (cur \* 1ll \* an) % m;

if (vals.count(cur)) {

int ans = n \* p - vals[cur] + add;

return ans;

}

}

return -1;

}

// Jacobi Symbol (m/n), m,n≥0 and n is odd

// (m/n)==1 x^2 == m (mod n) solvable, -1 unsolvable

#define NEGPOW(e) ((e) % 2 ? -1 : 1)

int jacobi(int a, int m) {

if (a == 0) return m == 1 ? 1 : 0;

if (a % 2) return NEGPOW(1LL \* (a - 1) \* (m - 1) / 4) \* jacobi(m % a, a);

else return NEGPOW((1LL \* m \* m - 1) / 8) \* jacobi(a / 2, m);

}

int extgcd(int a, int b, int &x, int &y) {

int g = a; x = 1; y = 0;

if (b != 0) g = extgcd(b, a % b, y, x), y -= 1LL \* (a / b) \* x;

return g; ///gcd(a, b)

}

int invMod(int a, int m) { //calc (1 / a) % m

int x, y; //extgcd = extended Euclid

if (extgcd(a, m, x, y) == 1) return (x % m + m) % m;

else return 0; // unsolvable

}

// No solution when: n^((p-1)/2) = (p - 1) <=> (-1) mod p

// powMod(a, n, p): (a^n) mod p

int sqrtMod(int n, int p) { //find x: x2 = n (mod p) p is prime

if (powMod(n, (p - 1) / 2, p) != 1) return -1;

int S, Q, W, i, m = invMod(n, p);

if (m == 0) return -1;

if (p % 4 == 3) return powMod(n, (p + 1) / 4, p);

for (Q = p - 1, S = 0; Q % 2 == 0; Q /= 2, ++S);

do { W = rand() % p; } while (W == 0 || jacobi(W, p) != -1);

for (int R = powMod(n, (Q + 1) / 2, p), V = powMod(W, Q, p); ; ) {

int z = (1LL \* R \* R % p) \* m % p;

for (i = 0; i < S && z % p != 1; z = 1LL \* z \* z % p, ++i);

if (i == 0) return R;

R = 1LL \* R \* powMod(V, 1LL << (S - i - 1), p) % p;

}

}

// Primitive root of modulo n is integer g iff for all a < n & gcd(a, n) == 1, there exist k: g^k = a mod n

// k is called discrete log of a (in case P is prime, can find in O(sqrt(P)) by noting that (P-1) is divisible by k)

// Exist if:

// - n is 1, 2, 4

// - n = p^k for odd prime p

// - n = 2\*p^k for odd prime p

int generator (int p) {

vector<int> fact;

int phi = p-1, n = phi;

for (int i=2; i\*i<=n; ++i)

if (n % i == 0) {

fact.push\_back (i);

while (n % i == 0) n /= i;

}

if (n > 1) fact.push\_back (n);

for (int res=2; res<=p; ++res) {

bool ok = true;

for (size\_t i=0; i<fact.size() && ok; ++i)

ok &= powmod (res, phi / fact[i], p) != 1;

if (ok) return res;

}

return -1;

}

// Giải phương trình: a1x1 + a2x2 + … + anxn ≡ b (module m)

// Trong đó a1, a2, …, an, b, m là các số nguyên dương.

int g[maxN], x[maxN];

bool congruenceEquation(vector<int> a, int b, int m, vector<int> &ret) {

int n = (int)a.size();

a.push\_back(m);

g[0] = a[0];

for (int i = 1; i <= n; i++) g[i] = \_\_gcd(g[i - 1], a[i]);

ret.clear();

if (b % g[n]) return false;

int val = b / g[n];

for (int i = n; i >= 1; i--) {

int xx, yy;

extgcd(g[i - 1], a[i], xx, yy);

x[i] = 1LL \* yy \* val % m;

val = 1LL \* xx \* val % m;

}

x[0] = val;

for (int i = 0; i <= n; i++) x[i] = (x[i] % m + m) % m;

for (int i = 0; i < n; i++) ret.push\_back(x[i]);

return true;

}

const double EPS = 1e-6;

struct Result {

int n; // Number of solutions

double x[3]; // Solutions};

Result solve\_cubic(double a, double b, double c, double d) {

long double a1 = b/a, a2 = c/a, a3 = d/a;

long double q = (a1\*a1 - 3\*a2)/9.0, sq = -2\*sqrt(q);

long double r = (2\*a1\*a1\*a1 - 9\*a1\*a2 + 27\*a3)/54.0;

double z = r\*r-q\*q\*q, theta;

Result s;

if(z <= EPS) {

s.n = 3; theta = acos(r/sqrt(q\*q\*q));

s.x[0] = sq\*cos(theta/3.0) - a1/3.0;

s.x[1] = sq\*cos((theta+2.0\*PI)/3.0) - a1/3.0;

s.x[2] = sq\*cos((theta+4.0\*PI)/3.0) - a1/3.0;

}

else {

s.n = 1; s.x[0] = pow(sqrt(z)+fabs(r),1/3.0);

s.x[0] += q/s.x[0]; s.x[0] \*= (r < 0) ? 1 : -1;

s.x[0] -= a1/3.0;}return s; }

vector<int> divisors(int p) {

vector<int> res;

for (int i = 1; i \* i <= p; i++) {

if (p % i == 0) {

res.push\_back(i);

res.push\_back(p / i);

}

}

sort(res.begin(), res.end());

res.resize(unique(res.begin(), res.end()) - res.begin());

return res;

}

bool isPower(int n, int p) {

while (n % p == 0) n /= p;

return n == 1;

}

long long pisano\_prime(int p) {

if (isPower(p, 2)) return 1LL \* p \* 3 / 2; ///2^k

if (isPower(p, 5)) return 1LL \* p \* 4; ///5^k

//F(n, p) = Fibo(n) % p

//Fibo(0) = 0, Fibo(1) = 1

if (p % 5 == 1 || p % 5 == 4) {

//pisano\_prime(p) là ước của (p - 1)

vector<int> divisor = divisors(p - 1);

for (int n : divisor) {

long long c = F(n, p);

long long d = F(n + 1, p);

if (c == 0 && d == 1) return n;

}

}

else {

//pinano\_prime(p) là ước của (2 \* p + 2)

vector<int> divisor = divisors(2 \* p + 2);

for (int n : divisor) {

long long c = F(n, p);

long long d = F(n + 1, p);

if (c == 0 && d == 1) return n;

}

}

return 0;

}

long long pisano(int n) {

//n = p1^w1 \* p2^w2 \* ... \* pk^wk

//a(i) = p\_i^({w\_i} - 1) \* pisano\_prime(p\_i)

//pisano(n) = lcm(a(1), a(2), ..., a(k))

vector<pair<int, int> > primes;

int x = n;

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

if (n % i == 0) {

int d = 0;

while (n % i == 0) {

n /= i;

d++;

}

primes.push\_back({i, d}); } }

if (n > 1) primes.push\_back({n, 1});

long long res = 1;

for (auto it : primes) {

int p = it.first;

int w = it.second;

long long cost = 1;

for (int i = 1; i < w; i++) cost \*= p;

cost \*= pisano\_prime(p);

res = lcm(res, cost);

}

return res;

}

typedef int num\_t;

namespace CRT {

num\_t res = 0;

num\_t prd = 1;

void clear() {res = 0, prd = 1;}

num\_t mul(num\_t a, num\_t b, num\_t p) {

a %= p, b %= p;

num\_t q = (num\_t) ((long double) a \* b / p);

num\_t r = a \* b - q \* p;

while (r < 0) r += p;

while (r >= p) r -= p;

return r;

}

template<typename num\_t>

pair<num\_t, num\_t> euclid(num\_t a, num\_t b) {

if (!b) return make\_pair(1, 0);

pair<num\_t, num\_t> r = euclid(b, a % b);

return make\_pair(r.second, r.first - a / b \* r.second);

}

void add(num\_t p, num\_t r) {

res += mul(r - res % p + p, euclid(prd, p).first + p, p) \* prd;

prd \*= p;

if (res >= prd) res -= prd;

}

}

int p[] = {2, 3, 5};

int r[] = {1, 2, 3};

CRT::clear();

for (int i = 0; i < 3; i++) CRT::add(p[i], r[i]);

for (int i = 0; i < 3; i++) assert(CRT::res % p[i] == r[i]);

const int mod = 998244353; //FFT

const int root = 1000;

const int root\_pw = 1 << 18;

const int prr = 10;

//typedef complex<double> base;

typedef int base;

void fft(vector<base> &a, bool inv = false){

int n = a.size(), j = 0;

vector<base> roots(n / 2);

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

int bit = (n >> 1);

while (j >= bit){ j -= bit; bit >>= 1; }

j += bit;

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

}

// double ang = 2 \* acos(-1) / n \* (inv ? -1 : 1);

// for (int i = 0; i < n / 2; ++i) roots[i] = base(cos(ang \* i), sin(ang \* i));

int ang = Pow(prr, (mod - 1) / n, mod);

if (inv) ang = Pow(ang, mod - 2, mod);

roots[0] = 1;

for (int i = 1; i < n / 2; ++i) roots[i] = 1LL \* roots[i - 1] \* ang % mod;

/\*

In NTT, let prr = primitive root. Then,

int ang = ipow(prr, (mod - 1) / n);

if(inv) ang = ipow(ang, mod - 2);

for(int i=0; i<n/2; i++) roots[i] = (i ? (1ll \* roots[i-1] \* ang % mod) : 1);

Others are same. If there is /= n, do \*= ipow(n, mod - 2).

In XOR convolution, roots[\*] = 1.

\*/

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

//// int wlen = ang;

//// for (int i = len; i < root\_pw; i <<= 1)

//// wlen = int (wlen \* 1ll \* wlen % mod);

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

//// int w = 1;

//// for (int j = 0; j < len / 2; ++j) {

//// int u = a[i + j], v = int(a[i + j + len / 2] \* 1ll \* w % mod);

//// a[i + j] = u + v < mod ? u + v : u + v - mod;

//// a[i + j + len / 2] = u - v >= 0 ? u - v : u - v + mod;

//// w = int(w \* 1ll \* wlen % mod);

//// }

//// }

//// }

// for (int i = 2; i <= n; i <<= 1){

// int step = n / i;

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

// for (int k = 0; k < i / 2; ++k){

// base u = a[j + k];

// base v = 1ll \* a[j + k + i / 2] \* roots[step \* k] % mod;

// a[j + k] = u + v;

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

// }

// }

// }

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

int step = n / i;

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

for (int k = 0; k < i / 2; ++k){

base u = a[j + k];

base v = 1ll \* a[j + k + i / 2] \* roots[step \* k] % mod;

a[j + k] = (1ll \* u + v) % mod;

a[j + k + i / 2] = (1ll \* u - v + mod) % mod;

}

}

}

// if (inv) for (int i = 0; i < n; ++i) a[i] /= n;

if (inv) {

for (int i = 0; i < n; ++i) a[i] = 1ll \* a[i] \* Pow(n, mod - 2, mod) % mod;

}

}

vector<long long> multiply(vector<long long> const& a, vector<long long> const& b) {

if (b.size() == 1) return a;

if (a.size() == 1) return b;

vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());

int n = 1;

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

fa.resize(n);

fb.resize(n);

fft(fa, false);

fft(fb, false);

for (int i = 0; i < n; i++) fa[i] \*= fb[i];

fft(fa, true);

vector<long long> result(n);

for (int i = 0; i < n; i++) result[i] = (long long)round(fa[i].real()); //or % MOD;

return result;

}

\* This is a simplex solver. Given m x n matrix A, m-vector b, n-vector c,

\* finds n-vector x such that Ax <= b (component-wise)

\* maximizing <c, x>, where <x, y> is the dot product of x and y.

template<typename num\_t>

struct LPsolver {

const num\_t eps = 1e-9;

inline int sign(num\_t x) {return x < -eps ? -1 : +eps < x;}

inline int sign(num\_t x, num\_t y) {return sign(x - y);}

int m, n;

vector<int> B, N;

vector<vector<double>> D;

LPsolver(const vector<vector<double>> &A, const vector<num\_t> &b, const vector<num\_t> &c) : m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, vector<num\_t>(n + 2)) {

for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A[i][j];

for (int i = 0; i < m; i++) {

B[i] = n + i;

D[i][n] = -1;

D[i][n + 1] = b[i];

}

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

N[j] = j;

D[m][j] = -c[j];

}

N[n] = -1;

D[m + 1][n] = 1;

}

void pivot(int r, int s) {

for (int i = 0; i < m + 2; i++) if (i != r) {

for (int j = 0; j < n + 2; j++) if (j != s) {

D[i][j] -= D[r][j] \* D[i][s] / D[r][s];

}

}

for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] /= D[r][s];

for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] /= -D[r][s];

D[r][s] = 1.0 / D[r][s];

swap(B[r], N[s]);

}

int simple(int phase) {

int x = phase == 1 ? m + 1 : m;

while (1) {

int s = -1;

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

if (phase == 2 && N[j] == -1) continue;

if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j] < N[s]) s = j;

}

if (sign(D[x][s]) >= 0) return 1;

int r = -1;

for (int i = 0; i < m; i++) {

if (sign(D[i][s]) <= 0) continue;

if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] || D[i][n + 1] / D[i][s] == D[r][n + 1] / D[r][s] && B[i] < B[r]) r = i;

}

if (r == -1) return 0;

pivot(r, s);

}

}

num\_t solve(vector<num\_t>& x) {

int r = 0;

for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;

if (sign(D[r][n + 1]) < 0) {

pivot(r, n);

if (!simple(1) || sign(D[m + 1][n + 1]) < 0) return -numeric\_limits<num\_t>::infinity();

for (int i = 0; i < m; i++) if (B[i] == -1) {

int s = -1;

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

if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[j] < N[s]) s = j;

}

pivot(i, s);

}

}

if (!simple(2)) return numeric\_limits<num\_t>::infinity();

x = vector<num\_t>(n);

for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];

return D[m][n + 1];

}

};

const long double EPS = 1e-9; //Khử gauss

vector <int> where;

int gauss (vector < vector<double> > a, vector<double> & ans) {

int n = (int) a.size();

int m = (int) a[0].size() - 1;

where.resize(m);

for (int i = 0; i < m; i++) where[i] = -1;

for (int col=0, row=0; col<m && row<n; ++col) {

int sel = row;

for (int i=row; i<n; ++i)

if (abs (a[i][col]) > abs (a[sel][col]))

sel = i;

if (abs (a[sel][col]) < EPS)

continue;

for (int i=col; i<=m; ++i)

swap (a[sel][i], a[row][i]);

where[col] = row;

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

if (i != row) {

double c = a[i][col] / a[row][col];

for (int j=col; j<=m; ++j)

a[i][j] -= a[row][j] \* c;

}

++row;

}

ans.assign (m, 0);

for (int i=0; i<m; ++i)

if (where[i] != -1)

ans[i] = a[where[i]][m] / a[where[i]][i];

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

double sum = 0;

for (int j=0; j<m; ++j)

sum += ans[j] \* a[i][j];

if (abs (sum - a[i][m]) > EPS)

return 0;

}

// If we need any solution (in case INF solutions), we should be

// ok at this point.

// If need to solve partially (get which values are fixed/INF value):

for (int i=0; i<m; ++i)

if (where[i] != -1) {

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

if (j != i && fabs(a[where[i]][j]) > EPS) {

where[i] = -1;

break;

}

}

}

// Then the variables which has where[i] == -1 --> INF values

for (int i=0; i<m; ++i)

if (where[i] == -1)

return module;

return 1;

}

#define EPS 1e-9

typedef double T;

typedef vector<T> ROW;

typedef vector<ROW> MATRIX;

inline int sign(T x) {return x < -EPS ? -1 : x > +EPS;}

MATRIX MatrixInverse(MATRIX a) {

int i, j, k, n = a.size();

MATRIX res;

res.resize(n);

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

res[i].resize(n);

for (j = 0; j < n; j++) res[i][j] = 0;

res[i][i] = 1;

}

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

if (!sign(a[i][i])) {

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

if (sign(a[j][i])) {

for (k = 0; k < n; k++) {

a[i][k] += a[j][k];

res[i][k] += res[j][k];

}

break;

}

}

if (j == n) {

res.clear();

return res;

}

}

T tmp = a[i][i];

for (k = 0; k < n; k++) {

a[i][k] /= tmp;

res[i][k] /= tmp;

}

for (j = 0; j < n; j++) {

if (j == i) continue;

tmp = a[j][i];

for (k = 0; k < n; k++) {

a[j][k] -= a[i][k] \* tmp;

res[j][k] -= res[i][k] \* tmp;

}

}

}

return res;

}

///Tam giác có diện tích lớn nhất + Tứ giác có diện tích lớn nhất

///Bao lồi -> ra tập đỉnh bao lồi A

///S(A, B, C): Diện tích tam giác ABC

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

int L = i;

for (int j = (i + 1) % n; j != i; j = (j + 1) % n) {

while (S(a[i],a[L],a[j]) < S(a[i],a[(L + 1) % n],a[j])) {

L = (L + 1) % n;

}

F1[i][j] = S(a[i], a[L], a[j]);

}

}

for (int i = n - 1; i >= 0; i--) {

int L = i;

for (int j = (i - 1 + n) % n; j != i; j = (j - 1 + n) % n) {

while (S(a[i], a[L], a[j]) < S(a[i], a[(L - 1 + n) % n], a[j])) {

L = (L - 1 + n) % n;

}

F2[i][j] = S(a[i], a[L], a[j]);

}

}

struct Point {

int x, yL, yR, type; //type = 1: cạnh mở (bên trái), type = -1: cạnh đóng (bên phải)

Point(){};

Point(int \_x, int \_yL, int \_yR, int \_type) : x(\_x), yL(\_yL), yR(\_yR), type(\_type) {};

bool operator < (const Point &A) {

if (x != A.x) return x < A.x;

return type < A.type;

}

};

struct Node {

int X, Y;

Node(){};

Node(int \_X, int \_Y) : X(\_X), Y(\_Y) {};

};

vector <int> posY; ///rời rạc hóa các tọa độ Y

int Length(int yL, int yR) {

return posY[yR] - posY[yL];

}

struct IT\_AreaLe { //Diện tích vùng giao lẻ hình chữ nhật chồng lên

Node IT[maxN \* 4];

int n;

int Lazy[maxN \* 4];

IT\_AreaLe(){};

IT\_AreaLe(int \_n) {

n = \_n;

for (int i = 0; i < n \* 4; i++) Lazy[i] = IT[i].X = IT[i].Y = 0;

}

void cn(int i, int L, int R) {

if (Lazy[i] == 0) return;

IT[i].Y = Length(L, R) - IT[i].Y;

if (L + 1 < R) { ///L + 1 >= R is leaf

Lazy[i << 1] = (Lazy[i << 1] + Lazy[i]) % 2;

Lazy[i << 1 | 1] = (Lazy[i << 1 | 1] + Lazy[i]) % 2;

}

Lazy[i] = 0;

}

void update(int i, int L, int R, int u, int v, int val) {

cn(i, L, R);

if (L >= v || R <= u) return;

if (L >= u && R <= v) { Lazy[i] += val; cn(i, L, R); return; }

int mid = (L + R) >> 1;

update(i << 1, L, mid, u, v, val);

update(i << 1 | 1, mid, R, u, v, val);

IT[i].Y = IT[i << 1].Y + IT[i << 1 | 1].Y;

}

void up\_Y(int yL, int yR, int val) {update(1, 0, n - 1, yL, yR, val); }

};

long long AreaLe(vector <Point> &a) {

sort(a.begin(), a.end());

int m = (int)posY.size();

IT\_AreaLe T(m);

long long Res = 0;

for (int i = 0; i < (int)a.size() - 1; i++) {

int yL = lower\_bound(posY.begin(), posY.end(), a[i].yL) - posY.begin();

int yR = lower\_bound(posY.begin(), posY.end(), a[i].yR) - posY.begin();

T.up\_Y(yL, yR, 1);

int X = a[i + 1].x - a[i].x;

int Y = T.IT[1].Y;

Res += 1LL \* X \* Y;

}

return Res;

}

#define EPS 1e-9

struct point\_t {

double x, y;

point\_t() : x(0), y(0) {}

point\_t(double x, double y) : x(x), y(y) {}

point\_t(const point\_t& p) : x(p.x), y(p.y) {}

int operator < (const point\_t& rhs) const {return make\_pair(y, x) < make\_pair(rhs.y, rhs.x);}

int operator == (const point\_t& rhs) const {return make\_pair(y, x) == make\_pair(rhs.y, rhs.x);}

point\_t operator + (const point\_t& p) const {return point\_t(x + p.x, y + p.y);}

point\_t operator - (const point\_t& p) const {return point\_t(x - p.x, y - p.y);}

point\_t operator \* (double c) const {return point\_t(x \* c, y \* c);}

point\_t operator / (double c) const {return point\_t(x / c, y / c);}

};

double cross(point\_t p, point\_t q) {return p.x \* q.y - p.y \* q.x;}

double area(point\_t a, point\_t b, point\_t c) {return fabs(cross(a, b) + cross(b, c) + cross(c, a)) / 2;}

double area2(point\_t a, point\_t b, point\_t c) {return cross(a, b) + cross(b, c) + cross(c, a);}

double dot(point\_t p, point\_t q) {return p.x \* q.x + p.y \* q.y;}

double dist(point\_t p, point\_t q) {return sqrt(dot(p - q, p - q));}

double dist2(point\_t p, point\_t q) {return dot(p - q, p - q);}

point\_t RotateCCW90(point\_t p) {return point\_t(-p.y, p.x);}

point\_t RotateCW90(point\_t p) {return point\_t(p.y, -p.x);}

point\_t RotateCCW(point\_t p, double t) {return point\_t(p.x \* cos(t) - p.y \* sin(t), p.x \* sin(t) + p.y \* cos(t));}

int sign(double x) {return x < -EPS ? -1 : x > EPS;}

int sign(double x, double y) {return sign(x - y);}

ostream& operator << (ostream& os, const point\_t& p) {

os << "(" << p.x << "," << p.y << ")";

return os;

}

//Project c on Line(a, b)

point\_t ProjectPointLine(point\_t a, point\_t b, point\_t c) {

return a + (b - a) \* dot(c - a, b - a) / dot(b - a, b - a);

}

point\_t ProjectPointSegment(point\_t a, point\_t b, point\_t c) {

double r = dot(b - a, b - a);

if (fabs(r) < EPS) return a;

r = dot(c - a, b - a) / r;

if (r < 0) return a;

if (r > 1) return b;

return a + (b - a) \* r;

}

double DistancePointSegment(point\_t a, point\_t b, point\_t c) {

return dist(c, ProjectPointSegment(a, b, c));

}

//Compute distance between point\_t (x, y, z) and plane ax + by + cz = d

double DistancePointPlane(double x, double y, double z, double a, double b, double c, double d) {

return fabs(a \* x + b \* y + c \* z - d) / sqrt(a \* a + b \* b + c \* c);

}

//Determine if lines from a to b and c to d are parallel or collinear

int LinesParallel(point\_t a, point\_t b, point\_t c, point\_t d) {

return fabs(cross(b - a, c - d)) < EPS;

}

int LinesCollinear(point\_t a, point\_t b, point\_t c, point\_t d) {

return LinesParallel(a, b, c, d) && fabs(cross(a - b, a - c)) < EPS && fabs(cross(c - d, c - a)) < EPS;

}

//Determine if line segment from a to b intersects with line segment from c to d

int SegmentsIntersect(point\_t a, point\_t b, point\_t c, point\_t d) {

if (LinesCollinear(a, b, c, d)) {

if (dist2(a, c) < EPS || dist2(a, d) < EPS || dist2(b, c) < EPS || dist2(b, d) < EPS) return 1;

if (dot(c - a, c - b) > 0 && dot(d - a, d - b) > 0 && dot(c - b, d - b) > 0) return 0;

return 1;

}

if (cross(d - a, b - a) \* cross(c - a, b - a) > 0) return 0;

if (cross(a - c, d - c) \* cross(b - c, d - c) > 0) return 0;

return 1;

}

//Compute intersection of line passing through a and b

//with line passing through c and d, assuming that unique

//intersection exists; for segment intersection, check if

//segments intersect first

point\_t ComputeLineIntersection(point\_t a, point\_t b, point\_t c, point\_t d) {

b = b - a; d = c - d; c = c - a;

return a + b \* cross(c, d) / cross(b, d);

}

//Compute center of circle given three points

point\_t ComputeCircleCenter(point\_t a, point\_t b, point\_t c) {

b = (a + b) / 2;

c = (a + c) / 2;

return ComputeLineIntersection(b, b + RotateCW90(a - b), c, c + RotateCW90(a - c));

}

//Determine if point is in a possibly non-convex polygon

//returns 1 for strictly interior points, 0 for

//strictly exterior points, and 0 or 1 for the remaining points.

int PointInPolygonSlow(const vector<point\_t>& p, point\_t q) {

int c = 0;

for (int i = 0; i < p.size(); i++) {

int j = (i + 1) % p.size();

if ((p[i].y <= q.y && q.y < p[j].y || p[j].y <= q.y && q.y < p[i].y) && q.x < p[i].x + (p[j].x - p[i].x) \* (q.y - p[i].y) / (p[j].y - p[i].y)) c = !c;

}

return c;

}

//Strictly inside convex Polygon //Log(N)

#define Det(a, b, c) ((b.x - a.x) \* (c.y - a.y) - (b.y - a.y) \* (c.x - a.x))

int PointInPolygon(vector<point\_t>& p, point\_t q) {

int a = 1, b = p.size() - 1, c;

if (Det(p[0], p[a], p[b]) > 0) swap(a, b);

//Allow on edge --> if (Det... > 0 || Det ... < 0)

if (Det(p[0], p[a], q) >= 0 || Det(p[0], p[b], q) <= 0) return 0;

while(abs(a - b) > 1) {

c = (a + b) / 2;

if (Det(p[0], p[c], q) > 0) b = c; else a = c;

}

//Alow on edge --> return Det... <= 0

return Det(p[a], p[b], q) < 0;

}

//Determine if point is on the boundary of a polygon

int PointOnPolygon(const vector<point\_t>& p, point\_t q) {

for (int i = 0; i < p.size(); i++) if (dist2(ProjectPointSegment(p[i], p[(i + 1) % p.size()], q), q) < EPS) return 1;

return 0;

}

//Compute intersection of line through points a and b with circle centered at c with radius r > 0

vector<point\_t> CircleLineIntersection(point\_t a, point\_t b, point\_t c, double r) {

vector<point\_t> res;

b = b - a; a = a - c;

double A = dot(b, b);

double B = dot(a, b);

double C = dot(a, a) - r \* r;

double D = B \* B - A \* C;

if (D < -EPS) return res;

res.push\_back(c + a + b \* (-B + sqrt(D + EPS)) / A);

if (D > EPS) res.push\_back(c + a + b \* (-B - sqrt(D)) / A);

return res;

}

//Compute intersection of circle centered at a with radius r with circle centered at b with radius R

vector<point\_t> CircleCircleIntersection(point\_t a, point\_t b, double r, double R) {

vector<point\_t> res;

double d = sqrt(dist2(a, b));

if (d > r + R || d + min(r, R) < max(r, R)) return res;

double x = (d \* d - R \* R + r \* r) / (2 \* d);

double y = sqrt(r \* r - x \* x);

point\_t v = (b - a) / d;

res.push\_back(a + v \* x + RotateCCW90(v) \* y);

if (y > 0) res.push\_back(a + v \* x - RotateCCW90(v) \* y);

return res;

}

//This code computes the area or centroid of a (possibly nonconvex)

//polygon, assuming that the coordinates are listed in a clockwise or

//counterclockwise fashion. Note that the centroid is often known as

//the "center of gravity" or "center of mass".

double ComputeSignedArea(const vector<point\_t>& p) {

double area = 0;

for(int i = 0; i < p.size(); i++) {

int j = (i + 1) % p.size();

area += p[i].x \* p[j].y - p[j].x \* p[i].y;

}

return area / 2.0;

}

double ComputeArea(const vector<point\_t>& p) {

return fabs(ComputeSignedArea(p));

}

point\_t ComputeCentroid(const vector<point\_t>& p) {

point\_t c(0, 0);

double scale = 6.0 \* ComputeSignedArea(p);

for (int i = 0; i < p.size(); i++) {

int j = (i + 1) % p.size();

c = c + (p[i] + p[j]) \* (p[i].x \* p[j].y - p[j].x \* p[i].y);

}

return c / scale;

}

//Tests whether or not a given polygon (in CW or CCW order) is simple

int IsSimple(const vector<point\_t>& p) {

for (int i = 0; i < p.size(); i++) {

for (int k = i + 1; k < p.size(); k++) {

int j = (i + 1) % p.size();

int l = (k + 1) % p.size();

if (i == l || j == k) continue;

if (SegmentsIntersect(p[i], p[j], p[k], p[l])) return 0;

}

}

return 1;

}

double Angle(point\_t a) {

double PI = acos((double) - 1);

if (a.x == 0) {

if (a.y > 0) return PI / 2;

return 3 \* PI / 2;

}

if (a.y == 0) {

if (a.x > 0) return 0;

return PI;

}

double res = atan(a.y / a.x);

if (a.x < 0) return res + PI;

if (a.y < 0) return res + 2 \* PI;

return res;}

///minimum enclosing circle O(N)

struct Circle {

Point center;

double r;

Circle(){};

Circle(double \_x, double \_y, double \_r) : center(Point(\_x, \_y)), r(\_r) {};

Circle(Point \_center, double \_r) : center(\_center), r(\_r) {};

Circle(const Circle &A) : center(A.center), r(A.r) {};

int operator < (const Circle &A) {

return r < A.r;

}

};

double dist(Point A, Point B) {

double x = A.x - B.x;

double y = A.y - B.y;

return sqrtl(x \* x + y \* y);

}

bool inSide(Point A, Circle C) {return dist(A, C.center) <= C.r;}

Point get\_circle\_center(double bx, double by, double cx, double cy){

double B = bx \* bx + by \* by;

double C = cx \* cx + cy \* cy;

double D = bx \* cy - by \* cx;

return {(cy \* B - by \* C) / (2 \* D), (bx \* C - cx \* B) / (2 \* D)};

}

Circle circle\_three(Point A, Point B, Point C) {

Point O = get\_circle\_center(B.x - A.x, B.y - A.y, C.x - A.x, C.y - A.y);

O.x += A.x;

O.y += A.y;

return Circle(O, dist(O, A));

}

Circle circle\_two(Point A, Point B) {

Point O = Point((A.x + B.x) / 2.0, (A.y + B.y) / 2.0);

return Circle(O, dist(A, B) / 2.0);

}

bool check\_valid\_circle(Circle C, vector<Point> p) {

for (Point A : p) {

if (!inSide(A, C)) return false;

}

return true;

}

// Function to return the minimum enclosing circle for N <= 3

Circle MEC\_small(vector<Point> p) {

assert((int)p.size() <= 3);

if (p.empty()) return Circle(Point(0, 0), 0);

if ((int)p.size() == 1) return Circle(p[0], 0);

if ((int)p.size() == 2) return circle\_two(p[0], p[1]);

for (int i = 0; i < 3; i++) {

for (int j = i + 1; j < 3; j++) {

Circle C = circle\_two(p[i], p[j]);

if (check\_valid\_circle(C, p)) return C;

}

}

Circle res = circle\_three(p[0], p[1], p[2]);

return res;

}

Circle welzl\_helper(vector<Point>& P, vector<Point> R, int n) {

if (n == 0 || R.size() == 3) return MEC\_small(R);

int idx = rand() % n;

Point p = P[idx];

swap(P[idx], P[n - 1]);

Circle C = welzl\_helper(P, R, n - 1);

if (inSide(p, C)) return C;

R.push\_back(p);

return welzl\_helper(P, R, n - 1);

}

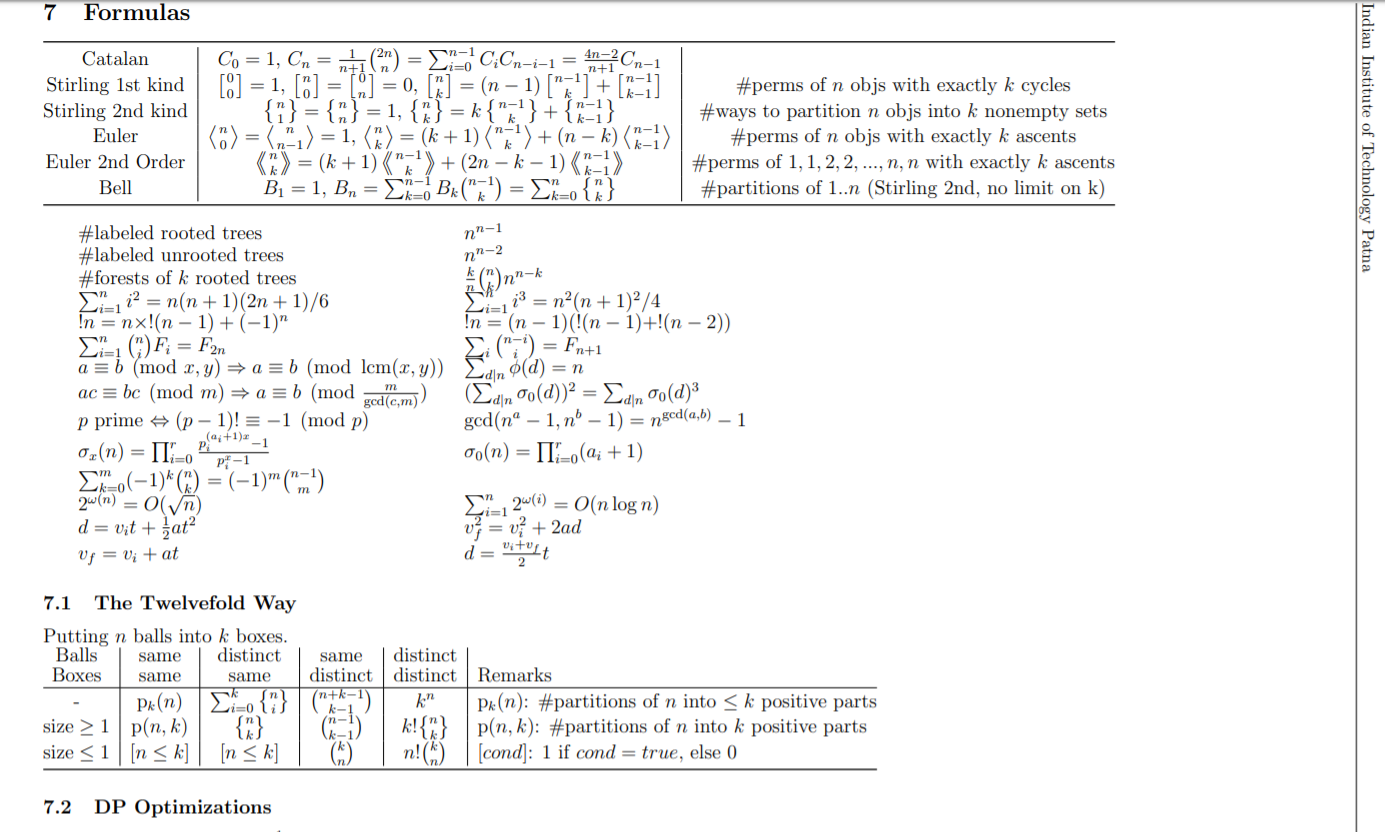
Circle MEC(vector<Point> p) {

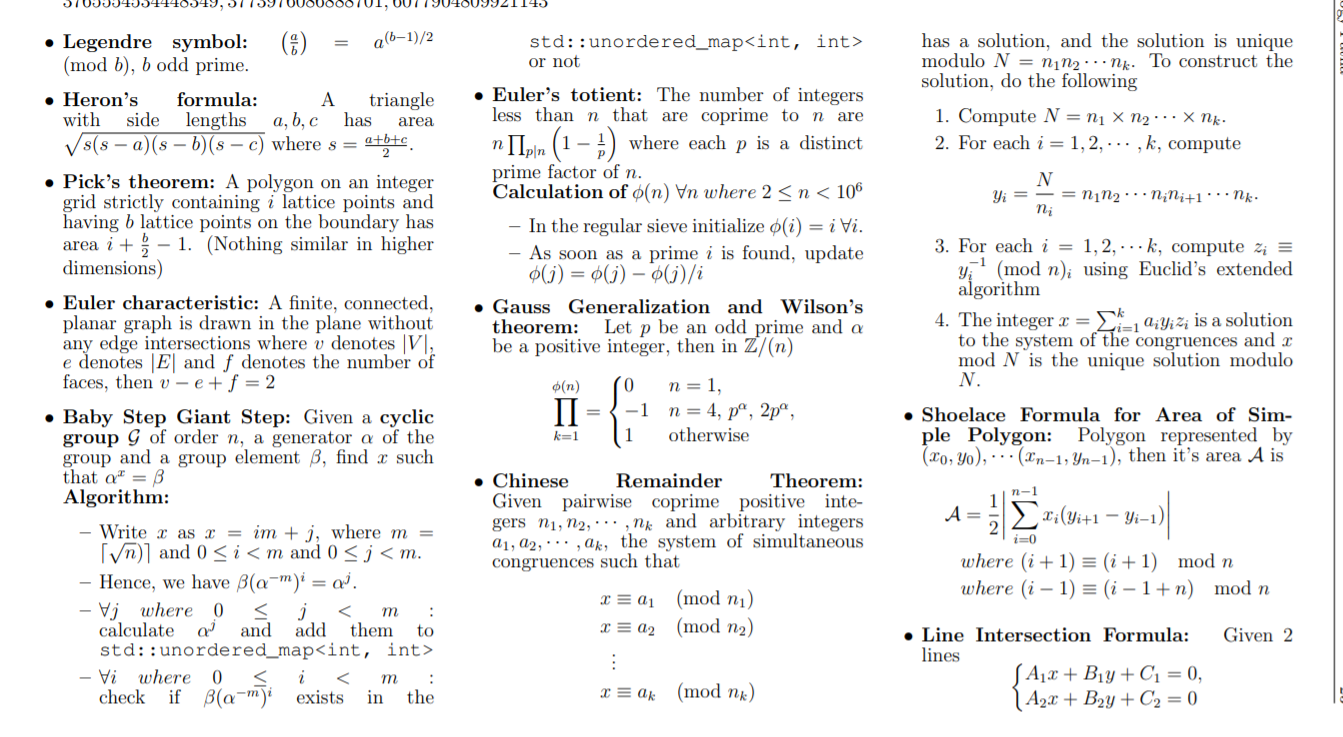
int n = (int)p.size();

random\_shuffle(p.begin(), p.end());

return welzl\_helper(p, {}, n);

}





Số Nguyên tố

8 digits - 59707699, 84765091, 64216913, 36853373, 91814719, 29647939, 99082553, 68007601, 35386633, 91221883

9 digits - 267222157, 248334941, 853519241, 879700489, 529560481, 160736231, 308615471, 722344243, 546428819, 528094447

12 digits - 744903658181, 805685255317, 901677551977, 645778995493, 951016942451, 743768119319, 463374658853, 390290791217, 730300933471

