Содержание

1 Общее

- Собственное вращение на угол φ с центром вращения в начале координат:
  $x' = x\cos\varphi - y\sin\varphi$
  $y' = x\sin\varphi + y\cos\varphi$
- Расстояние между точками по сфере: $L = R \cdot \arccos(\cos\theta_1 \cdot \cos\theta_2 + \sin\theta_1 \cdot \sin\theta_2 \cdot \cos(\varphi_1 - \varphi_2))$ где θ – широты (от $-\pi$ до π), φ – долготы (от $-\pi$ до π)
- Объем шарового сегмента: $V = \pi h^2(R - \frac{1}{3}h)$, где h – высота от вершины сектора до секущей плоскости
- Площадь поверхности шарового сегмента: $S = 2\pi Rh$, где h – высота
- Код Грея: $g_n = n \oplus \frac{n}{2}$
- Числа Фибоначчи: $F_0 = 0, F_1 = 1, F_n = \frac{(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n}{\sqrt{5}}$
- Sum-xor property: $a + b = a \oplus b + 2(a\&b), a + b = a|b + a\&b, a \oplus b = a|b - a\&b$
- Число граней в планарном графе(с учётом бесконечной): $R = 2 - V + E$
- Сумма арифметической прогрессии: $S_n = \frac{n(a_1+a_n)}{2}$
- Сумма геометрической прогрессии: $S_n = \frac{b_1(q^n-1)}{q-1}$

2 Коды

2.1 Basic setup

```cpp
#include <bits/stdc++.h>

using namespace std;

#define sz(x) (int) ((x).size())
#define all(x) (x).begin(), (x).end()
#define rall(x) (x).rbegin(), (x).rend()

typedef long long ll;
typedef __int128 int128;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;

const char en = '\n';
const int INF = 1e9 + 7;
const ll INFLL = 1e18;

mt19937 rnd(chrono::high_resolution_clock::now().time_since_epoch().count()
    );

template<class T>
istream &operator>>(istream &is, vector<T> &a) {
    for (auto &i : a) {
        is >> i;
    }
    return is;
}

void solve() {
}

int32_t main() {
#ifdef LOCAL
    freopen("input.txt", "r", stdin);
#else
    ios_base::sync_with_stdio(0);
    cin.tie(0);
#endif
    solve();
    return 0;
}
```

2.2 Бесполезное

Санитайзеры:

```
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wshadow -g -fsanitize=
    undefined -fsanitize=bounds -fsanitize=address -D_GLIBCXX_DEBUG")

-Wall -Wextra -pedantic -Wformat=2 -Wfloat-equal -Wconversion -Wlogical-op
    -Wshift-overflow=2 -Wduplicated-cond -Wcast-qual -Wcast-align -Werror
```

Прагмы:

```cpp
#pragma GCC optimize("Ofast,no-stack-protector")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4)
#pragma GCC target("popcnt,abm,mmx,avx,avx2,tune=native")
#pragma GCC optimize("unroll-loops")
#pragma GCC optimize("fast-math")
#pragma GCC optimize("section-anchors")
#pragma GCC optimize("profile-values")
#pragma GCC optimize("profile-reorder-functions")
#pragma GCC optimize("tracer")
#pragma GCC optimize("vpt")
#pragma GCC optimize("rename-registers")
#pragma GCC optimize("move-loop-invariants")
#pragma GCC optimize("unswitch-loops")
#pragma GCC optimize("function-sections")
#pragma GCC optimize("data-sections")
#pragma GCC optimize("branch-target-load-optimize")
#pragma GCC optimize("branch-target-load-optimize2")
#pragma GCC optimize("btr-bb-exclusive")
```

Встроенный декартач:

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template<class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
ordered_set q;
q.find_by_order(1);
q.order_of_key(2);
```

Atomic hashset, hashmap:

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
gp_hash_table<int, int> table;
-----------------------------------------------const int RANDOM = chrono::
    high_resolution_clock::now()
                                                                        .
    time_since_epoch()
                                                                        .
    count();
struct chash {
    int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<key, int, chash> table;
-----------------------------------------------typedef cc_hash_table<int,
    int, hash<int>> ht;
```

Перебор всех подмасок и надмасок:

```
for (int submask = mask;; submask = (submask - 1) & mask) {
    // use submask
    if (submask == 0) break;
}

for (int upmask = mask;; upmask = (upmask + 1) | mask) {
    // use upmask
    if (upmask == maxmask) break;
}
```

2.3 Мосты

```
void dfs(int v, int p = -1) {
    used[v] = true;
    tin[v] = fup[v] = timer++;
    for (auto to : g[v]) {
        if (to == p) {
            continue;
        }
        if (used[to]) {
            fup[v] = min(fup[v], tin[to]);
        } else {
            dfs(to, v);
            fup[v] = min(fup[v], fup[to]);
            if (fup[to] > tin[v]) {
                IS_BRIDGE(v, to);
            }
        }
    }
}
```

2.4 Точки сочленения

```
void dfs(int v, int p = -1) {
    used[v] = true;
    tin[v] = fup[v] = timer++;
    int children = 0;
    for (auto to : g[v]) {
        if (to == p) {
            continue;
        }
        if (used[to]) {
            fup[v] = min(fup[v], tin[to]);
        } else {
            dfs(to, v);
            fup[v] = min(fup[v], fup[to]);
            if (fup[to] >= tin[v] && p != -1) {
                IS_CUTPOINT(v);
            }
            ++children;
        }
    }
    if (p == -1 && children > 1) {
        IS_CUTPOINT(v);
    }
}
```

2.5 DCP (TheEvilBird)

```
struct Query {
    char type;
    int v, u;

    Query(char type) : type(type) {}
    Query(char type, int v, int u) : type(type), v(v), u(u) {}
};


struct DCP {
    int n, k, ans;// n - vertex, k - queries
    vector<int> par, rk;
    vector<pair<pii, int>> hist;
    // 0 - par, 1 - rk, 2 - ans;

    int qL, qR;
    pii edge;
    vector<vector<pii>> tree;
    vector<Query> qs;

    DCP(int _n, int _k) {
        n = ans = _n;
        par.resize(n);
        rk.resize(n, 1);
        for (int i = 0; i < n; ++i) par[i] = i;
        k = _k;
        tree.assign(4 * k, vector<pii>());
    }

    int dsu_get(int v) {
        while (par[v] != v) v = par[v];
        return v;
    }

    void dsu_unite(int a, int b) {
        a = dsu_get(a);
        b = dsu_get(b);
        if (a == b) return;
        if (rk[a] > rk[b]) swap(a, b);
        hist.emplace_back((pii){0, a}, par[a]);
        hist.emplace_back((pii){2, -1}, ans);
        par[a] = b;
        --ans;
        if (rk[a] == rk[b]) {
            hist.emplace_back((pii){1, b}, rk[b]);
            ++rk[b];
        }
    }

    void dsu_unite(pii e) {
        dsu_unite(e.first, e.second);
    }

    void cancel(pair<pii, int> &el) {
        int &type = el.first.first;
        int &id = el.first.second;
        int &val = el.second;
        if (type == 0) {
            par[id] = val;
        } else if (type == 1) {
            rk[id] = val;
        } else if (type == 2) {
            ans = val;
        }
    }

    void add_edge(int _qL, int _qR, pii e) {// [L, R]
        qL = _qL;
        qR = _qR + 1;
        edge = e;
        add_edge_tree(1, 0, k);
    }

    void add_edge_tree(int v, int l, int r) {
        if (qL <= l && r <= qR) {
            tree[v].emplace_back(edge);
            return;
        }
        int m = (l + r) / 2, vL = 2 * v, vR = vL + 1;
        if (qL < m) add_edge_tree(vL, l, m);
        if (m < qR) add_edge_tree(vR, m, r);
    }

    void go(vector<Query> &_qs) {
        qs = _qs;
        go_tree(1, 0, k);
    }

    void go_tree(int v, int l, int r) {
        int siz = sz(hist);
        for (auto &e : tree[v]) {
            dsu_unite(e);
```

```
        }
        if (l + 1 == r) {
            if (qs[l].type == '?') {
                cout << ans << en;
            }
        } else {
            int m = (l + r) / 2, vL = 2 * v, vR = vL + 1;
            go_tree(vL, l, m);
            go_tree(vR, m, r);
        }
        while (sz(hist) > siz) {
            cancel(hist.back());
            hist.pop_back();
        }
    }
};

void solve() {
    int n, k;
    cin >> n >> k;
    DCP dcp(n, k);
    set<pair<pii, int>> edges;
    vector<Query> qs;
    for (int i = 0; i < k; ++i) {
        char tp;
        cin >> tp;
        if (tp == '?') {
            qs.emplace_back(tp);
        } else {
            int v, u;
            cin >> v >> u;
            --v;
            --u;
            if (v > u) swap(v, u);
            qs.emplace_back(tp, v, u);
            if (tp == '+') {
                edges.emplace((pii){v, u}, i);
            } else {
                auto it = edges.lower_bound({(pii){v, u}, 0});
                dcp.add_edge(it->second, i, it->first);
                edges.erase(it);
            }
        }
    }
    for (auto &e : edges) {
        dcp.add_edge(e.second, k - 1, e.first);
    }
    if (k) dcp.go(qs);
}
```

2.6 MaxFlow (TheEvilBird)

```
struct MaxFlow {
    struct Edge {
        ll flow, cap;
        int to, id;

        Edge() {}

        Edge(ll flow, ll cap, int to, int id) : flow(flow), cap(cap), to(to
    ), id(id) {}
    };


    int n;
    vector<vector<Edge>> g;
    vector<int> d, head, used;
    ll max_cap;
    int s, t;

    MaxFlow() {}

    MaxFlow(int _n) {
        n = _n;
        g.resize(n);
    }

    void add_edge(int from, int to, ll cap) {
        g[from].emplace_back(0, cap, to, sz(g[to]));
        g[to].emplace_back(0, 0, from, sz(g[from]) - 1);
    }

    bool bfs() {
        d.assign(n, INF);
        d[s] = 0;
        queue<int> q;
        q.push(s);
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (auto e : g[v]) {
                if (d[e.to] == INF && e.cap - e.flow >= max_cap) {
                    d[e.to] = d[v] + 1;
                    q.push(e.to);
                }
            }
        }
        return d[t] != INF;
    }

    ll dfs(int v, ll cur_flow) {
        if (v == t) {
            return cur_flow;
        }
        for (; head[v] < sz(g[v]); ++head[v]) {
            auto &e = g[v][head[v]];
            if (e.cap - e.flow >= max_cap && d[v] + 1 == d[e.to]) {
                ll new_flow = dfs(e.to, min(cur_flow, e.cap - e.flow));
                if (new_flow) {
                    e.flow += new_flow;
                    g[e.to][e.id].flow -= new_flow;
                    return new_flow;
                }
            }
        }
        return 0;
    }

    ll find_max_flow(int _s, int _t) {
        s = _s;
        t = _t;
        ll res = 0;
        for (int k = 30; k >= 0; --k) {
            max_cap = (1 << k);
            while (bfs()) {
                head.assign(n, 0);
                ll flow = 0;
                do {
                    flow = dfs(s, INFLL);
                    res += flow;
                } while (flow);
            }
        }
        return res;
    }

    ll dfs_const_flow(int v, ll cur_flow) {
        used[v] = 1;
        if (v == t) {
            return cur_flow;
        }
        for (auto &e : g[v]) {
            if (!used[e.to] && e.cap - e.flow > 0) {
                ll new_flow = dfs_const_flow(e.to, min(cur_flow, e.cap - e.
     flow));
                if (new_flow) {
                    e.flow += new_flow;
                    g[e.to][e.id].flow -= new_flow;
                    return new_flow;
                }
            }
        }
        return 0;
    }

    bool find_const_flow(int _s, int _t, ll F) {
        s = _s;
        t = _t;
        ll res = 0, flow = 0;
        max_cap = F;
        do {
            used.assign(n, 0);
            flow = dfs_const_flow(s, INF);
            res += flow;
        } while (flow && res < F);
        return res == F;
    }

    ll get_edge_flow(int v, int id) {
        return g[v][id].flow;
    }
};
```

2.7 MinCostMaxFlow (TheEvilBird)

```
struct MinCostMaxFlow {
    struct Edge {
        ll flow, cap, price;
        int to, id;

        Edge() {}

        Edge(ll flow, ll cap, ll price, int to, int id) : flow(flow), cap(
     cap), price(price), to(to), id(id) {}
    };
```

```
    int n;
    int s, t;
    ll ans;
    vector<vector<Edge>> g;
    vector<int> d;
    vector<ll> add_f;
    vector<pii> par;

    MinCostMaxFlow() {}

    MinCostMaxFlow(int _n) {
        n = _n;
        g.resize(n);
    }

    void add_edge(int from, int to, ll cap, ll price) {
        g[from].emplace_back(0, cap, price, to, sz(g[to]));
        g[to].emplace_back(0, 0, -price, from, sz(g[from]) - 1);
    }

    ll get_edge_flow(int v, int id) {
        return g[v][id].flow;
    }

    void FB() {
        d.assign(n, INF);
        add_f.assign(n, 0);
        par.assign(n, {-1, -1});
        d[s] = 0;
        add_f[0] = INF;
        queue<int> q;
        q.push(s);
        vector<int> used(n, 0);
        used[s] = 1;
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            used[v] = 0;
            for (int i = 0; i < sz(g[v]); ++i) {
                auto &e = g[v][i];
                if (e.flow < e.cap && d[e.to] > d[v] + e.price) {
                    d[e.to] = d[v] + e.price;
                    add_f[e.to] = min(add_f[v], e.cap - e.flow);
                    par[e.to] = {v, i};
                    if (!used[e.to]) {
                        q.push(e.to);
                        used[e.to] = 1;
                    }
                }
            }
        }
    }

    void push_flow(ll flow) {
        int cur = t;
        while (cur != s) {
            int prev = par[cur].first, id = par[cur].second;
            g[prev][id].flow += flow;
            g[cur][g[prev][id].id].flow -= flow;
            ans += g[prev][id].price * flow;
            cur = prev;
        }
    }

    ll min_cost_max_flow(int _s, int _t) {
        ans = 0;
        s = _s;
        t = _t;
        while (true) {
            FB();
            ll flow = add_f[t];
            if (flow == 0) {
                break;
            }
            push_flow(flow);
        }
        return ans;
    }
};
```

2.8 Кун

```
bool dfs(int v) {
    if (used[v])
        return false;
    used[v] = true;
    for (auto u : g[v]) {
        if (back[u] == -1 || dfs(back[u])) {
            back[u] = v;
            return true;
        }
    }
    return false;
}
```

2.9 HLD (TheEvilBird)

```
struct HLD {

    // insert SegTree code
    struct SegTree {};

    int n, T;
    SegTree st;
    vector<vector<int>> tree;
    vector<int> par, siz, tin, tout, head;
    ll ans;

    HLD(int _n) {
        n = _n;
        tree.resize(n);
        par.resize(n, -1);
        siz.resize(n, 0);
        tin.resize(n);
        tout.resize(n);
        head.resize(n);
        st = SegTree(n);
    }

    void add_edge(int v, int u) {
        tree[v].emplace_back(u);
        tree[u].emplace_back(v);
    }

    void build(int v = 0) {
        dfs_siz(v, v);
        T = 0;
        head[v] = v;
        dfs_hld(v, v);
    }

    void dfs_siz(int v, int p) {
        par[v] = p;
        siz[v] = 1;
        for (auto &u : tree[v]) {
            if (u != p) {
                dfs_siz(u, v);
                siz[v] += siz[u];
            }
        }
        for (int i = 0; i < sz(tree[v]); ++i) {
            int x = tree[v][0], u = tree[v][i];
            if (x == p || siz[u] > siz[x]) {
                swap(tree[v][0], tree[v][i]);
            }
        }
    }

    void dfs_hld(int v, int p) {
        tin[v] = T++;
        for (auto u : tree[v]) {
            if (u == p) {
                continue;
            }
            if (u == tree[v][0]) {
                head[u] = head[v];
            } else {
                head[u] = u;
            }
            dfs_hld(u, v);
        }
        tout[v] = T;
    }

    void update(int v, int val) {
        st.update_segment(tin[v], tin[v], val);
    }

    bool is_anc(int v, int u) {
        return tin[v] <= tin[u] && tout[u] <= tout[v];
    }

    void go_up(int &v, int u) {
        while (!is_anc(head[v], u)) {
            ans = max(ans, st.get(tin[head[v]], tin[v]));
            v = par[head[v]];
        }
    }

    ll get(int v, int u) {// max on path
        ans = -INFLL;
        go_up(v, u);
        go_up(u, v);
        if (!is_anc(v, u)) {
            swap(v, u);
```

```
        }
        ans = max(ans, st.get(tin[v], tin[u]));
        return ans;
    }
};
```

2.10 Dominator tree (TheEvilBird)

```
struct Edge {
    int from, to, id;

    Edge() = default;
    Edge(int from, int to, int id) : from(from), to(to), id(id) {}
};

struct DSU {
    int n;
    vector<int> par;
    vector<pii> mn;

    DSU() = default;
    DSU(int n) : n(n) {
        par.resize(n);
        mn.resize(n);
        init();
    }

    void init() {
        for (int i = 0; i < n; ++i) {
            par[i] = i;
            mn[i] = {INF, i};
        }
    }

    int get(int v) {
        if (par[v] == v) {
            return v;
        }
        int p = get(par[v]);
//        mn[v] = min(mn[v], mn[par[v]]);
        if (mn[par[v]].first < mn[v].first) {
            mn[v] = mn[par[v]];
        }
        par[v] = p;
        return p;
    }

    void unite(int a, int b) {
        par[a] = b;
    }
};

struct DominatorTree {
    int n;
    vector<Edge> edges;
    vector<int> sdom, idom, tin, order, par, used, dp;
    vector<vector<int>> g, rg, queries;
    DSU dsu_sdom, dsu_idom;

    DominatorTree() = default;
    DominatorTree(int n) : n(n), dsu_sdom(n), dsu_idom(n) {
        sdom.resize(n, INF); // semi-dominator
        idom.resize(n, INF); // immediate dominator
        tin.resize(n, -1);
        par.resize(n);
        used.resize(n, 0);
        dp.resize(n, INF);
        g.resize(n);
        rg.resize(n);
        queries.resize(n);
    }

    void add_edge(int from, int to) {
        edges.emplace_back(from, to, sz(edges));
    }

    void dfs(int v) {
        tin[v] = sz(order);
        order.emplace_back(v);
        for (auto i: g[v]) {
            const auto &e = edges[i];
            if (tin[e.to] == -1) {
                par[e.to] = v;
                dfs(e.to);
            }
        }
    }

    void dfs_idom(int v) {
        used[v] = 1;
        for (auto i: g[v]) {
            const auto &e = edges[i];
            if (!used[e.to]) {
                dfs_idom(e.to);
            }
        }
        for (auto u: queries[v]) {
            dsu_idom.get(u);
            dp[u] = dsu_idom.mn[u].second;
        }
        dsu_idom.mn[v] = {sdom[v], v};
        for (auto i: g[v]) {
            const auto &e = edges[i];
            if (par[e.to] == v) {
                dsu_idom.unite(e.to, v);
            }
        }
    }

    void build(int s) {
        for (int i = 0; i < sz(edges); ++i) {
            g[edges[i].from].emplace_back(i);
            rg[edges[i].to].emplace_back(i);
        }
        // reorder vertex
        dfs(s);

        // build sdom
        for (int _ = sz(order) - 1; _ >= 0; --_) {
            int v = order[_];
            if (v == s) {
                continue;
            }
            for (auto i: rg[v]) {
                const auto &e = edges[i];
                if (tin[e.from] == -1) {
                    continue;
                }
                if (tin[e.from] < tin[v]) {
                    sdom[v] = min(sdom[v], tin[e.from]);
                }
                else {
                    int u = dsu_sdom.get(e.from);
                    sdom[v] = min(sdom[v], dsu_sdom.mn[e.from].first);
                }
            }
            dsu_sdom.mn[v] = {sdom[v], v};
            for (auto i: g[v]) {
                const auto &e = edges[i];
                if (v == par[e.to]) {
                    dsu_sdom.unite(e.to, v);
                }
            }
        }

        // build queries for idoms
        for (int i = 0; i < n; ++i) {
            if (i == s || sdom[i] == INF || tin[i] == -1) {
                continue;
            }
            queries[order[sdom[i]]].emplace_back(i);
        }
        dfs_idom(s);

        // build idom
        idom[s] = tin[s];
        for (auto v: order) {
            if (v == s) {
                continue;
            }
            if (v == dp[v]) {
                idom[v] = sdom[v];
            }
            else {
                idom[v] = idom[dp[v]];
            }
        }
    }

    int get_idom(int v) {
        return (idom[v] == INF ? -1 : order[idom[v]]);
    }
};
```

2.11 Link-Cut (TheEvilBird)

```
struct Node {
    Node *ch[2] = {nullptr, nullptr};
    Node *par = nullptr;
    bool rev = false;
    int val, mn;
    int siz = 1;

    Node() {}

    Node(int val) : val(val), mn(val) {}
```

```
};

typedef Node *pnode;

int get_siz(pnode v) {
    return (v == nullptr ? 0 : v->siz);
}

int get_min(pnode v) {
    return (v == nullptr ? INF : v->mn);
}

void update(pnode v) {
    v->siz = 1 + get_siz(v->ch[0]) + get_siz(v->ch[1]);
    v->mn = min(v->val, min(get_min(v->ch[0]), get_min(v->ch[1])));
}

void push(pnode v) {
    if (v == nullptr || !v->rev) {
        return;
    }
    if (v->ch[0] != nullptr) {
        v->ch[0]->rev ^= 1;
    }
    if (v->ch[1] != nullptr) {
        v->ch[1]->rev ^= 1;
    }
    swap(v->ch[0], v->ch[1]);
    v->rev = 0;
}

bool is_root(pnode v) {
    return (v->par == nullptr ||
            (v->par->ch[0] != v && v->par->ch[1] != v));
}

int child_num(pnode v) {
    return (v->par->ch[1] == v);
}

void attach(pnode v, pnode p, int num) {
    if (v != nullptr) {
        v->par = p;
    }
    if (p != nullptr) {
        p->ch[num] = v;
    }
}

void rotate(pnode v) {
    int num = child_num(v);
    pnode p = v->par, vb = v->ch[num ^ 1];
    pnode g = (p == nullptr ? nullptr : p->par);
    if (g != nullptr) {
        if (!is_root(p)) {
            g->ch[child_num(p)] = v;
        }
    }
    v->par = g;
    attach(p, v, num ^ 1);
    attach(vb, p, num);
    update(p);
    update(v);
}

void splay(pnode v) {
    vector<pnode> st;
    pnode cur = v;
    st.emplace_back(cur);
    while (!is_root(cur)) {
        cur = cur->par;
        st.emplace_back(cur);
    }
    for (int i = sz(st) - 1; i >= 0; --i) {
        push(st[i]);
    }
    while (!is_root(v)) {
        if (!is_root(v->par)) {
            if (child_num(v) == child_num(v->par)) {
                rotate(v->par);
            } else {
                rotate(v);
            }
        }
        rotate(v);
    }
}

void expose(pnode v) {
    splay(v);
    v->ch[1] = nullptr;
    update(v);
    while (v->par != nullptr) {
        splay(v->par);
        attach(v, v->par, 1);
        update(v->par);
        splay(v);
    }
}

void make_root(pnode v) {
    expose(v);
    v->rev ^= 1;
}

void link(pnode v, pnode u) {
    make_root(v);
    make_root(u);
    u->par = v;
}

void cut(pnode v, pnode u) {
    make_root(v);
    make_root(u);
    push(u);
    u->ch[1] = nullptr;
    v->par = nullptr;
}

bool is_connected(pnode v, pnode u) {
    make_root(v);
    make_root(u);
    if (is_root(v) && u != v) {
        return false;
    } else {
        return true;
    }
}

int get_min(pnode v, pnode u) {
    make_root(v);
    make_root(u);
    return get_min(u);
}
```

2.12 Личао (FedShat)

```
struct LiChao {// max
    struct Line {
        ll k = 0, b = -INFLL;

        Line() = default;

        Line(ll k, ll b) : k(k), b(b){};

        ll operator()(ll x) {
            return k * x + b;
        }
    };

    struct Node {
        Node *l = nullptr, *r = nullptr;
        Line cur;

        Node() = default;
    };

    Node *root = nullptr;
    int n = 1e9 + 1;

    void make_node(Node *&v) {
        if (v == nullptr) {
            v = new Node();
        }
    }

    void add(Node *&v, int l, int r, Line cur) {
        make_node(v);
        int m = (l + r) / 2;
        if (cur(m) > v->cur(m)) {
            swap(cur, v->cur);
        }
        if (l + 1 == r) {
            return;
        }
        if (cur(l) > v->cur(l)) {
            add(v->l, l, m, cur);
        } else {
            add(v->r, m, r, cur);
        }
    }

    void add(Line cur) {
        add(root, 0, n, cur);
    }

    ll get(Node *v, int l, int r, int x) {
        if (v == nullptr) {
            return -INFLL;
```

```
        }
        ll ans = v->cur(x);
        if (l + 1 == r) {
            return ans;
        }
        int m = (l + r) / 2;
        if (x < m) {
            ans = max(ans, get(v->l, l, m, x));
        } else {
            ans = max(ans, get(v->r, m, r, x));
        }
        return ans;
    }

    ll get(int x) {
        return get(root, 0, n, x);
    }
};
```

2.13 Segment Tree (TheEvilBird)

```
struct SegTree {

    static const ll off = 0;

    struct Node {
        ll val = 0;
        ll push = off;

        Node() {}

        Node(ll val) : val(val) {}

        Node operator+(const Node &other) const {
            return Node(val + other.val);
        }

        void operator+=(const Node &other) {
            val += other.val;
        }

        void use_push(int len = 1) {
            val += push * (ll) (len);
        }

        void update_push(ll pushed) {
            push += pushed;
        }
    };

    int n, qL, qR;
    ll val;
    Node ans;


    vector<Node> tree;
    vector<ll> a;

    SegTree() {}

    SegTree(int _n) {
        n = _n;
        tree.assign(4 * n, 0);
    }

    void update_vertex(int v, int l, int r) {// [l, r)
        int m = (l + r) / 2, vL = 2 * v, vR = vL + 1;
        push(vL, l, m);
        push(vR, m, r);
        tree[v] = tree[vL] + tree[vR];
    }

    void push(int v, int l, int r) {// [l, r)
        if (tree[v].push == off) return;
        int m = (l + r) / 2, vL = 2 * v, vR = vL + 1;
        tree[v].use_push(r - l);
        if (l + 1 != r) {
            tree[vL].update_push(tree[v].push);
            tree[vR].update_push(tree[v].push);
        }
        tree[v].push = off;
    }

    void build(vector<ll> &_a) {
        a = _a;
        build_tree(1, 0, n);
    }

    void build_tree(int v, int l, int r) {// [l, r)
        if (l + 1 == r) {
            tree[v] = Node(a[l]);
            return;
        }
        int m = (l + r) / 2, vL = 2 * v, vR = vL + 1;
        build_tree(vL, l, m);
        build_tree(vR, m, r);
        update_vertex(v, l, r);
    }

    void update_segment(int _qL, int _qR, ll _val) {// [_qL, _qR]
        qL = _qL;
        qR = _qR + 1;
        val = _val;
        update_segment_tree(1, 0, n);
    }

    void update_segment_tree(int v, int l, int r) {// [l, r)
        push(v, l, r);
        if (qL <= l && r <= qR) {
            tree[v].update_push(val);
            push(v, l, r);
            return;
        }
        int m = (l + r) / 2, vL = 2 * v, vR = vL + 1;
        if (qL < m) update_segment_tree(vL, l, m);
        if (m < qR) update_segment_tree(vR, m, r);
        update_vertex(v, l, r);
    }

    ll get(int _qL, int _qR) {// [_qL, _qR]
        qL = _qL;
        qR = _qR + 1;
        ans = Node();
        get_tree(1, 0, n);
        return ans.val;
    }

    void get_tree(int v, int l, int r) {// [l, r)
        push(v, l, r);
        if (qL <= l && r <= qR) {
            ans = ans + tree[v];
            return;
        }
        int m = (l + r) / 2, vL = 2 * v, vR = vL + 1;
        if (qL < m) get_tree(vL, l, m);
        if (m < qR) get_tree(vR, m, r);
    }
};
```

2.14 Segment Tree Down (TheEvilBird)

```
struct SegTreeDown {

    struct Node {
        ll val = 0;

        Node() {}
        Node(ll val) : val(val) {}

        Node operator+(const Node &other) const {
            return Node(val + other.val);
        }

        void operator+=(const Node &other) {
            val += other.val;
        }
    };

    int n;
    vector<Node> tree;

    SegTreeDown(int _n) {
        n = _n;
        tree.assign(2 * n, Node());
    }

    void build(vector<ll> &a) {
        for (int i = 0; i < n; ++i) {
            tree[i + n] = Node(a[i]);
        }
        for (int i = n - 1; i >= 1; --i) {
            tree[i] = tree[2 * i] + tree[2 * i + 1];
        }
    }

    void update(int i, ll val) {
        i += n;
        tree[i] = val;
        i /= 2;
        while (i != 0) {
            tree[i] = tree[2 * i] + tree[2 * i + 1];
            i /= 2;
        }
    }

    ll get(int l, int r) {// [l, r)
```

```
        --r;
        l += n;
        r += n;
        Node ans;
        while (l <= r) {
            if (l % 2 == 1) {
                ans += tree[l];
                ++l;
            }
            if (r % 2 == 0) {
                ans += tree[r];
                --r;
            }
            l /= 2;
            r /= 2;
        }
        return ans.val;
    }
};
```

2.15 Segment Tree Beats (TheEvilBird)

```
struct SegTree {
    struct Node {
        ll max, sec_max;
        int cnt_max;
        ll min, sec_min;
        int cnt_min;
        ll sum;
        ll push_add;
        ll push_eq;
    };

    int n, qL, qR;
    ll val, ans;

    vector<Node> tree;
    vector<ll> a;

    SegTree(int _n) {
        n = _n;
        tree.assign(4 * n, {0, -INFLL, 1, 0, INFLL, 1, 0, 0, -1});
    }

    void update_vertex(int v, int l, int r) {
        int m = (l + r) / 2, vL = 2 * v, vR = vL + 1;

        tree[v].sum = tree[vL].sum + tree[vR].sum;

        tree[v].max = max(tree[vL].max, tree[vR].max);
        tree[v].sec_max = max(tree[vL].sec_max, tree[vR].sec_max);
        tree[v].cnt_max = 0;
        if (tree[vL].max == tree[v].max) {
            tree[v].cnt_max += tree[vL].cnt_max;
        } else {
            tree[v].sec_max = max(tree[v].sec_max, tree[vL].max);
        }
        if (tree[vR].max == tree[v].max) {
            tree[v].cnt_max += tree[vR].cnt_max;
        } else {
            tree[v].sec_max = max(tree[v].sec_max, tree[vR].max);
        }

        tree[v].min = min(tree[vL].min, tree[vR].min);
        tree[v].sec_min = min(tree[vL].sec_min, tree[vR].sec_min);
        tree[v].cnt_min = 0;
        if (tree[vL].min == tree[v].min) {
            tree[v].cnt_min += tree[vL].cnt_min;
        } else {
            tree[v].sec_min = min(tree[v].sec_min, tree[vL].min);
        }
        if (tree[vR].min == tree[v].min) {
            tree[v].cnt_min += tree[vR].cnt_min;
        } else {
            tree[v].sec_min = min(tree[v].sec_min, tree[vR].min);
        }
    }

    void recalc_eq(int v, int l, int r, ll cur) {
        tree[v].max = tree[v].min = tree[v].push_eq = cur;
        tree[v].sec_max = -INFLL;
        tree[v].sec_min = INFLL;
        tree[v].cnt_max = tree[v].cnt_min = r - l;
        tree[v].sum = cur * (ll) (r - l);
        tree[v].push_add = 0;
    }

    void recalc_add(int v, int l, int r, ll cur) {
        if (tree[v].min == tree[v].max) {
            recalc_eq(v, l, r, tree[v].max + cur);
            return;
        }
        tree[v].max += cur;
        if (tree[v].sec_max != -INFLL) {
            tree[v].sec_max += cur;
        }

        tree[v].min += cur;
        if (tree[v].sec_min != INFLL) {
            tree[v].sec_min += cur;
        }

        tree[v].sum += (ll) (r - l) * cur;
        tree[v].push_add += cur;
    }

    void recalc_min(int v, int l, int r, ll cur) {
        if (tree[v].min >= cur) {
            recalc_eq(v, l, r, cur);
            return;
        }
        if (tree[v].max <= cur) return;
        if (tree[v].sec_min == tree[v].max) {
            tree[v].sec_min = cur;
        }
        tree[v].sum -= (ll) (tree[v].max - cur) *
                       (ll) (tree[v].cnt_max);
        tree[v].max = cur;
    }

    void recalc_max(int v, int l, int r, ll cur) {
        if (tree[v].max <= cur) {
            recalc_eq(v, l, r, cur);
            return;
        }
        if (tree[v].min >= cur) return;
        if (tree[v].sec_max == tree[v].min) {
            tree[v].sec_max = cur;
        }
        tree[v].sum += (ll) (tree[v].max - cur) *
                       (ll) (tree[v].cnt_max);
        tree[v].min = cur;
    }

    void push(int v, int l, int r) {
        if (l + 1 == r) return;
        int m = (l + r) / 2, vL = 2 * v, vR = vL + 1;
        if (tree[v].push_eq != -1) {
            recalc_eq(vL, l, m, tree[v].push_eq);
            recalc_eq(vR, m, r, tree[v].push_eq);
            tree[v].push_eq = -1;
            return;
        }

        recalc_add(vL, l, m, tree[v].push_add);
        recalc_add(vR, m, r, tree[v].push_add);
        tree[v].push_add = 0;

        recalc_min(vL, l, m, tree[v].max);
        recalc_min(vR, m, r, tree[v].max);

        recalc_max(vL, l, m, tree[v].min);
        recalc_max(vR, m, r, tree[v].min);
    }

    void build(vector<ll> &_a) {
        a = _a;
        build_tree(1, 0, n);
    }

    void build_tree(int v, int l, int r) {
        if (l + 1 == r) {
            tree[v] = {a[l], -INFLL, 1, a[l],
                       INFLL, 1, a[l], 0, -1};
            return;
        }
        int m = (l + r) / 2, vL = 2 * v, vR = vL + 1;
        build_tree(vL, l, m);
        build_tree(vR, m, r);
        update_vertex(v, l, r);
    }

    void update_segment_min(int _qL, int _qR, ll _val) {
        qL = _qL;
        qR = _qR + 1;
        val = _val;
        update_segment_min_tree(1, 0, n);
    }

    void update_segment_min_tree(int v, int l, int r) {
        if (tree[v].max <= val) return;
        if (qL <= l && r <= qR && tree[v].sec_max < val) {
            recalc_min(v, l, r, val);
            return;
        }
        push(v, l, r);
        int m = (l + r) / 2, vL = 2 * v, vR = vL + 1;
        if (qL < m) update_segment_min_tree(vL, l, m);
        if (m < qR) update_segment_min_tree(vR, m, r);
```

```
        update_vertex(v, l, r);
    }

    void update_segment_max(int _qL, int _qR, ll _val) {
        qL = _qL;
        qR = _qR + 1;
        val = _val;
        update_segment_max_tree(1, 0, n);
    }

    void update_segment_max_tree(int v, int l, int r) {
        if (tree[v].min >= val) return;
        if (qL <= l && r <= qR && tree[v].sec_min > val) {
            recalc_max(v, l, r, val);
            return;
        }
        push(v, l, r);
        int m = (l + r) / 2, vL = 2 * v, vR = vL + 1;
        if (qL < m) update_segment_max_tree(vL, l, m);
        if (m < qR) update_segment_max_tree(vR, m, r);
        update_vertex(v, l, r);
    }

    void update_segment_add(int _qL, int _qR, ll _val) {
        qL = _qL;
        qR = _qR + 1;
        val = _val;
        update_segment_add_tree(1, 0, n);
    }

    void update_segment_add_tree(int v, int l, int r) {
        if (qL <= l && r <= qR) {
            recalc_add(v, l, r, val);
            return;
        }
        push(v, l, r);
        int m = (l + r) / 2, vL = 2 * v, vR = vL + 1;
        if (qL < m) update_segment_add_tree(vL, l, m);
        if (m < qR) update_segment_add_tree(vR, m, r);
        update_vertex(v, l, r);
    }

    ll get(int _qL, int _qR) {
        qL = _qL;
        qR = _qR + 1;
        ans = 0;
        get_tree(1, 0, n);
        return ans;
    }

    void get_tree(int v, int l, int r) {
        if (qL <= l && r <= qR) {
            ans += tree[v].sum;
            return;
        }
        push(v, l, r);
        int m = (l + r) / 2, vL = 2 * v, vR = vL + 1;
        if (qL < m) get_tree(vL, l, m);
        if (m < qR) get_tree(vR, m, r);
    }

    void print_all() {
        print_all_tree(1, 0, n);
    }

    void print_all_tree(int v, int l, int r) {
        if (l + 1 == r) {
            cout << tree[v].sum << en;
            return;
        }
        push(v, l, r);
        int m = (l + r) / 2, vL = 2 * v, vR = vL + 1;
        print_all_tree(vL, l, m);
        print_all_tree(vR, m, r);
    }
};
```

2.16 Persistent Segment Tree (Sweezyk)

```
struct Node {
    Node *l, *r;
    int mx;
    Node() {
        mx = -1;
        l = r = nullptr;
    }
};

const int N = 1e7 + 4e6;
const int LG = 20;

Node *nodes[N];
int ptr;

Node *new_node() {
    return nodes[ptr++];
}

Node *get_left(Node *t) {
    if (t && t->l) return t->l;
    return nullptr;
}

Node *get_right(Node *t) {
    if (t && t->r) return t->r;
    return nullptr;
}

int get_max(Node *t) {
    if (!t) return 0;
    return t->mx;
}

void update(int i, int val, Node *t, Node *old_t, int lx, int rx) {
    if (lx + 1 == rx) {
        t->mx = max(t->mx, val);
        return;
    }
    int m = (lx + rx) / 2;
    if (i < m) {
        t->l = new_Node();
        if (old_t && old_t->l) {
            t->l->mx = old_t->l->mx;
        }
        t->r = get_right(old_t);
        update(i, val, t->l, get_left(old_t), lx, m);
    } else {
        t->r = new_Node();
        if (old_t && old_t->r) {
            t->r->mx = old_t->r->mx;
        }
        t->l = get_left(old_t);
        update(i, val, t->r, get_right(old_t), m, rx);
    }
    t->mx = max(get_max(get_left(t)), get_max(get_right(t)));
};

int get(int l, int r, Node *t, int lx, int rx) {
    if (!t || lx >= r || rx <= l) return -1;
    if (lx >= l && rx <= r) return t->mx;
    int m = (lx + rx) / 2;
    return max(get(l, r, t->l, lx, m), get(l, r, t->r, m, rx));
}
```

2.17 Fenwick (TheEvilBird)

```
struct Fenwick {
    int n;
    vector<ll> f;

    Fenwick(int _n) {
        n = _n;
        f.assign(n + 1, 0);
    }

    void update(int x, ll delta) {
        for (int i = x; i <= n; i += i & -i) {
            f[i] += delta;
        }
    }

    ll get_sum(int x) {
        ll s = 0;
        for (int i = x; i > 0; i -= i & -i) {
            s += f[i];
        }
        return s;
    }

    ll get(int L, int R) {// [L, R]
        return get_sum(R) - get_sum(L - 1);
    }
};
```

2.18 Sparse table (TheEvilBird)

```
struct Sparse {
    int n;
    vector<int> lg;
    vector<vector<ll>> table;

    Sparse(int _n) {
```

```
        n = _n;
        lg.resize(n + 1);
        for (int i = 2; i <= n; ++i) {
            lg[i] = lg[i / 2] + 1;
        }
        table.resize(lg[n] + 1, vector<ll>(n));
        for (int i = 0; i < n; ++i) table[0][i] = INF;
    }

    void build() {
        for (int i = 1; i < table.size(); ++i) {
            for (int j = 0; j + (1 << i) <= n; ++j) {
                table[i][j] = min(table[i - 1][j],
                                  table[i - 1][j + (1 << (i - 1))]);
            }
        }
    }

    ll get(int L, int R) {// [L, R]
        int D = R - L + 1, x = lg[D];
        return min(table[x][L], table[x][R - (1 << x) + 1]);
    }
};
```

2.19 Treap (Sweezyk)

```
struct Node {
    int x, y, size, push, cnt;
    Node *l, *r;

    Node(int val) : x(val), y(rng()), size(1), push(0), cnt(0), l(nullptr),
      r(nullptr) {}
};

void push(Node *t) {
    if (t == nullptr) return;
    int p = t->push;
    if (p == 0) return;
    if (t->l != nullptr) {
        t->l->cnt += p;
        t->l->push += p;
    }
    if (t->r != nullptr) {
        t->r->cnt += p;
        t->r->push += p;
    }
    t->push = 0;
}

int size(Node *t) {
    return (t ? t->size : 0);
}

void update(Node *t) {
    if (t == nullptr) return;
    t->size = size(t->l) + size(t->r) + 1;
}

pair<Node *, Node *> split(Node *t, int k) {
    if (t == nullptr) return {nullptr, nullptr};
    if (k == 0) return {nullptr, t};
    push(t);
    if (size(t->l) + 1 <= k) {
        auto [l, r] = split(t->r, k - size(t->l) - 1);
        t->r = l;
        update(t);
        return {t, r};
    } else {
        auto [l, r] = split(t->l, k);
        t->l = r;
        update(t);
        return {l, t};
    }
}

Node *merge(Node *tl, Node *tr) {
    if (tl == nullptr) return tr;
    if (tr == nullptr) return tl;
    push(tl);
    push(tr);
    if (tl->y > tr->y) {
        tl->r = merge(tl->r, tr);
        update(tl);
        return tl;
    } else {
        tr->l = merge(tl, tr->l);
        update(tr);
        return tr;
    }
}

void dfs(Node *t) {
    if (t == nullptr) return;
    push(t);
    dfs(t->l);
    cout << t->x << ' ' << t->cnt << '\n';
    dfs(t->r);
}

void solve() {
    int n, m;
    cin >> n >> m;
    Node *root = nullptr;
    for (int i = 1; i <= n; i++) {
        Node *add = new Node(i);
        root = merge(root, add);
    }
    for (int i = 0; i < m; i++) {
        int l, r;
        cin >> l >> r;
        auto [L, R] = split(root, r);
        auto [L1, L2] = split(L, l - 1);
        L2->push = 1;
        L2->cnt += 1;
        root = merge(L2, merge(L1, R));
    }
    dfs(root);
}
```

2.20 Extended GCD (Sweezyk)

```
template<typename T>
T extgcd(T a, T b, T &x, T &y) {
    if (a == 0) {
        x = 0;
        y = 1;
        return b;
    }
    T p = b / a;
    T g = extgcd(b - p * a, a, y, x);
    x -= p * y;
    return g;
}

template<typename T>
bool diophantine(T a, T b, T c, T &x, T &y, T &g) {
    if (a == 0 && b == 0) {
        if (c == 0) {
            x = y = g = 0;
            return true;
        }
        return false;
    }
    if (a == 0) {
        if (c % b == 0) {
            x = 0;
            y = c / b;
            g = abs(b);
            return true;
        }
        return false;
    }
    if (b == 0) {
        if (c % a == 0) {
            x = c / a;
            y = 0;
            g = abs(a);
            return true;
        }
        return false;
    }
    g = extgcd(a, b, x, y);
    if (c % g != 0) {
        return false;
    }
    T dx = c / a;
    c -= dx * a;
    T dy = c / b;
    c -= dy * b;
    x = dx + (T) ((__int128) x * (c / g) % b);
    y = dy + (T) ((__int128) y * (c / g) % a);
    g = abs(g);
    return true;
    // |x|, |y| <= max(|a|, |b|, |c|) [tested]
}

bool crt(long long k1, long long m1, long long k2, long long m2, long long
     &k, long long &m) {
    k1 %= m1;
    if (k1 < 0) k1 += m1;
    k2 %= m2;
    if (k2 < 0) k2 += m2;
    long long x, y, g;
    if (!diophantine(m1, -m2, k2 - k1, x, y, g)) {
        return false;
    }
```

```
    long long dx = m2 / g;
    long long delta = x / dx - (x % dx < 0);
    k = m1 * (x - dx * delta) + k1;
    m = m1 / g * m2;
    assert(0 <= k && k < m);
    return true;
}
```

2.21 FFT (FedShat)

```
constexpr int P = 998244353;
using i64 = long long;

// assume -P <= x < 2P
int norm(int x) {
    if (x < 0) {
        x += P;
    }
    if (x >= P) {
        x -= P;
    }
    return x;
}

template<class T>
T power(T a, int b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}

struct Z {
    int x;
    Z(int x = 0) : x(norm(x)) {}
    int val() const {
        return x;
    }
    Z operator-() const {
        return Z(norm(P - x));
    }
    Z inv() const {
        assert(x != 0);
        return power(*this, P - 2);
    }
    Z &operator*=(const Z &rhs) {
        x = i64(x) * rhs.x % P;
        return *this;
    }
    Z &operator+=(const Z &rhs) {
        x = norm(x + rhs.x);
        return *this;
    }
    Z &operator-=(const Z &rhs) {
        x = norm(x - rhs.x);
        return *this;
    }
    Z &operator/=(const Z &rhs) {
        return *this *= rhs.inv();
    }
    friend Z operator*(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res *= rhs;
        return res;
    }
    friend Z operator+(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res += rhs;
        return res;
    }
    friend Z operator-(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res -= rhs;
        return res;
    }
    friend Z operator/(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res /= rhs;
        return res;
    }
};

std::vector<int> rev;
std::vector<Z> roots{0, 1};
void dft(std::vector<Z> &a) {
    int n = a.size();

    if (int(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; i++) {
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
        }
    }

    for (int i = 0; i < n; i++) {
        if (rev[i] < i) {
            std::swap(a[i], a[rev[i]]);
        }
    }
    if (int(roots.size()) < n) {
        int k = __builtin_ctz(roots.size());
        roots.resize(n);
        while ((1 << k) < n) {
            Z e = power(Z(3), (P - 1) >> (k + 1));
            for (int i = 1 << (k - 1); i < (1 << k); i++) {
                roots[2 * i] = roots[i];
                roots[2 * i + 1] = roots[i] * e;
            }
            k++;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                Z u = a[i + j];
                Z v = a[i + j + k] * roots[k + j];
                a[i + j] = u + v;
                a[i + j + k] = u - v;
            }
        }
    }
}

void idft(std::vector<Z> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    Z inv = (1 - P) / n;
    for (int i = 0; i < n; i++) {
        a[i] *= inv;
    }
}
```

2.22 KTO (FedShat)

```
struct Eq {// x = a (mod m)
    ll a, m;

    Eq(){};

    Eq(ll a, ll m) : a(a), m(m){};
};

ll binpow(ll a, ll n, ll m) {
    if (n == 0) {
        return 1;
    }
    if (n % 2 == 0) {
        int128_t b = binpow(a, n / 2, m);
        return (b * b) % m;
    }
    int128_t x = binpow(a, n - 1, m);
    return (a * x) % m;
}

ll binpow(ll a, ll n) {
    if (n == 0) {
        return 1;
    }
    if (n % 2 == 0) {
        ll b = binpow(a, n / 2);
        return b * b;
    }
    return a * binpow(a, n - 1);
}

ll phi(ll a) {
    ll d = 2, k = a;
    map<ll, int> cnt;
    while (d * d <= a) {
        if (k % d == 0) {
            k /= d;
            ++cnt[d];
        } else {
            ++d;
        }
    }
    if (k != 1) {
        ++cnt[k];
    }
    ll ans = 1;
    for (auto i : cnt) {
```

```
        ans *= binpow(i.first, i.second - 1) * (i.first - 1);
    }
    return ans;
}

ll gcd(ll a, ll b) {
    return std::gcd(abs(a), abs(b));
}

Eq solve(Eq ai, Eq bi) {
    if (ai.m == -1 || bi.m == -1) {
        return {0, -1};
    }
    ll a = ai.m, b = bi.m, c = ai.a - bi.a;
    ll d = ::gcd(a, b);
    a /= d;
    b /= d;
    if (c % d != 0) {
        return {0, -1};
    }
    c /= d;
    ll x = (((int128_t) -c * (int128_t) binpow(a, phi(b) - 1, b)) % b + b)
     % b;
    x = ((int128_t) ai.m * (int128_t) x + ai.a) % lcm(ai.m, bi.m);
    return {x, lcm(ai.m, bi.m)};
}
```

2.23 Обратные по простому модулю

Пусть дан простой модуль m. Для каждого числа из $[1, m-1]$ найти обратное к нему.

```
r[1] = 1;
for (int i = 2; i < m; ++i) {
    r[i] = (m - (m / i) * r[m % i] % m) % m;
}
```

2.24 Обратные факториалы

```
int inv(int a, int m) {
    if (a == 1)
        return 1;
    return (1 - inv(m % a, a) * m) / a + m;
}

{
    f[0] = 1;
    for (int i = 1; i < N; i++) {
        f[i] = i * f[i - 1] % mod;
    }

    r[N - 1] = inv(f[N - 1]) for (int i = N - 1; i >= 1; i--) {
        r[i - 1] = r[i] * i % mod;
    }
}
```

2.25 Гаусс

```
int gauss(vector<vector<double>> a,
          vector<double> &ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;
    vector<int> where(m, -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 =
            }
        ++row;
        a[i][col] / a[row][col];
        for (int j = col; j <= m; ++j)
            a[i][j] -= a[row][j] * c;
    }
    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;
    }
    for (int i = 0; i < m; ++i)
        if (where[i] == -1) return INF;
    return 1;
}
```

Бинарный

```
int gauss(vector<bitset<N>> a, int n, int m,
          bitset<N> &ans) {
    vector<int> where(m, -1);
    for (int col = 0, row = 0; col < m && row < n; ++col) {
        for (int i = row; i < n; ++i)
            if (a[i][col]) {
                swap(a[i], a[row]);
                break;
            }
        if (!a[row][col]) {
            continue;
        }
        where[col] = row;
        for (int i = 0; i < n; ++i) {
            if (i != row && a[i][col]) a[i] ^= a[row];
        }
        ++row;
    }
}
```

2.26 Быстрая факторизация (FedShat)

```
ll binpow(ll a, ll n, ll mod) {
    if (n == 0) {
        return 1;
    }
    if (n % 2 == 0) {
        int128_t b = binpow(a, n / 2, mod);
        return (b * b) % mod;
    }
    return (((int128_t) a) * binpow(a, n - 1, mod)) % mod;
}

constexpr int N = 1e7;
vector<int> pr, lp;

bool prime(ll n) {
    if (n <= N) {
        return binary_search(all(pr), n);
    }
    int iter = 60;
    int s = 0;
    ll d = n - 1;
    while (d % 2 == 0) {
        d /= 2;
        ++s;
    }
    auto test = [&](ll a) {
        if (binpow(a, d, n) == 1) {
            return true;
        }
        ll _2r = 1;
        for (int r = 0; r < s; ++r) {
            auto tmp = binpow(binpow(a, d, n), _2r, n);
            if (tmp == n - 1) {
                return true;
            }
            _2r *= 2;
        }
        return false;
    };
    for (int _ = 0; _ < iter; ++_) {
        ll a = uniform_int_distribution<ll>(1, n - 1)(rnd);
        if (!test(a)) {
            return false;
        }
    }
    return true;
}

ll f(ll x, ll n) {
    return ((int128_t) x * (int128_t) x + (int128_t) 2) % n;
}

ll pollard(ll n) {
    ll a = uniform_int_distribution<ll>(0, n - 1)(rnd);
```

```
    ll x = a, y = a, d = 1;
    constexpr int iter = 5e4;
    for (int _ = 0; _ < iter; ++_) {
        x = f(f(x, n), n);
        y = f(y, n);
        d = gcd(abs(x - y), n);
        if (d != 1 && d != n) {
            break;
        }
    }
    if (d == 1 || d == n) {
        pollard(n);
    }
    return d;
}

vector<ll> res;

void factor(ll n) {
    if (n <= N) {
        while (true) {
            if (lp[n] == 0) {
                break;
            }
            res.push_back(lp[n]);
            n /= lp[n];
        }
        return;
    }
    if (prime(n)) {
        res.push_back(n);
        return;
    }
    ll d = pollard(n);
    factor(n / d);
    factor(d);
}

void solve() {
    ll n;
    cin >> n;
    lp.resize(N + 1);
    for (int i = 2; i <= N; ++i) {
        if (lp[i] == 0) {
            lp[i] = i;
            pr.push_back(i);
        }
        for (int j = 0; j < (int) pr.size() && pr[j] <= lp[i] && i * pr[j]
     <= N; ++j) {
            lp[i * pr[j]] = pr[j];
        }
    }
    factor(n);
}
```

2.27 Префикс-функция

```
vector<int> prefix_function(string &s) {
    int n = (int) s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; ++i) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j])
            j = pi[j - 1];
        if (s[i] == s[j]) {
            ++j;
        }
        pi[i] = j;
    }
    return pi;
}
```

2.28 Z-функция

```
vector<int> z_function(string &s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r) {
            z[i] = min(r - i + 1, z[i - l]);
        }
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            ++z[i];
        }
        if (i + z[i] - 1 > r) {
            l = i;
            r = i + z[i] - 1;
        }
    }
    return z;
}
```

2.29 Суфмас (TheEvilBird)

```
int mod(int x, int m) {
    if (x < 0) x += m;
    if (x >= m) x -= m;
    return x;
}

vector<int> sufix_array(string s) {
    s += '$';
    int n = sz(s);
    vector<int> arr(n), narr(n), head(n), c(n), nc(n);
    for (int i = 0; i < n; ++i) {
        arr[i] = i;
    }
    sort(all(arr), [&](int x, int y) {
        return s[x] < s[y];
    });
    int cl = 0;
    c[arr[0]] = cl;
    head[0] = 0;
    for (int i = 1; i < n; ++i) {
        if (s[arr[i]] != s[arr[i - 1]]) {
            head[++cl] = i;
        }
        c[arr[i]] = cl;
    }
    for (int k = 1; k < n && cl < n; k *= 2) {
        for (int i = 0; i < n; i++) {
            int j = mod(arr[i] - k, n);
            narr[head[c[j]]++] = j;
        }
        cl = 0;
        head[0] = 0;
        pii prev = {c[narr[0]], c[mod(narr[0] + k, n)]};
        for (int i = 1; i < n; i++) {
            pii cur = {c[narr[i]], c[mod(narr[i] + k, n)]};
            if (cur != prev) {
                head[++cl] = i;
            }
            nc[narr[i]] = cl;
            prev = cur;
        }
        swap(arr, narr);
        swap(c, nc);
    }
    // returns sufix array without $
    return vector<int>(arr.begin() + 1, arr.end());
}

vector<int> build_lcp(string &s, vector<int> &suf) {
    int n = sz(s);
    vector<int> lcp(n - 1), order(n);
    for (int i = 0; i < n; ++i) {
        order[suf[i]] = i;
    }
    int l = 0;
    for (int i = 0; i < n; ++i) {
        int id = order[i];
        if (id + 1 == n) {
            l = 0;
            continue;
        }
        int j = suf[id + 1];
        if (l) --l;
        while (max(i + l, j + l) < n &&
               s[i + l] == s[j + l]) {
            ++l;
        }
        lcp[id] = l;
    }
    return lcp;
}
```

2.30 Суфавтомат (TheEvilBird)

```
struct Node {
    int go[26];
    int suf, prev, term, len;

    Node() {
        for (auto &i : go) {
            i = -1;
        }
        len = 0;
        suf = -1;
```

```
        prev = -1;
        term = 0;
    }
};

vector<Node> automat;

int add(int a, int ch) {
    int b = sz(automat);
    automat.emplace_back();
    automat[b].prev = a;
    automat[b].suf = 0;
    automat[b].len = automat[a].len + 1;
    for (; a != -1; a = automat[a].suf) {
        if (automat[a].go[ch] == -1) {
            automat[a].go[ch] = b;
            continue;
        }
        int c = automat[a].go[ch];
        if (automat[c].prev == a) {
            automat[b].suf = c;
            break;
        }
        int d = sz(automat);
        automat.emplace_back();
        automat[d].suf = automat[c].suf;
        automat[d].len = automat[a].len + 1;
        automat[c].suf = d;
        automat[b].suf = d;
        automat[d].prev = a;
        for (int i = 0; i < 26; ++i) {
            automat[d].go[i] = automat[c].go[i];
        }
        for (; a != -1 && automat[a].go[ch] == c; a = automat[a].suf) {
            automat[a].go[ch] = d;
        }
        break;
    }
    // returns id of the added vertex
    return b;
}
```

2.31 Ахо-Корасик (Sweezyk)

```
struct Node {
    int par;
    int par_c;
    int go[26];
    int term;
    int link;
    int super;
    int cnt;
};

const int N = 1e6 + 5;
int ptr = 1;
Node trie[N];

void add(string s) {
    int cur = 1;
    for (auto &q : s) {
        int c = q - 'a';
        if (trie[cur].go[c]) {
            cur = trie[cur].go[c];
        } else {
            ++ptr;
            trie[cur].go[c] = ptr;
            trie[ptr].par = cur;
            trie[ptr].par_c = c;
            cur = ptr;
        }
    }
    trie[cur].cnt++;
}

void build() {
    queue<int> q;
    q.push(1);
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        if (v != 1) {
            if (trie[v].par == 1) {
                trie[v].link = trie[v].super = 1;
            } else {
                trie[v].link = trie[trie[trie[v].par].link].go[trie[v].
    par_c];
                trie[v].super = (trie[trie[v].link].cnt ? trie[v].link :
    trie[trie[v].link].super);
            }
        }
        trie[v].cnt += trie[trie[v].link].cnt;
        for (int c = 0; c < 26; c++) {
            if (trie[v].go[c]) {
                q.push(trie[v].go[c]);
            } else {
                if (v == 1) {
                    trie[v].go[c] = 1;
                } else {
                    trie[v].go[c] = trie[trie[v].link].go[c];
                }
            }
        }
    }
}

trie[1].link = trie[1].super = trie[1].par = 1;
build();
```

2.32 Манакер

```
vector<int> d1(n);
int l = 0, r = -1;
for (int i = 0; i < n; ++i) {
    int k = (i > r ? 1 : min(d1[l + r - i], r - i + 1));
    while (i + k < n && i - k >= 0 && s[i + k] == s[i - k]) {
        ++k;
    }
    d1[i] = k;
    if (i + k - 1 > r) {
        l = i - k + 1;
        r = i + k - 1;
    }
}
vector<int> d2(n);
l = 0;
r = -1;
for (int i = 0; i < n; ++i) {
    int k = (i > r ? 0 : min(d2[l + r - i + 1], r - i + 1));
    while (i + k < n && i - k - 1 >= 0 && s[i + k] == s[i - k - 1]) {
        ++k;
    }
    d2[i] = k;
    if (i + k - 1 > r) {
        l = i - k;
        r = i + k - 1;
    }
}
```

2.33 CHT (FedShat)

```
struct Line {
    ll k = 0, b = -INFLL;
    Line() = default;
    Line(ll k, ll b) : k(k), b(b){};

    ld operator()(ld x) {
        return k * x + b;
    }
};

bool operator<(Line a, Line b) {
    return a.k < b.k || (a.k == b.k && a.b > b.b);
}

bool operator<(pair<Line, ld> a, pair<Line, ld> b) {
    return a.second < b.second;
}

struct CHT {
    vector<pair<Line, ld>> convex;

    void add(Line a) {
        while (!convex.empty() && a(convex.back().second) > convex.back().
     first(convex.back().second)) {
            convex.pop_back();
        }
        ld xn = -INFLL;
        if (!convex.empty()) {
            xn = (a.b - convex.back().first.b + 0.0) / (convex.back().first
     .k - a.k);
        }
        convex.push_back({a, xn});
    }

    CHT(vector<Line> lines) {
        sort(all(lines));
        for (int i = 0; i < (int) lines.size();) {
            int j = i;
            while (j < (int) lines.size() && lines[i].k == lines[j].k) {
                ++j;
            }
```

```
            add(lines[i]);
            i = j;
        }
    }

    ld get(ld x) {
        auto it = upper_bound(all(convex), pair(Line(), x)) - convex.begin
     ();
        return convex[it - 1].first(x);
    }
};
```

```
void debug_out(Head H, Tail... T) {
    cerr << " " << to_string(H);
    debug_out(T...);
}

#ifdef LOCAL
#define debug(...) cerr << "[" << #__VA_ARGS__ << "]:", debug_out(
    __VA_ARGS__)
#else
#define debug(...) 42
#endif
```

2.34 Дебаг Туриста

```
template<typename A, typename B>
string to_string(pair<A, B> p);

template<typename A, typename B, typename C>
string to_string(tuple<A, B, C> p);

template<typename A, typename B, typename C, typename D>
string to_string(tuple<A, B, C, D> p);

string to_string(const string &s) {
    return '"' + s + '"';
}

string to_string(const char *s) {
    return to_string((string) s);
}

string to_string(bool b) {
    return (b ? "true" : "false");
}

string to_string(vector<bool> v) {
    bool first = true;
    string res = "{";
    for (int i = 0; i < static_cast<int>(v.size()); i++) {
        if (!first) {
            res += ", ";
        }
        first = false;
        res += to_string(v[i]);
    }
    res += "}";
    return res;
}

template<size_t N>
string to_string(bitset<N> v) {
    string res = "";
    for (size_t i = 0; i < N; i++) {
        res += static_cast<char>('0' + v[i]);
    }
    return res;
}

template<typename A>
string to_string(A v) {
    bool first = true;
    string res = "{";
    for (const auto &x : v) {
        if (!first) {
            res += ", ";
        }
        first = false;
        res += to_string(x);
    }
    res += "}";
    return res;
}

template<typename A, typename B>
string to_string(pair<A, B> p) {
    return "(" + to_string(p.first) + ", " + to_string(p.second) + ")";
}

template<typename A, typename B, typename C>
string to_string(tuple<A, B, C> p) {
    return "(" + to_string(get<0>(p)) + ", " + to_string(get<1>(p)) + ", "
     + to_string(get<2>(p)) + ")";
}

template<typename A, typename B, typename C, typename D>
string to_string(tuple<A, B, C, D> p) {
    return "(" + to_string(get<0>(p)) + ", " + to_string(get<1>(p)) + ", "
     + to_string(get<2>(p)) + ", " + to_string(get<3>(p)) + ")";
}

void debug_out() { cerr << endl; }

template<typename Head, typename... Tail>
```

2.35 Геометрия (TheEvilBird)

```
const long double PI = 3.14159265358979323846264338327950;
const long double EPS = 1e-7;

/*
 * Rotate relative to the origin, a - rotation angle:
 * new_x = x cos(a) - y sin(a)
 * new_y = x sin(a) + y cos(a)
 *
 * new_x = x cos(a) + y sin(a)
 * new_y = x sin(a) - y cos(a)
 *
 *
 * rad = degree * PI / 180
 * degree = rad * 180 / PI
*/

struct Point {
    ll x = 0, y = 0;
    // int id = -1;

    Point() : x(0), y(0) {}
    Point(ll _x, ll _y) : x(_x), y(_y) {}

    bool operator==(const Point other) const {
        return x == other.x && y == other.y;
    }

    void operator*=(ll k) {
        x *= k;
        y *= k;
    }
};

istream &operator>>(istream &is, Point &point) {
    is >> point.x >> point.y;
    return is;
}

ostream &operator<<(ostream &os, const Point &point) {
    os << point.x << ' ' << point.y;
    return os;
}

struct Vec {
    ll x, y;

    Vec() : x(0), y(0) {}
    Vec(ll _x, ll _y) : x(_x), y(_y) {}
    Vec(Point a) : x(a.x), y(a.y) {}
    Vec(Point a, Point b) : x(b.x - a.x), y(b.y - a.y) {}

    ld len() const {
        return sqrtl(x * x + y * y);
    }

    ll len_sq() const {
        return (x * x + y * y);
    }

    Vec operator*(ll k) const {
        return {x * k, y * k};
    }

    void operator*=(ll k) {
        x *= k;
        y *= k;
    }

    Vec operator+(const Vec other) const {
        return {x + other.x, y + other.y};
    }

    Vec operator-(const Vec other) const {
        return {x - other.x, y - other.y};
    }

    bool operator==(const Vec other) const {
        return x == other.x && y == other.y;
    }
```

```
    bool operator!=(const Vec other) const {
        return !(*this == other);
    }

    ll operator*(const Vec other) const {// dot product
        return x * other.x + y * other.y;
    }

    ll operator%(const Vec other) const {// cross product
        return x * other.y - y * other.x;
    }
};

istream &operator>>(istream &is, Vec &vec) {
    is >> vec.x >> vec.y;
    return is;
}

ostream &operator<<(ostream &os, const Vec &vec) {
    os << vec.x << ' ' << vec.y;
    return os;
}

typedef vector<Point> Polygon;
typedef vector<Vec> vPolygon;

bool cmp_vectors(Vec a, Vec b) {
    if ((a.y < 0 || (a.y == 0 && a.x < 0)) &&
        (b.y > 0 || (b.y == 0 && b.x > 0))) {
        return true;
    }
    if ((b.y < 0 || (b.y == 0 && b.x < 0)) &&
        (a.y > 0 || (a.y == 0 && a.x > 0))) {
        return false;
    }
    return (a % b > 0 || (a % b == 0 && a.len_sq() < b.len_sq()));
}

int get_sign(ld kek) {
    if (kek > 0) return 1;
    if (kek < 0) return -1;
    return 0;
}

Polygon build_convex_hull(Polygon &a) {
    int n = sz(a);
    for (int i = 1; i < n; ++i) {
        if ((a[i].y < a[0].y) || (a[i].y == a[0].y && a[i].x < a[0].x))
     swap(a[0], a[i]);
    }
    sort(a.begin() + 1, a.end(), [&](Point A, Point B) {
        Vec oa(a[0], A), ob(a[0], B);
        if ((oa % ob) == 0) return oa.len_sq() < ob.len_sq();
        return (oa % ob) > 0;
    });
    Polygon hull = {a[0]};
    for (int i = 1; i < n; ++i) {
        while (sz(hull) >= 2) {
            Vec ab(hull[sz(hull) - 2], hull[sz(hull) - 1]), bp(hull[sz(hull
     ) - 1], a[i]);
            if ((ab % bp) <= 0) {
                hull.pop_back();
            } else {
                break;
            }
        }
        hull.emplace_back(a[i]);
    }
    return hull;
}

ll area_of_polygon(Polygon &poly) {
    ll res = 0;
    int n = sz(poly);
    for (int i = 2; i < n; ++i) {
        Vec ab(poly[0], poly[i - 1]), ac(poly[0], poly[i]);
        res += (ab % ac);
    }
    // don't forget to divide the result by 2!
    return res;
}

ld perimeter_of_polygon(Polygon &poly) {
    ld res = 0;
    int n = sz(poly);
    for (int i = 0; i < n; ++i) {
        Vec v(poly[i], poly[(i + 1 == n ? 0 : i + 1)]);
        res += v.len();
    }
    return res;
}

ll diameter_of_polygon(Polygon &poly) {
    int n = sz(poly), x = 1;
    ll ans = 0;
    for (int i = 0; i < n; ++i) {
        int j = (i + 1) % n;
        while (true) {
            Vec ac(poly[i], poly[x]), ab(poly[i], poly[j]), cd(poly[x],
     poly[(x + 1) % n]);
            ans = max(ans, ac.len_sq());
            if ((ab % cd) <= 0) {
                break;
            }
            x++;
            x %= n;
        }
    }
    // don't forget to extract root!
    return ans;
}

ld angle(const Vec &a, const Vec &b) {
    return fabsl(atan2(a % b, a * b) / PI * 180);
}

ld angle_rad(const Vec &a, const Vec &b) {
    return (atan2(a % b, a * b));
}

ld angle_rad(const Point &a) {
    return atan2(a.y, a.x);
}

ld from_point_to_line(const Point &p, const Point &a, const Point &b) {
    Vec ba(b, a), ap(a, p);
    return fabs((ba % ap) / ba.len());
}

ld from_point_to_ray(const Point &p, const Point &a, const Point &b) {
    Vec ba(b, a), ap(a, p), ab(a, b);
    if ((ab * ap) < 0) {
        return ap.len();
    } else {
        return fabs((ba % ap) / ba.len());
    }
}

ld from_point_to_segment(const Point &p, const Point &a, const Point &b) {
    Vec ab(a, b), ap(a, p), bp(b, p), ba(b, a);
    if ((ab * ap) < 0) {
        return fabsl(ap.len());
    } else if ((ab * bp) > 0) {
        return fabsl(bp.len());
    } else {
        return fabsl((ba % ap) / ba.len());
    }
}

bool point_on_line(const Point &p, const Point &a, const Point &b) {
    Vec ab(a, b), ap(b, p);
    return (ab % ap) == 0;
}

bool point_on_ray(const Point &p, const Point &a, const Point &b) {
    Vec ab(a, b), ap(a, p);
    return ((ab % ap) == 0 && (ab * ap) > 0 || (a == p));
}

bool point_on_segment(const Point &p, const Point &a, const Point &b) {
    Vec ap(a, p), bp(b, p), ab(a, b);
    return ((ap * bp) <= 0 && (ap % ab) == 0);
}

bool point_in_angle(const Point &p, const Point &a, const Point &o, const
     Point &b) {
    Vec oa(o, a), ob(o, b), op(o, p);
    if ((oa % ob) < 0) swap(oa, ob);
    return ((oa % op) >= 0 && (ob % op) <= 0);
}

bool segment_intersection(const Point &a, const Point &b, const Point &c,
     const Point &d) {
    Vec ab(a, b), cd(c, d), ac(a, c), ad(a, d), cb(c, b), ca(c, a);
    if (get_sign((ab % ac)) * get_sign((ab % ad)) <= 0 && get_sign((cd % ca
     )) * get_sign((cd % cb)) <= 0) {
        ll x1 = max(min(a.x, b.x), min(c.x, d.x)), x2 = min(max(a.x, b.x),
     max(c.x, d.x));
        ll y1 = max(min(a.y, b.y), min(c.y, d.y)), y2 = min(max(a.y, b.y),
     max(c.y, d.y));
        return (x1 <= x2 && y1 <= y2);
    }
    return false;
}

bool rays_intersection(const Point &a, const Point &b, const Point &c,
     const Point &d) {
    Vec ab(a, b), cd(c, d);
    ll k = 1e6;
    ab *= k;
    cd *= k;
```

```
    Point nb(ab.x + a.x, ab.y + a.y), nd(cd.x + c.x, cd.y + c.y);
    return segment_intersection(a, nb, c, nd);
}

bool lines_intersection(const Point &a, const Point &b, const Point &c,
     const Point &d) {
    Vec ab(a, b), cd(c, d);
    return ((ab % cd) != 0);
}

int point_in_polygon(const Point &p, const Polygon &poly) {
    // 0 - outside, 1 - inside, 2 - border;
    int n = sz(poly);
    if (point_in_angle(p, poly[n - 1], poly[0], poly[1])) {
        int l = 1, r = n - 1;
        while (r - l > 1) {
            int md = (l + r) / 2;
            if (point_in_angle(p, poly[md], poly[0], poly[l])) {
                r = md;
            } else {
                l = md;
            }
        }
        if (point_in_angle(p, poly[r], poly[0], poly[l]) &&
            point_in_angle(p, poly[0], poly[l], poly[r]) &&
            point_in_angle(p, poly[l], poly[r], poly[0])) {
            if (point_on_segment(p, poly[l], poly[r]) ||
                point_on_segment(p, poly[(l ? l - 1 : n - 1)], poly[l]) ||
                point_on_segment(p, poly[r], poly[(r + 1 == n ? 0 : r + 1)
     ])) {
                return 2;
            } else {
                return 1;
            }
        } else {
            return 0;
        }
    } else {
        return 0;
    }
}

int point_in_nonconvex_polygon(const Point &p, const Polygon &poly) {
    // 0 - outside, 1 - inside, 2 - border;
    int n = sz(poly);
    for (int i = 0; i < n; ++i) {
        if (point_on_segment(p, poly[i], poly[(i + 1) % n])) {
            return 2;
        }
    }
    ld s = 0.0;
    for (int i = 0; i < n; ++i) {
        Vec pa(p, poly[i]), pb(p, poly[(i + 1 == n ? 0 : i + 1)]);
        s += angle_rad(pa, pb);
    }
    if (s >= PI || s <= -PI) {
        return 1;
    } else {
        return 0;
    }
}

Polygon minkowski_sum(Polygon &a, Polygon &b) {
    // a[0], b[0]: y - max, y1 = y2 => x - max. Against clockwise
    int n = sz(a), m = sz(b);
    assert(n >= 3 && m >= 3);
    Point high_a = a[0], high_b = b[0];
    vPolygon va(n), vb(m);
    for (int i = 0; i < n; ++i) {
        va[i] = Vec(a[i], a[(i + 1) % n]);
    }
    for (int i = 0; i < m; ++i) {
        vb[i] = Vec(b[i], b[(i + 1) % m]);
    }
    //    sort(all(va), cmp_vectors);
    //    sort(all(vb), cmp_vectors);
    vPolygon vc;
    merge(all(va), all(vb), back_inserter(vc), cmp_vectors);
    Point high_c(high_a.x + high_b.x, high_a.y + high_b.y);
    Polygon c(sz(vc) + 1);
    c[0] = high_c;
    for (int i = 0; i < sz(c) - 1; ++i) {
        c[i + 1] = Point(c[i].x + vc[i].x, c[i].y + vc[i].y);
    }
    return c;
}

ld from_polygon_to_polygon(Polygon a, Polygon b) {
    for (auto &i : b) {
        i *= -1;
    }
    int pos = 0;
    for (int i = 1; i < sz(b); ++i) {
        if ((b[i].y > b[pos].y) ||
            (b[i].y == b[pos].y && b[i].x > b[pos].x)) {
            pos = i;
        }
    }
    rotate(b.begin(), b.begin() + pos, b.end());
    Polygon c = minkowski_sum(a, b);
    int n = sz(c);
    Point p(0, 0);
    ld ans = 1e20;
    for (int i = 0; i < n - 1; ++i) {
        ans = min(ans, from_point_to_segment(p, c[i], c[i + 1]));
    }
    return ans;
}

ll diameter_of_polygon_minkowski(Polygon &a) {
    Polygon ra = a;
    for (auto &i : ra) {
        i *= -1;
    }
    int pos = 0;
    for (int i = 1; i < sz(a); ++i) {
        if ((a[i].y > a[pos].y) ||
            (a[i].y == a[pos].y && a[i].x > a[pos].x)) {
            pos = i;
        }
    }
    rotate(a.begin(), a.begin() + pos, a.end());
    pos = 0;
    for (int i = 1; i < sz(a); ++i) {
        if ((ra[i].y > ra[pos].y) ||
            (ra[i].y == ra[pos].y && ra[i].x > ra[pos].x)) {
            pos = i;
        }
    }
    rotate(ra.begin(), ra.begin() + pos, ra.end());
    Polygon c = minkowski_sum(a, ra);
    int n = sz(c);
    ll ans = 0;
    for (int i = 0; i < n; ++i) {
        ans = max(ans, Vec(c[i]).len_sq());
    }
    // don't forget to extract root!
    return ans;
}
```