

Team notebook

one more try

November 25, 2022



Contents

1	Data Structures	1
1.1	2D BIT With Sparse SegTree	1
1.2	2D BIT with ordered _{set}	1
1.3	BIT with range updates and point queries	2
1.4	Binary Indexed Tree	2
1.5	DSU	2
1.6	Lazy SegTree	3
1.7	Mergeable Segment Tree	3
1.8	Minimum Deque	4
1.9	Persistent SegTree	4
1.10	SparseTable	5
1.11	Treap	5
2	Geometry	6
2.1	Convex Hull Trick	6
2.2	Convex Hull	7
2.3	Dynamic Convex Hull Trick	7
2.4	Point	8
2.5	Rotational Sweep	9
3	Graphs	10
3.1	2-Edge Connected Components	10
3.2	BCC	10

3.3	Bipartite Matching	11
3.4	Dijkstra	11
3.5	Dinic	11
3.6	Edmonds-Karp	12
3.7	Eulerian Tour	13
3.8	Hungarian	13
3.9	Min Cost Max Flow	14
3.10	Tarjan SCC	14
4	Math	15
4.1	Basis	15
4.2	CatalanNumbers	15
4.3	Extended Euclidean Algorithm	15
4.4	FFT	15
4.5	Fraction	16
4.6	Gauss	16
4.7	Integration	17
4.8	Matrix Struct	17
4.9	Modulo operations	18
4.10	PermutationExp	18
5	Misc	18
5.1	Custom Vector	18
6	Notes	19
6.1	Notes	19
7	Strings	19
7.1	Bitset Trie	19
7.2	Hashing	19
7.3	Hashing2	21
7.4	KMP	22
7.5	SuffixArray	22

8	Trees	23
8.1	Centroid Decomposition	23
8.2	HeavyLight	24
8.3	Lowest Common Ancestor	24

1 Data Structures

1.1 2D BIT With Sparse SegTree

```

int root[maxn], seg[maxn * 100], L[maxn * 100], R[maxn * 100], nxt_id;
int modify(int id, int pos, int val, int l = 1, int r = n) {
    if (!id) {
        id = ++nxt_id;
    }
    if (l == r) {
        seg[id] = val;
        return id;
    }
    int mid = (l + r) / 2;
    if (pos <= mid) {
        L[id] = modify(L[id], pos, val, l, mid);
    } else {
        R[id] = modify(R[id], pos, val, mid + 1, r);
    }
    seg[id] = seg[L[id]] + seg[R[id]];
    return id;
}
int query(int id, int x, int y, int l = 1, int r = n) {
    if (l > y || r < x || !id) {
        return 0;
    }
    if (l >= x && r <= y) {
        return seg[id];
    }
    int mid = (l + r) / 2;
    return query(L[id], x, y, l, mid) + query(R[id], x, y, mid + 1, r);
}
void add(int x, int y) {
    for (; x <= n; x += x & -x) {
        root[x] = modify(root[x], y, 1);
    }
}
int get(int x, int lo, int hi) {
    int ret = 0;
    for (; x > 0; x -= x & -x) {
        ret += query(root[x], lo, hi);
    }
    return ret;
}
int get(int l, int r, int lo, int hi) {
    return get(r, lo, hi) - get(l - 1, lo, hi);
}

```

1.2 2D BIT with ordered_{set}

```

#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>;

struct BIT {
    ordered_set<int> f[maxn];
    void add(int i, int val) {
        for(; i <= maxn; i += i & -i)
            f[i].insert(val);
    }
    int get(int i, int lo, int hi) {
        int ret = 0;
        for (; i > 0; i -= i & -i) {
            ret += f[i].order_of_key(hi + 1) - f[i].order_of_key(lo);
        }
        return ret;
    }
    int get(int l, int r, int lo, int hi) {
        return get(r, lo, hi) - get(l - 1, lo, hi);
    }
};

```

1.3 BIT with range updates and point queries

```

/* Description: BIT with range updates and point queries */
#include <bits/stdc++.h>
#define pb push_back
#define all(a) a.begin(), a.end()
#define sz(a) (int)a.size()
#define x first
#define y second
using namespace std;
typedef long long ll;
typedef long double ld;
typedef pair<int, int> pii;
const int maxn = 1e5 + 100;

template<class T, int N>
struct BIT {
    T f[N + 1];
    BIT() {
        memset(f, 0, sizeof f);
    }
    void upd(int pos, T val) {
        for (; pos <= N; pos += pos & -pos)
            f[pos] += val;
    }
    void upd(int l, int r, T val) {

```

```

        upd(1, val);
        upd(r + 1, -val);
    }
    T query(int pos) {
        T ret = 0;
        for (; pos > 0; pos -= pos & -pos) {
            ret += f[pos];
        }
        return ret;
    }
};

int main()
{
    ios_base::sync_with_stdio(false), cin.tie(0);
    BIT<ll, 100000> tree;
    tree.upd(3, 5, 1);
    cout << tree.query(3) << "\n";
    cout << tree.query(4) << "\n";
    cout << tree.query(5) << "\n";
    cout << tree.query(2) << "\n";
    cout << tree.query(6) << "\n";
    return 0;
}

```

1.4 Binary Indexed Tree

```

template<class T, int N>
struct BIT {
    T f[N + 1];
    BIT() {
        memset(f, 0, sizeof f);
    }
    void upd(int pos, T val) {
        for (; pos <= N; pos += pos & -pos)
            f[pos] += val;
    }
    T query(int r) {
        T ret = 0;
        for (; r > 0; r -= r & -r) {
            ret += f[r];
        }
        return ret;
    }
    T query(int l, int r) {
        return query(r) - query(l - 1);
    }
};

```

1.5 DSU

```

template<int N>
struct DSU {
    int par[N + 1], sz[N + 1];
    DSU() {
        for (int i = 0; i <= N; i++) {
            par[i] = i, sz[i] = 1;
        }
    }
    int root(int v) {
        return v == par[v] ? v : par[v] = root(par[v]);
    }
    bool unite(int a, int b) {
        a = root(a);
        b = root(b);
        if (a == b) {
            return false;
        }
        if (sz[a] < sz[b]) {
            swap(a, b);
        }
        sz[a] += sz[b];
        par[b] = a;
        return true;
    }
};

```

1.6 Lazy SegTree

```

#include <bits/stdc++.h>
#define pb push_back
#define all(a) a.begin(), a.end()
#define sz(a) (int)a.size()
#define x first
#define y second
using namespace std;
typedef long long ll;
typedef long double ld;
typedef pair<int, int> pii;
const int maxn = 1e5 + 100;

template<class T, int N>
struct SegTree {
    T seg[N * 4], lazy[N * 4];
    SegTree() {
        memset(seg, 0, sizeof seg);
        memset(lazy, 0, sizeof lazy);
    }
    void build(T arr[], int id = 1, int l = 0, int r = N - 1) {
        if (r == l) {
            seg[id] = arr[l];
            return;
        }
    }
};

```

```

    int mid = (l + r) / 2;
    build(arr, id * 2, l, mid);
    build(arr, id * 2 + 1, mid + 1, r);
    seg[id] = seg[id * 2] + seg[id * 2 + 1];
}
void upd(int id, int l, int r, T val) {
    seg[id] += (T)(r - l + 1) * val;
    lazy[id] += val;
}
void shift(int id, int l, int r) {
    int mid = (l + r) / 2;
    upd(id * 2, l, mid, lazy[id]);
    upd(id * 2 + 1, mid + 1, r, lazy[id]);
    lazy[id] = 0;
}
void modify(int x, int y, T val, int id = 1, int l = 0, int r = N - 1) {
    if (l > y || r < x) {
        return;
    }
    if (l >= x && r <= y) {
        upd(id, l, r, val);
        return;
    }
    shift(id, l, r);
    int mid = (l + r) / 2;
    modify(x, y, val, id * 2, l, mid);
    modify(x, y, val, id * 2 + 1, mid + 1, r);
    seg[id] = seg[id * 2] + seg[id * 2 + 1];
}
T query(int x, int y, int id = 1, int l = 0, int r = N - 1) {
    if (l > y || r < x) {
        return 0;
    }
    if (l >= x && r <= y) {
        return seg[id];
    }
    shift(id, l, r);
    int mid = (l + r) / 2;
    return query(x, y, id * 2, l, mid) + query(x, y, id * 2 + 1, mid + 1, r);
}
};

```

1.7 Mergeable Segment Tree

```

#include <bits/stdc++.h>
#define pb push_back
#define all(a) a.begin(), a.end()
#define sz(a) (int)a.size()
#define x first
#define y second
#define debug(...) cout << "[" << __VA_ARGS__ << ": " << __VA_ARGS__ << "]\n"
#define rd() abs((int)rng())

```

```

using namespace std;
typedef long long ll;
typedef long double ld;
typedef pair<int, int> pii;
const int maxn = 2e5 + 100;
const int mod = 1e9 + 7;
mt19937 rng(chrono::high_resolution_clock::now().time_since_epoch().count());

struct node {
    int l, r;
};
int n, q, ans[maxn], nxt_id;
node tree[maxn * 150];
int root[105];
pii split(int id, int k, int lo = 1, int hi = n) { /// >= goes to the right
    if (!id) {
        return {0, 0};
    }
    if (lo == hi) {
        return (lo >= k ? pii(0, id) : pii(id, 0));
    }
    int other = ++nxt_id;
    int mid = (lo + hi) / 2;
    if (k <= mid) {
        pii p = split(tree[id].l, k, lo, mid);
        tree[id].l = p.y;
        tree[other].l = p.x;
        return {other, id};
    } else {
        pii p = split(tree[id].r, k, mid + 1, hi);
        tree[id].r = p.x;
        tree[other].r = p.y;
        return {id, other};
    }
}
int merge(int a, int b) {
    if (!a) return b;
    if (!b) return a;
    tree[a].l = merge(tree[a].l, tree[b].l);
    tree[a].r = merge(tree[a].r, tree[b].r);
    return a;
}
int ins(int id, int pos, int lo = 1, int hi = n) {
    if (!id) id = ++nxt_id;
    if (lo == hi) return id;
    int mid = (lo + hi) / 2;
    if (pos <= mid) {
        tree[id].l = ins(tree[id].l, pos, lo, mid);
    } else {
        tree[id].r = ins(tree[id].r, pos, mid + 1, hi);
    }
    return id;
}
void go(int id, int x, int lo = 1, int hi = n) {
    if (!id) {

```

```

        return;
    }
    if (lo == hi) {
        ans[lo] = x;
        return;
    }
    int mid = (lo + hi) / 2;
    go(tree[id].l, x, lo, mid);
    go(tree[id].r, x, mid + 1, hi);
}
int main()
{
    ios_base::sync_with_stdio(false), cin.tie(0);
    cin >> n;
    for (int i = 1; i <= n; i++) {
        int x;
        cin >> x;
        root[x] = ins(root[x], i);
    }
    cin >> q;
    for (int i = 0; i < q; i++) {
        int l, r, x, y;
        cin >> l >> r >> x >> y;
        if (x == y) {
            continue; /// DOESNT WORK IF X = Y
        }
        pii a = split(root[x], l);
        pii b = split(a.y, r + 1);
        root[y] = merge(root[y], b.x);
        root[x] = merge(a.x, b.y);
    }
    for (int i = 1; i <= 100; i++) {
        go(root[i], i);
    }
    for (int i = 1; i <= n; i++) {
        cout << ans[i] << " ";
    }
    cout << "\n";
    return 0;
}

```

1.8 Minimum Deque

```

struct MinDeque {
    deque<pii> D;
    int lo, hi;
    MinDeque() {
        lo = 1, hi = 1;
    }
    void ins(int val) {
        while (sz(D) && D.back().x >= val) {
            D.pop_back();
        }
    }
}

```

```

    }
    D.pb({val, hi++});
}
void del() {
    if (sz(D) && D.front().y == lo++) {
        D.pop_front();
    }
}
int get() {
    return sz(D) ? D.front().x : mod;
}
};

```

1.9 Persistent SegTree

```

int root[maxn], seg[maxn * 60], L[maxn * 60], R[maxn * 60], next_id; /// required
size can be different depending on problem
void build(int id = 1, int l = 1, int r = n) {
    if (l == r) {
        return;
    }
    int mid = (l + r) / 2;
    L[id] = ++next_id;
    R[id] = ++next_id;
    build(L[id], l, mid);
    build(R[id], mid + 1, r);
}
int modify(int pos, int delta, int id, int l = 1, int r = n) {
    int new_id = ++next_id;
    seg[new_id] = seg[id] + delta;
    if (l == r) {
        return new_id;
    }
    int mid = (l + r) / 2;
    L[new_id] = L[id];
    R[new_id] = R[id];
    if (pos <= mid) {
        L[new_id] = modify(pos, delta, L[new_id], l, mid);
    } else {
        R[new_id] = modify(pos, delta, R[new_id], mid + 1, r);
    }
    return new_id;
}
}

```

1.10 SparseTable

```

/// precompute logs
int lg[MAXN+1];
lg[1] = 0;
for (int i = 2; i <= MAXN; i++)

```

```

    lg[i] = lg[i/2] + 1;
// precompute sparse table
int st[K + 1][MAXN];
std::copy(array.begin(), array.end(), st[0]);
for (int i = 1; i <= K; i++)
    for (int j = 0; j + (1 << i) <= N; j++)
        st[i][j] = min(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
// min query
int i = lg[R - L + 1];
int minimum = min(st[i][L], st[i][R - (1 << i) + 1]);

```

1.11 Treap

```

/* Description: treap with max heap property */
#include <bits/stdc++.h>
#define pb push_back
#define all(a) a.begin(), a.end()
#define sz(a) (int)a.size()
#define x first
#define y second
using namespace std;
typedef long long ll;
typedef long double ld;
typedef pair<int, int> pii;
const int maxn = 1e5 + 100;
mt19937 rng(chrono::high_resolution_clock::now().time_since_epoch().count());

struct node {
    int val, pri, sz;
    node *l, *r;

    node(int v) {
        val = v, sz = 1, pri = rng();
        l = r = NULL;
    }

    void recalc() {
        sz = 1 + (l ? l->sz : 0) + (r ? r->sz : 0);
    }
};
typedef node* pnode;

pair<pnode, pnode> split(pnode t, int v) { /// >= v goes to the right
    if (!t) {
        return {t, t};
    }

    if (v <= t->val) {
        auto p = split(t->l, v);
        t->l = p.second;
        t->recalc();
        return {p.first, t};
    }
}

```

```

    } else {
        auto p = split(t->r, v);
        t->r = p.first;
        t->recalc();
        return {t, p.second};
    }
}

pair<pnode, pnode> split_by_order(pnode t, int v) {
    if (!t) {
        return {t, t};
    }
    int tmp = t->l ? t->l->sz : 0;
    if (v <= tmp) {
        auto p = split_by_order(t->l, v);
        t->l = p.second;
        t->recalc();
        return {p.first, t};
    } else {
        auto p = split_by_order(t->r, v - tmp - 1);
        t->r = p.first;
        t->recalc();
        return {t, p.second};
    }
}

pnode merge(pnode left, pnode right) {
    if (!left) {
        return right;
    }
    if (!right) {
        return left;
    }

    if (left->pri > right->pri) {
        left->r = merge(left->r, right);
        left->recalc();
        return left;
    } else {
        right->l = merge(left, right->l);
        right->recalc();
        return right;
    }
}

pnode ins(pnode x, int v) {
    auto a = split(x, v);
    auto b = split(a.second, v + 1);
    return merge(a.first, merge(new node(v), b.second));
}

pnode del(pnode x, int v) {
    auto a = split(x, v), b = split(a.second, v + 1);
    return merge(a.first, b.second);
}

```

```

pnode root;

int order_of_key(int x) {
    auto a = split(root, x);
    int t = a.first ? a.first->sz : 0;
    root = merge(a.first, a.second);
    return t;
}

int find_by_order(int x) {
    auto a = split_by_order(root, x);
    auto b = split_by_order(a.first, x - 1);
    int t = b.second->val;
    root = merge(merge(b.first, b.second), a.second);
    return t;
}

int main()
{
    ios_base::sync_with_stdio(false), cin.tie(0);
    int Q;
    cin >> Q;
    for (int i = 0; i < Q; i++) {
        char c;
        int d;
        cin >> c >> d;
        if (c == 'I') {
            root = ins(root, d);
        } else if (c == 'D') {
            root = del(root, d);
        } else if (c == 'K') {
            if (!root || root->sz < d) {
                cout << "invalid\n";
            } else {
                cout << find_by_order(d) << "\n";
            }
        } else {
            cout << order_of_key(d) << "\n";
        }
    }
}

```

2 Geometry

2.1 Convex Hull Trick

```

struct line {
    ll k, b;
    line() = default;
    line(ll kk, ll bb) {
        k = kk;
        b = bb;
    }
};

```

```

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

ld inter(line f, line s) {
    return 1.0 * (s.b - f.b) / (f.k - s.k);
}

int siz, it;
line hull[maxn];
void add(line a) {
    while (siz >= 2 && inter(hull[siz - 1], hull[siz - 2]) >= inter(hull[siz - 2], a)) {
        siz--;
    }
    hull[siz++] = a;
}

// general approach is binary search
ll get(ll x) {
    // change to < for maximum
    while (it + 1 < siz && hull[it].eval(x) > hull[it + 1].eval(x)) {
        it++;
    }
    return hull[it].eval(x);
}

```

2.2 Convex Hull

```

#include <bits/stdc++.h>
using namespace std;
using ll = long long;
struct point {
    ll x, y;
    point(ll a = 0, ll b = 0) {
        x = a, y = b;
    }
    bool operator<(const point &o) const {
        return make_pair(x, y) < make_pair(o.x, o.y);
    }
    bool operator==(const point& o) const {
        return x == o.x && y == o.y;
    }
    point operator-(const point &b) const {
        return point(x - b.x, y - b.y);
    }
};

ll cross(point a, point b) {
    return a.x * b.y - a.y * b.x;
}

ll cross(point a, point b, point c) {
    return cross(b - a, c - a);
}

```

```

}
vector<point> convex_hull(vector<point>pts) {
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()), pts.end());
    vector<point>top, bot;
    for (int i = 0; i < pts.size(); ++i) {
        while (bot.size() >= 2 && cross(bot[bot.size() - 2], bot.back(), pts[i])
            <= 0) {
            bot.pop_back();
        }
        bot.push_back(pts[i]);
    }
    bot.pop_back();
    for (int i = pts.size() - 1; i >= 0; --i) {
        while (top.size() >= 2 && cross(top[top.size() - 2], top.back(), pts[i])
            <= 0) {
            top.pop_back();
        }
        top.push_back(pts[i]);
    }
    top.pop_back();
    top.insert(top.end(), bot.begin(), bot.end());
    return top;
}

int main() {
    ios_base::sync_with_stdio(false), cin.tie(0);
    int n;
    cin >> n;
    vector<point>pts;
    for (int i = 0; i < n; i++) {
        point p;
        cin >> p.x >> p.y;
        pts.push_back(p);
    }
    vector<point>hull = convex_hull(pts);
    cout << "-----\n";
    for (point p : hull) {
        cout << p.x << " " << p.y << "\n";
    }
    return 0;
}

```

2.3 Dynamic Convex Hull Trick

```

bool Q;
struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const {
        return Q ? p < o.p : k < o.k;
    }
};

```

```

// Max hull, for minimum make k and m negative
struct LineContainer : multiset<Line> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) { x->p = inf; return false; }
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());
        Q = 1; auto l = *lower_bound({0,0,x}); Q = 0;
        return l.k * x + l.m;
    }
};

```

2.4 Point

```

const ld eps = 1e-9;
int sgn(ld a) {
    return (a > 0) - (a < 0);
}
struct point {
    // Use ll everywhere instead of ld if coordinates are always integers
    ld x, y;
    point(ld a = 0, ld b = 0) {
        x = a, y = b;
    }
    bool operator==(const point& o) const {
        return x == o.x && y == o.y;
    }
    point operator+(const point &b) const {
        return point(x + b.x, y + b.y);
    }
    point operator-(const point &b) const {
        return point(x - b.x, y - b.y);
    }
    point operator*(const ld &a) const {
        return point(x * a, y * a);
    }
    point operator/(const ld &a) const {
        return point(x / a, y / a);
    }
};

```



```

}
ld vlensq() {
    return x * x + y * y;
}
ld vlen() {
    return sqrtl(vlensq());
}
point perp() const {
    return point(-y, x);
}
// normalize vector
point toUnit() {
    return *this / vlen();
}
};
ld dot(point a, point b) {
    return a.x * b.x + a.y * b.y;
}
ld cross(point a, point b) {
    return a.x * b.y - a.y * b.x;
}
point proj(point v, point u) {
    // projection of v onto u
    return u * (dot(v, u) / dot(u, u));
}
// check if line segments AB and CD intersect
bool isect(point a, point b, point c, point d) {
    int A = sgn(cross(c - a, d - a));
    int B = sgn(cross(c - b, d - b));
    if (A * B != 0 && A == B) {
        return false;
    }
    int C = sgn(cross(a - c, b - c));
    int D = sgn(cross(a - d, b - d));
    if (C * D != 0 && C == D) {
        return false;
    }
    return true;
}
// intersection point of line segments AB and CD
point inter(point a, point b, point c, point d) {
    point ab = b - a;
    point cd = d - c;
    ld top = cross(a, ab) - cross(c, ab);
    ld bot = cross(cd, ab);
    // if(cd.x * top % bot != 0) // FLOATING POINT - CHANGE TO DOUBLE IF NECESSARY
    // return point(mod, mod);
    // if(cd.y * top % bot != 0) // FLOATING POINT - CHANGE TO DOUBLE IF NECESSARY
    // return point(mod, mod);
    ld X = c.x + cd.x * top / bot;
    ld Y = c.y + cd.y * top / bot;
    // or alternatively (works with doubles)
    // return c + cd * (top / bot)
    return point(X, Y);
}

```

```

// checks if circles with centers a, b and radiuses r1, r2 intersect
// intersection points are written to parameter out
bool circleInter(point a, point b, double r1, double r2, pair<point, point>* out) {
    if (a == b) { assert(r1 != r2); return false; }
    point vec = b - a;
    double d2 = vec.vlensq(), sum = r1+r2, dif = r1-r2,
        p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
    if (sum*sum < d2 || dif*dif > d2) return false;
    point mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
    *out = {mid + per, mid - per};
    return true;
}
// point in polygon
bool insidePoly(point p, vector<point>poly) {
    point mx = {-1e9, -1e9};
    for (point pp : poly) {
        if (pp.x > mx.x) {
            mx = pp;
        }
    }
    mx.x++;
    int cnt = 0;
    for (int i = 0; i < poly.size(); i++) {
        point a = poly[i];
        point b = poly[(i + 1) % poly.size()];
        if (isect(p, mx, a, b)) {
            cnt++;
        }
    }
    return cnt == 1;
}
// get unit vector that is orthogonal to given vector
point getUnitOrthog(point vec) {
    if (abs(vec.x) < eps || abs(vec.y) < eps) {
        swap(vec.x, vec.y);
        return vec.toUnit();
    }
    point orthog(1, 0);
    orthog.y = -orthog.x * vec.x / vec.y;
    return orthog.toUnit();
}
// circle with center c, radius r and line through points a and b
vector<point> circleLine(point c, double r, point a, point b) {
    point ab = b - a, p = a + ab * dot(c - a, ab) / ab.vlensq();
    double s = cross(b - a, c - a), h2 = r*r - s*s / ab.vlensq();
    if (h2 < 0) return {};
    if (h2 == 0) return {p};
    point h = ab.toUnit() * sqrt(h2);
    return {p - h, p + h};
}

```

2.5 Rotational Sweep

```

#include <bits/stdc++.h>
#define pb push_back
#define all(a) a.begin(), a.end()
#define sz(a) (int)a.size()
#define x first
#define y second
#define debug(...) " [<#__VA_ARGS__<<": "<__VA_ARGS__<<"]"
using namespace std;
typedef long long ll;
typedef long double ld;
typedef pair<int, int> pii;
const int maxn = 1e6 + 100;
const int mod = 1e9 + 7;
mt19937 rng(chrono::high_resolution_clock::now().time_since_epoch().count());
struct point
{
    ll x, y;
    point() {}
    point(ll a, ll b)
    {
        x=a;
        y=b;
    }
    point operator+(const point &o)
    {
        return point(x+o.x, y+o.y);
    }
    point operator-(const point &o)
    {
        return point(x-o.x, y-o.y);
    }
};
ll cross(point a, point b)
{
    return a.x*b.y-a.y*b.x;
}
int n;
ll ans;
vector<point>v;
void sweep(int id) {
    vector<point>u;
    for(int i = 0; i < n; i++) {
        if(i != id) {
            u.push_back(v[i]);
        }
    }
    point p = v[id];
    sort(u.begin(), u.end(), [&](point a, point b) {
        if((a.y >= p.y) != (b.y >= p.y)) {
            return a.y >= p.y;
        }
        return cross(a - p, b - p) < 0;
    });
    int r = 0, cnt = 0;

```

```

for(int l = 0; l < u.size(); l++) {
    if (l > 0) {
        cnt--;
    }
    int rr = (r + 1) % u.size();
    while (cnt != n - 2 && cross(u[l] - p, u[rr] - p) <= 0) {
        cnt++;
        rr = (rr + 1) % u.size();
    }
    r = rr - 1;
    ll R = (r - l + u.size()) % u.size(), L = n - R - 2;
    ans += R * (R - 1) / 2 * L * (L - 1) / 2;
}
}
int main()
{
    ios_base::sync_with_stdio(false), cin.tie(0);
    cin >> n;
    v.resize(n);
    for(int i = 0; i < n; i++)
        cin >> v[i].x >> v[i].y;
    for(int i = 0; i < n; i++)
        sweep(i);
    cout << ans / 2 << "\n";
    return 0;
}

```

3 Graphs

3.1 2-Edge Connected Components

```

/* Description: Finding 2-edge connected components using Tarjan's algorithm and
   DSU */
#include <bits/stdc++.h>
#define pb push_back
using namespace std;
typedef long long ll;
typedef long double ld;
const int maxn = 3e5 + 100;
int n, m, val[maxn], low[maxn], timer = 1, par[maxn], sz[maxn];
vector<int> adj[maxn], g[maxn];
vector<pair<int, int>> bridges;
int root(int v) {
    return v == par[v] ? v : par[v] = root(par[v]);
}
void unite(int a, int b) {
    a = root(a), b = root(b);
    if (a == b) {
        return;
    }
    if (sz[a] < sz[b]) {
        swap(a, b);
    }

```

```

    }
    par[b] = a;
    sz[a] += sz[b];
}
// comments are for articulation point code
void tarjan(int v, int p = -1) {
    val[v] = low[v] = timer++;
    // int children = 0;
    for (auto to : adj[v]) {
        if (to == p) {
            continue;
        }
        if (!val[to]) {
            tarjan(to, v);
            low[v] = min(low[v], low[to]);
            // if (low[to] >= val[v] && p != -1)
            //     IS_ARTICULATION_POINT(v);
            // ++children;
            if (low[to] > val[v]) {
                bridges.push_back({v, to});
            } else {
                unite(v, to);
            }
        } else {
            low[v] = min(low[v], val[to]);
        }
    }
    // if (p == -1 && children > 1)
    //     IS_ARTICULATION_POINT(v);
}
void construct() {
    for (pair<int, int> p : bridges) {
        int a = root(p.first), b = root(p.second);
        g[a].push_back(b);
        g[b].push_back(a);
    }
}
int main() {
    ios_base::sync_with_stdio(false), cin.tie(0);
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        par[i] = i, sz[i] = 1;
    }
    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    tarjan(1);
    construct();
    // constructs a tree from the graph
    // edges are bridges in the original graph
    // vertices are combined into 1 if they are 2-edge connected

```

```

    return 0;
}

```

3.2 BCC

```

template<class T> bool ckmin(T& a, const T& b) { return a > b ? a = b, 1 : 0; }
template<class T> bool ckmax(T& a, const T& b) { return a < b ? a = b, 1 : 0; }
struct BCC {
    vector<vector<pair<int, int>>> adj; vector<pair<int, int>> ed;
    vector<vector<int>> edgeSets, vertSets; // edges for each bcc
    int N, ti = 0; vector<int> disc, stk;
    void init(int _N) { N = _N; disc.resize(N), adj.resize(N); }
    void ae(int x, int y) {
        adj[x].emplace_back(y, ed.size()), adj[y].emplace_back(x, ed.size()),
        ed.emplace_back(x, y);
    }
    int dfs(int x, int p = -1) { // return lowest disc
        int low = disc[x] = ++ti;
        for (auto &e : adj[x]) if (e.second != p) {
            if (!disc[e.first]) {
                stk.push_back(e.second); // disc[x] < LOW -> bridge
                int LOW = dfs(e.first, e.second); ckmin(low, LOW);
                if (disc[x] <= LOW) { // get edges in bcc
                    edgeSets.emplace_back(); vector<int>& tmp =
                    edgeSets.back(); // new bcc
                    for (int y = -1; y != e.second; )
                        tmp.push_back(y = stk.back()),
                        stk.pop_back();
                }
            } else if (disc[e.first] < disc[x]) // back-edge
                ckmin(low, disc[e.first]), stk.push_back(e.second);
        }
        return low;
    }
    void gen() {
        for (int i=0; i<N; i++) if (!disc[i]) dfs(i);
        vector<bool> in(N);
        for (auto &c : edgeSets) { // edges contained within each BCC
            vertSets.emplace_back(); // so you can easily create block
            cut tree
            auto ad = [&](int x) {
                if (!in[x]) in[x] = 1, vertSets.back().push_back(x);
            };
            for (auto &e : c) ad(ed[e].first), ad(ed[e].second);
            for (auto &e : c) in[ed[e].first] = in[ed[e].second] = 0;
        }
    }
};

```

3.3 Bipartite Matching

```

int n;
bool vis[maxn];
int match[maxn];
vector<int>adj[maxn];
bool dfs(int v) {
    if (vis[v]) {
        return false;
    }
    vis[v] = true;
    for (int to : adj[v]) {
        if (match[to] == -1 || dfs(match[to])) {
            match[v] = to, match[to] = v;
            return true;
        }
    }
    return false;
}
void matching() {
    memset(match, -1, sizeof match);
    while (true) {
        memset(vis, 0, sizeof vis);
        bool found = false;
        for (int i = 1; i <= n; i++) {
            if (!vis[i] && match[i] == -1) {
                found |= dfs(i);
            }
        }
        if (!found) {
            break;
        }
    }
}

```

3.4 Dijkstra

```

ll D[maxn];
vector<pii> adj[maxn];
void Dijkstra(int st) {
    fill(D, D + maxn, LLONG_MAX);
    priority_queue<pii, vector<pii>, greater<pii>> Q;
    Q.push({D[st] = 0, st});
    while (!Q.empty()) {
        pii v = Q.top();
        Q.pop();
        if (D[v.y] < v.x) {
            continue;
        }
        for (auto to : adj[v.y]) {
            if (v.x + to.y < D[to.x]) {
                Q.push({D[to.x] = v.x + to.y, to.x});
            }
        }
    }
}

```

```

    }
}

```

3.5 Dinic

```

struct edge {
    int to, flow, cap;
};
vector<edge> edges;
vector<int> adj[maxn];
void add_edge(int a, int b, int cap) {
    edge ab = {b, 0, cap};
    edge ba = {a, 0, 0};
    adj[a].push_back(edges.size());
    edges.push_back(ab);
    adj[b].push_back(edges.size());
    edges.push_back(ba);
}
int vis[maxn], ptr[maxn], level[maxn], s, t;
bool bfs() {
    queue<int> Q;
    Q.push(s);
    while (!Q.empty()) {
        int v = Q.front();
        Q.pop();
        for (int id : adj[v]) {
            edge e = edges[id];
            if (e.cap - e.flow < 1) {
                continue;
            }
            if (level[e.to] != -1) {
                continue;
            }
            level[e.to] = level[v] + 1;
            Q.push(e.to);
        }
    }
    return level[t] != -1;
}
int dfs(int v, int pushed) {
    if (pushed == 0) {
        return 0;
    }
    if (v == t) {
        return pushed;
    }
    for (int &j = ptr[v]; j < adj[v].size(); j++) {
        edge e = edges[adj[v][j]];
        if (level[v] + 1 != level[e.to] || e.cap - e.flow < 1) {
            continue;
        }
        int tr = dfs(e.to, min(pushed, e.cap - e.flow));
    }
}

```

```

        if (tr == 0) {
            continue;
        }
        edges[adj[v][j]].flow += tr;
        edges[adj[v][j] ^ 1].flow -= tr;
        return tr;
    }
    return 0;
}
int max_flow() {
    int flow = 0;
    while (true) {
        memset(level, -1, sizeof level);
        level[s] = 0;
        if (!bfs()) {
            break;
        }
        memset(ptr, 0, sizeof ptr);
        while (int pushed = dfs(s, mod)) {
            flow += pushed;
        }
    }
    return flow;
}

```

3.6 Edmonds-Karp

```

struct edge {
    int to, flow, cap, cost;
};
vector<edge> edges;
vector<int> adj[maxn];
void add_edge(int a, int b, int cap) {
    adj[a].push_back(edges.size());
    edges.push_back({b, 0, cap});
    adj[b].push_back(edges.size());
    edges.push_back({a, 0, 0});
}
int vis[maxn], pid[maxn], s, t;
int bfs() {
    queue<pair<int, int>> Q;
    vis[s] = 1;
    Q.push({s, mod});
    while (!Q.empty()) {
        int v, flow;
        tie(v, flow) = Q.front();
        Q.pop();
        if (v == t) {
            return flow;
        }
        for (int j : adj[v]) {
            edge e = edges[j];

```

```

            if (!vis[e.to] && e.cap - e.flow > 0) {
                vis[e.to] = 1;
                pid[e.to] = j;
                Q.push({e.to, min(e.cap - e.flow, flow)});
            }
        }
    }
    return 0;
}
int max_flow() {
    int maxflow = 0;
    while (true) {
        memset(vis, 0, sizeof vis);
        int cur_flow = bfs();
        if (cur_flow == 0) {
            break;
        }
        maxflow += cur_flow;
        int v = t;
        while (v != s) {
            int j = pid[v];
            edges[j].flow += cur_flow;
            edges[j ^ 1].flow -= cur_flow;
            v = edges[j ^ 1].to;
        }
    }
    return maxflow;
}

```

3.7 Eulerian Tour

```

int n, m, even;
set<int> adj[maxn];
vector<int> tour;
void dfs(int v) {
    for (auto it = adj[v].begin(); adj[v].size() > 0;) {
        int to = *it;
        it = adj[v].erase(it);
        adj[to].erase(v);
        dfs(to);
    }
    tour.push_back(v);
}
int main() {
    ios_base::sync_with_stdio(false), cin.tie(0);
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        adj[a].insert(b);
        adj[b].insert(a);
    }
}

```

```

for (int i = 1; i <= n; i++) {
    if (adj[i].size() % 2 == 0) {
        ++even;
    }
}
if (even != n && even != n - 2) {
    /// no tour exists
    return 0;
}
int start = 1;
if (even == n - 2) {
    for (int i = 1; i <= n; i++) {
        if (adj[i].size() % 2 == 1) {
            start = i;
            break;
        }
    }
}
dfs(start);
for (int v : tour) {
    cout << v << " ";
}
cout << "\n";
return 0;
}

```

3.8 Hungarian

```

template<class T> bool ckmin(T& a, const T& b) { return a > b ? a = b, 1 : 0; }
template<class T> bool ckmax(T& a, const T& b) { return a < b ? a = b, 1 : 0; }

// jobs 1..n, workers 1..m
// need row 0 (unused) of size m+1, otherwise runtime error
int hungarian(const vector<vector<int>>& a) {
    int n = sz(a) - 1, m = sz(a[0]) - 1;
    vector<int> u(n + 1), v(m + 1); // potentials
    vector<int> p(m + 1); // p[j] -> job picked by worker j
    for (int i = 1; i <= n; i++) { // find alternating path with job i
        p[0] = i; int j0 = 0; // add "dummy" worker 0
        vector<int> dist(m + 1, INT_MAX), pre(m + 1, -1); // prev vertex on shortest path
        vector<bool> done(m + 1, false);
        do { // dijkstra
            done[j0] = true; // fix dist[j0], update dists from j0
            int i0 = p[j0], j1;
            int delta = INT_MAX;
            for (int j = 1; j <= m; j++)
                if (!done[j]) {
                    auto cur = a[i0][j] - u[i0] - v[j];
                    if (ckmin(dist[j], cur))
                        pre[j] = j0;
                    if (ckmin(delta, dist[j]))

```

```

                        j1 = j;
                }
            }
            for (int j = 0; j <= m; j++) { // subtract constant from all edges going
                // from done -> not done vertices, lowers all
                // remaining dists by constant
                if (done[j])
                    u[p[j]] += delta, v[j] -= delta;
                else
                    dist[j] -= delta;
            }
            j0 = j1;
        } while (p[j0]); // potentials adjusted so that all edge weights are
            // non-negative
            // perfect matching has zero weight and
            // costs of augmenting paths do not change
        while (j0) { // update jobs picked by workers on alternating path
            int j1 = pre[j0];
            p[j0] = p[j1];
            j0 = j1;
        }
    }
    return -v[0]; // min cost
}

```

3.9 Min Cost Max Flow

```

struct edge {
    int to, flow, cap, cost;
    edge(int a = 0, int b = 0, int c = 0, int d = 0) {
        to = a, flow = b, cap = c, cost = d;
    }
};
vector<edge> edges;
vector<int> adj[maxn];
void add_edge(int a, int b, int cap, int cost) {
    edge ab(b, 0, cap, cost);
    edge ba(a, 0, 0, -cost);
    adj[a].push_back(edges.size());
    edges.push_back(ab);
    adj[b].push_back(edges.size());
    edges.push_back(ba);
}
int vis[maxn], pid[maxn], s, t;
int dist[maxn];
pii dijkstra() {
    priority_queue<pair<int, pii>, vector<pair<int, pii>>, greater<pair<int, pii>>> Q;
    vis[s] = 1;
    Q.push({0, {s, mod}});
    while (!Q.empty()) {
        int v, flow, w;
        pair<int, pii> p = Q.top();

```

```

    Q.pop();
    w = p.x;
    v = p.y.x;
    flow = p.y.y;
    if (dist[v] < w) {
        continue;
    }
    if (v == t) {
        return make_pair(w, flow);
    }
    for (int j : adj[v]) {
        edge e = edges[j];
        if (!vis[e.to] && e.cap - e.flow > 0) {
            if (w + e.cost < dist[e.to]) {
                dist[e.to] = w + e.cost;
                pid[e.to] = j;
                Q.push({dist[e.to], {e.to, min(e.cap - e.flow, flow)}});
            }
        }
    }
}
return make_pair(0, 0);
}
}

pii max_flow() {
    int maxflow = 0, mincost = 0;
    while (true) {
        fill(dist, dist + maxn, mod);
        pii cur_flow = dijkstra();
        if (cur_flow.y == 0) {
            break;
        }
        maxflow += cur_flow.y;
        mincost += cur_flow.x;
        int v = t;
        while (v != s) {
            int j = pid[v];
            edges[j].flow += cur_flow.y;
            edges[j ^ 1].flow -= cur_flow.y;
            v = edges[j ^ 1].to;
        }
    }
    return make_pair(mincost, maxflow);
}
}

```

3.10 Tarjan SCC

```

int val[maxn], low[maxn], scc[maxn], siz[maxn], tim, onStack[maxn], cnt;
stack<int> st;
void tarjan(int v) {
    low[v] = val[v] = tim++;
    onStack[v] = 1;
    st.push(v);

```

```

    for (int to : adj[v]) {
        if (val[to] == 0) {
            tarjan(to);
            low[v] = min(low[v], low[to]);
        } else if (onStack[to]) {
            low[v] = min(low[v], val[to]);
        }
    }
    if (low[v] == val[v]) {
        cnt++;
        while (true) {
            int u = st.top();
            st.pop();
            onStack[u] = 0;
            scc[u] = cnt;
            ++siz[cnt];
            if (u == v) {
                break;
            }
        }
    }
}
}

```

4 Math

4.1 Basis

```

/* Basis of vectors modulo 2 (for xor related problems) */
ll basis[62];
void add(ll a) {
    for (int bit = 61; bit >= 0; bit--) {
        if (a >> bit & 1) {
            if (basis[bit]) {
                a ^= basis[bit];
            } else {
                basis[bit] = a;
                return;
            }
        }
    }
}
}

```

4.2 CatalanNumbers

```

// Number of correct bracket sequence consisting of n opening and n closing
// brackets.
// Number of rooted full binary trees with n + 1 leaves (vertices are not
// numbered).

```

```
// A rooted binary tree is full if every vertex has either two children or no
// children
// Number of triangulations of a convex polygon with n + 2 sides
// Number of non-isomorphic full binary trees with n internal nodes (having at
// least 1 child)
// Number of ways to connect the 2n points on a circle to form n disjoint chords
// C_0 = C_1 = 1
// C_n = Sum(k=0 to n-1) (C_k * C_{n-1-k}), n >= 2
// Analytical formula C_n = (1 / (n + 1)) * choose(2n, n)
int catalan[MAX];
void init() {
    catalan[0] = catalan[1] = 1;
    for (int i=2; i<=n; i++) {
        catalan[i] = 0;
        for (int j=0; j < i; j++) {
            catalan[i] += (catalan[j] * catalan[i-j-1]) % MOD;
            if (catalan[i] >= MOD) {
                catalan[i] -= MOD;
            }
        }
    }
}
```

4.3 Extended Euclidean Algorithm

```
const ll mod = 1e9 + 7;
ll gcd(ll a, ll b, ll &x, ll &y) {
    if (b == 0) {
        x = 1, y = 0;
        return a;
    }
    ll x1, y1;
    ll ret = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return ret;
}
ll modinv(ll a) {
    ll x, y;
    gcd(a, mod, x, y);
    return (x % mod + mod) % mod;
}
```

4.4 FFT

```
#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
```

```
typedef vector<int> vi;
typedef complex<double> C;
typedef vector<double> vd;

void fft(vector<C>& a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1); // (~ 10% faster if double)
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n); rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        rep(i, k, 2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
    }
    vi rev(n);
    rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i, 0, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j, 0, k) {
            // C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled) ///
            include-line
            auto x = (double *)&rt[j+k], y = (double *)&a[i+j+k]; /// exclude-line
            C z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]); /// exclude-line
            a[i + j + k] = a[i + j] - z;
            a[i + j] += z;
        }
    }
    // conv(a, b) = c, where c[x] = Sum_over_i (a[i] * b[x - i])
    // rounding is safe if (Sum(a_i^2) + Sum(b_i^2)) * log_2(N) < 9 * 10^14 (in
    // practice 10^16)
    vd conv(const vd& a, const vd& b) {
        if (a.empty() || b.empty()) return {};
        vd res(sz(a) + sz(b) - 1);
        int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
        vector<C> in(n), out(n);
        copy(all(a), begin(in));
        rep(i, 0, sz(b)) in[i].imag(b[i]);
        fft(in);
        for (C& x : in) x *= x;
        rep(i, 0, n) out[i] = in[-i & (n - 1)] - conj(in[i]);
        fft(out);
        rep(i, 0, sz(res)) res[i] = imag(out[i]) / (4 * n);
        return res;
    }
}
```

4.5 Fraction

```
/*
 * Description: Fraction struct
 */
#include <bits/stdc++.h>
#define pb push_back
#define all(a) a.begin(), a.end()
```



```

#define sz(a) (int)a.size()
#define x first
#define y second
using namespace std;
typedef long long ll;
typedef long double ld;
typedef pair<int, int> pii;
const int maxn = 1e5 + 100;
struct fraction {
    ll sk, v;
    fraction() {
        sk = 0, v = 1;
    }
    fraction(ll _sk, ll _v) {
        sk = _sk, v = _v;
        ll dbd = __gcd(sk, v);
        sk /= dbd, v /= dbd;
        if (v < 0) sk *= -1, v *= -1;
    }
};
fraction abs(fraction f) { return fraction(abs(f.sk), f.v); }
bool operator<(const fraction &a, const fraction &b) { return a.sk * b.v < b.sk * a.v; }
bool operator>(const fraction &a, const fraction &b) { return a.sk * b.v > b.sk * a.v; }
bool operator==(const fraction &a, const fraction &b) { return a.sk == b.sk && a.v == b.v; }
bool operator!=(const fraction &a, const fraction &b) { return !(a == b); }
fraction operator+(const fraction &a, const fraction &b) { return fraction(a.sk * b.v + b.sk * a.v, a.v * b.v); }
fraction operator+=(fraction &a, const fraction &b) { return a = a + b; }
fraction operator-(const fraction &a, const fraction &b) { return fraction(a.sk * b.v - b.sk * a.v, a.v * b.v); }
fraction operator-=(fraction &a, const fraction &b) { return a = a - b; }
fraction operator*(const fraction &a, const fraction &b) { return fraction(a.sk * b.sk, a.v * b.v); }
fraction operator*=(fraction &a, const fraction &b) { return a = a * b; }
fraction operator*(int a, const fraction &b) { return fraction(a * b.sk, b.v); }
fraction operator*(const fraction &a, int b) { return fraction(a.sk * b, a.v); }
fraction operator/(const fraction &a, const fraction &b) { return fraction(a.sk * b.v, a.v * b.sk); }
fraction operator/=(fraction &a, const fraction &b) { return a = a / b; }
ostream &operator<<(ostream &strm, const fraction &a) {
    if (a.v == 1)
        strm << a.sk;
    else
        strm << a.sk << "/" << a.v;
    return strm;
}
int main() {
    ios_base::sync_with_stdio(false), cin.tie(0);
    int q, w, e, r;
    cin >> q >> w >> e >> r;
    fraction a(q, w);
    fraction b(e, r);

```

```

    cout << "< " << (a < b) << "\n";
    cout << "> " << (a > b) << "\n";
    cout << "==" << (a == b) << "\n";
    cout << "!=" << (a != b) << "\n";
    cout << "+" << (a + b) << "\n";
    cout << "-" << (a - b) << "\n";
    cout << "*" << (a * b) << "\n";
    cout << "/" << (a / b) << "\n";
    return 0;
}

```

4.6 Gauss

```

const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be infinity or a big number

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 = 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;
    }
}

```

```

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

vector<int> GaussModulo(vector<vector<int>> mat) {
    for (int i = 0; i < mat.size(); i++) {
        int val = mat[i][i];
        int mul = modinv(val);
        for (int k = 0; k < mat[i].size(); k++)
            mat[i][k] = 11l * mat[i][k] * mul % mod;
        for (int j = 0; j < mat.size(); j++)
            if (i != j) {
                int times = mat[j][i];
                for (int k = 0; k < mat[j].size(); k++)
                    mat[j][k] = (mat[j][k] - 11l * times * mat[i][k] % mod + mod) %
                        mod;
            }
    }
    vector<int> ret;
    for (int i = 0; i < mat.size(); i++) {
        ret.push_back(mat[i].back());
    }
    return ret;
}

```

4.7 Integration

```

const int N = 1000 * 1000; // number of steps (already multiplied by 2)
double simpson_integration(double a, double b) {
    double h = (b - a) / N;
    double s = f(a) + f(b); // a = x_0 and b = x_2n
    for (int i = 1; i <= N - 1; ++i) {
        double x = a + h * i;
        s += f(x) * ((i & 1) ? 4 : 2);
    }
    s *= h / 3;
    return s;
}

```

4.8 Matrix Struct

```

int add(int a, int b) {
    int c = a + b;
    if (c > mod) {
        c -= mod;
    }
    return c;
}

```

```

}
int mul(int a, int b) {
    ll c = (11l)a * b;
    c %= mod;
    return (int)c;
}

struct matrix {
    int N, M;
    vector<vector<int>> mat;
    matrix(int n, int m) {
        N = n, M = m;
        mat = vector<vector<int>>(N, vector<int>(M, 0));
    }
    void ident() {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                mat[i][j] = (i == j);
            }
        }
    }
    matrix(vector<vector<int>> arr) {
        N = arr.size(), M = arr[0].size();
        mat = vector<vector<int>>(N, vector<int>(M, 0));
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                mat[i][j] = arr[i][j];
            }
        }
    }
    matrix operator*(const matrix &o) {
        matrix ret(N, o.M);
        for (int i = 0; i < ret.N; i++) {
            for (int j = 0; j < ret.M; j++) {
                for (int k = 0; k < M; k++) {
                    ret.mat[i][j] = add(ret.mat[i][j], mul(mat[i][k], o.mat[k][j]));
                }
            }
        }
        return ret;
    }
};

matrix pwr(matrix a, ll pw) {
    matrix ret(a);
    ret.ident();
    while (pw > 0) {
        if (pw & 1) {
            ret = ret * a;
        }
        a = a * a;
        pw >>= 1;
    }
    return ret;
}

```

4.9 Modulo operations

```
typedef long long ll;
const ll mod = 1e9 + 7;
ll pwr(ll num, ll pw) {
    ll ret = 1;
    while (pw > 0) {
        if (pw & 1) {
            ret = (ret * num) % mod;
        }
        num = (num * num) % mod;
        pw >>= 1;
    }
    return ret;
}
ll modinv(ll a) { return pwr(a, mod - 2); }
ll add(ll a, ll b) { return (a + b) % mod; }
ll sub(ll a, ll b) { return (a - b + mod) % mod; }
ll mul(ll a, ll b) { return (a * b) % mod; }
```

4.10 PermutationExp

```
vector<int> applyPermutation(vector<int> sequence, vector<int> permutation) {
    vector<int> newSequence(sequence.size());
    for (int i = 0; i < sequence.size(); i++) {
        newSequence[permutation[i]] = sequence[i];
    }
    return newSequence;
}

vector<int> permute(vector<int> sequence, vector<int> permutation, long long b) {
    while (b > 0) {
        if (b & 1) {
            sequence = applyPermutation(sequence, permutation);
        }
        permutation = applyPermutation(permutation, permutation);
        b >>= 1;
    }
    return sequence;
}
```

5 Misc

5.1 Custom Vector

```
/* Description: Implementation of vector with negative indexing */
#include <bits/stdc++.h>
using namespace std;
```

```
template<class T>
struct MyVector {
    int offset;
    vector<T> vec_;

    MyVector(int n, int off, T &&values = T()) {
        vec_.resize(n, values);
        offset = off;
    }

    T& operator[](int i) {
        i += offset;
        assert(0 <= i && i < (int)vec_.size());
        return vec_[i];
    }
};

int main()
{
    ios_base::sync_with_stdio(false), cin.tie(0);
    // Example usage
    MyVector<MyVector<int>> dp(10 + 1, 5, MyVector<int>(8 + 1, 4));

    for (int i = -5; i <= 5; i++) {
        for (int j = -4; j <= 4; j++) {
            dp[i][j] = i * j;
        }
    }

    for (int i = -5; i <= 5; i++) {
        for (int j = -4; j <= 4; j++) {
            cout << i << " " << j << " " << dp[i][j] << "\n";
        }
    }
    return 0;
}
```

6 Notes

6.1 Notes

```
/*
---
Pick's theorem:
If a polygon has all integer coordinates then the area A is
A = i + b / 2 - 1
where i is the number of integer points interior to the polygon
b is the number of integer points on the boundary
---
Shoelace formula:
1/2 * (x1*y2 - x2*y1 + x2*y3 - x3*y2 + ... + xn*y1 - x1*yn)
```

```

---
Euler's formula:
For a connected planar graph with v vertices, e edges and f faces (including
    outer)
v - e + f = 2
*/
// Mo's sorting order
bool cmp(Query A, Query B)
{
    if (A.l / S != B.l / S) return A.l / S < B.l / S;
    return A.r > B.r
}

```

7 Strings

7.1 Bitset Trie

```

template <int N>
struct trie {
    int t[N + 1][2], nxt = 1;

    trie() {
        memset(t, 0, sizeof t);
    }

    void ins(string s) {
        int cur = 0;
        for (int i = 0; i < s.size(); ++i) {
            int bit = s[i] - '0';
            if (!t[cur][bit]) {
                t[cur][bit] = nxt++;
            }
            cur = t[cur][bit];
        }
    }

    bool find(string s) {
        int cur = 0;
        for (int i = 0; i < s.size(); ++i) {
            int bit = s[i] - '0';
            if (!t[cur][bit]) {
                return false;
            }
            cur = t[cur][bit];
        }
        return true;
    }
};

```

7.2 Hashing

```

/*
Works, but might be a little slow
https://judge.yosupo.jp/submission/80957
*/
bool is_prime(int a) {
    if (a == 1) {
        return false;
    }
    if (a % 2 == 0) {
        return a == 2;
    }
    for (int i = 3; i * i <= a; i += 2) {
        if (a % i == 0) {
            return false;
        }
    }
    return true;
}

int pwr(int a, int pw, int MOD) {
    int ret = 1;
    while (pw > 0) {
        if (pw & 1) {
            ret = 1ll * ret * a % MOD;
        }
        a = 1ll * a * a % MOD;
        pw >>= 1;
    }
    return ret;
}

int modinv(int a, int MOD) {
    return pwr(a, MOD - 2, MOD);
}

struct hashing {
    vector<vector<int>> >ha;
    vector<vector<int>> >rha;
    static vector<vector<int>> >pw;
    static vector<vector<int>> >inv_pw;
    static vector<int> mods;
    static vector<int> BASE_INV;
    static int BASE;
    static int M;
    int N;

    static vector<int> zero_vector(int len) {
        vector<int> ret;
        for (int i = 0; i < len; i++) {
            ret.push_back(0);
        }
        return ret;
    }
}

```

```

struct hash {
    vector<int>h;
    int len;

    hash() {
        h = zero_vector(M);
        len = 0;
    }

    hash(vector<int>h, int len) {
        this->h = h;
        this->len = len;
    }

    bool operator==(const hash& b) const {
        return h == b.h && len == b.len;
    }

    hash operator-(const hash& b) {
        hash c;
        c.h = h;

        for (int j = 0; j < M; j++) {
            c.h[j] -= b.h[j];
            if (c.h[j] < 0) {
                c.h[j] += mods[j];
            }
            c.h[j] = 111 * c.h[j] * inv_pw[b.len][j] % mods[j];
        }

        c.len = len - b.len;
        return c;
    }

    hash operator--(const hash& b) {
        hash c = *this;
        c = c - b;
        this->h = c.h;
        this->len = c.len;
        return *this;
    }

    hash operator+(const hash& b) {
        hash c;
        c.h = h;

        for (int j = 0; j < M; j++) {
            c.h[j] += 111 * pw[len][j] * b.h[j] % mods[j];
            if (c.h[j] >= mods[j]) {
                c.h[j] -= mods[j];
            }
        }

        c.len = len + b.len;
    }

```

```

        return c;
    }

    hash operator+=(const hash& b) {
        hash c = *this;
        c = c + b;
        this->h = c.h;
        this->len = c.len;
        return *this;
    }

    bool operator<(const hash& b) const {
        for (int i = 0; i < M; i++) {
            if (h[i] != b.h[i]) {
                return h[i] < b.h[i];
            }
        }
        return len < b.len;
    }
};

hashing(string s) {
    if (M == 0 || BASE == 0) {
        cerr << "Initialize mods and base before creating a hashing instance!";
        exit(1);
    }
    init_string(s);
}

static void init_base() {
    mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
    uniform_int_distribution<>distrib(3, (int)1e9);
    while (true) {
        BASE = distrib(rng);
        if (is_prime(BASE)) {
            break;
        }
    }
}

static void init_mods(vector<int>new_mods) {
    mods = new_mods;
    M = (int)mods.size();
    BASE_INV = vector<int>(M);
    for (int j = 0; j < M; j++) {
        BASE_INV[j] = modinv(BASE, mods[j]);
    }
}

static void init_pows(int MAX_POW) {
    pw = vector<vector<int>>(MAX_POW + 1, vector<int>(M));
    inv_pw = vector<vector<int>>(MAX_POW + 1, vector<int>(M));

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

```

```

        if (i == 0) {
            pw[i][j] = 1;
            inv_pw[i][j] = 1;
        } else {
            pw[i][j] = 111 * pw[i - 1][j] * BASE % mods[j];
            inv_pw[i][j] = 111 * inv_pw[i - 1][j] * BASE_INV[j] % mods[j];
        }
    }
}

void init_string(string s) {
    N = (int)s.size();

    ha = vector<vector<int>>(N, vector<int>(M));
    rha = vector<vector<int>>(N, vector<int>(M));

    init_pows(N * 2);

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            ha[i][j] = 111 * pw[i][j] * (s[i] - 'a' + 1) % mods[j];
            rha[i][j] = 111 * pw[i][j] * (s[N - i - 1] - 'a' + 1) % mods[j];

            if (i > 0) {
                ha[i][j] += ha[i - 1][j];
                if (ha[i][j] >= mods[j]) {
                    ha[i][j] -= mods[j];
                }
                rha[i][j] += rha[i - 1][j];
                if (rha[i][j] >= mods[j]) {
                    rha[i][j] -= mods[j];
                }
            }
        }
    }

    hash get_hash(int i, int j, bool reverse) {
        if (i > j) {
            return hash();
        }

        hash pref_i, pref_j;
        if (reverse) {
            i = N - i - 1;
            j = N - j - 1;
            swap(i, j);
            pref_i = (i == 0 ? hash() : hash(rha[i - 1], i));
            pref_j = hash(rha[j], j + 1);
        } else {
            pref_i = (i == 0 ? hash() : hash(ha[i - 1], i));
            pref_j = hash(ha[j], j + 1);
        }
    }
}

```

```

        return pref_j - pref_i;
    }
};

vector<vector<int>> hashing::pw;
vector<vector<int>> hashing::inv_pw;
vector<int> hashing::mods;
vector<int> hashing::BASE_INV;
int hashing::BASE;
int hashing::M;

```

7.3 Hashing2

```

#include <iostream>
#include <vector>
using namespace std;

typedef long long ll;

struct RollingHash {
    static const ll mod0 = 1000000123, mod1 = 1000000007;
    static const ll base0 = 137, base1 = 163;

    vector<ll> po0, po1;
    vector<ll> hsh0, hsh1;

    void init(string str) {
        if (!po0.size()) po0 = po1 = {1};
        hsh0 = {base0}, hsh1 = {base1};

        for (int i = 0; i < (int)str.size(); i++) {
            if (str[i] >= 'A' && str[i] <= 'Z') str[i] += 'a' - 'A';
            hsh0.push_back((hsh0.back() * base0 % mod0 + str[i]) % mod0);
            hsh1.push_back((hsh1.back() * base1 % mod1 + str[i]) % mod1);
        }

        pair<ll, ll> cut(int l, int r) {
            while (po0.size() <= r + 1) {
                po0.push_back(po0.back() * base0 % mod0);
                po1.push_back(po1.back() * base1 % mod1);
            }

            return make_pair((hsh0[r + 1] + mod0 - (hsh0[l] * po0[r - l + 1] % mod0)) % mod0,
                            (hsh1[r + 1] + mod1 - (hsh1[l] * po1[r - l + 1] % mod1)) % mod1);
        }
    } rh;
}

```

7.4 KMP

```

/* Description: Linear time string matching */
#include <bits/stdc++.h>
#define pb push_back
#define all(a) a.begin(), a.end()
#define sz(a) (int)a.size()
#define x first
#define y second
using namespace std;
typedef long long ll;
typedef long double ld;
typedef pair<int, int> pii;
const int maxn = 1e5 + 100;
vector<int> build_lps(string s) {
    vector<int> lps(sz(s), 0);
    for (int i = 1; i < sz(s); ++i) {
        int id = lps[i - 1];
        while (id && s[i] != s[id]) {
            id = lps[id - 1];
        }
        lps[i] = id + (s[i] == s[id]);
    }
    return lps;
}
vector<int> KMP(string s, string pat) {
    vector<int> lps = build_lps(pat + '\0' + s), res;
    for (int i = 0; i < sz(lps); ++i) {
        if (lps[i] == sz(pat)) {
            res.pb(i - 2 * sz(pat));
        }
    }
    return res;
}
int main() {
    ios_base::sync_with_stdio(false), cin.tie(0);
    string s, pat;
    cin >> s >> pat;
    vector<int> res = KMP(s, pat);
    for (auto aa : res) {
        cout << aa << " ";
    }
    cout << "\n";
    return 0;
}

```

7.5 SuffixArray

```

vector<int> sort_cyclic_shifts(string const& s) {
    int n = s.size();
    const int alphabet = 256;
    vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
    for (int i = 0; i < n; i++)
        cnt[s[i]]++;

```

```

    for (int i = 1; i < alphabet; i++)
        cnt[i] += cnt[i - 1];
    for (int i = 0; i < n; i++)
        p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i - 1]])
            classes++;
        c[p[i]] = classes - 1;
    }
    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0)
                pn[i] += n;
        }
        fill(cnt.begin(), cnt.begin() + classes, 0);
        for (int i = 0; i < n; i++)
            cnt[c[pn[i]]]++;
        for (int i = 1; i < classes; i++)
            cnt[i] += cnt[i - 1];
        for (int i = n - 1; i >= 0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        classes = 1;
        for (int i = 1; i < n; i++) {
            pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n]};
            pair<int, int> prev = {c[p[i - 1]], c[(p[i - 1] + (1 << h)) % n]};
            if (cur != prev)
                ++classes;
            cn[p[i]] = classes - 1;
        }
        c.swap(cn);
    }
    return p;
}
vector<int> suffix_array_construction(string s) {
    s += "$";
    vector<int> sorted_shifts = sort_cyclic_shifts(s);
    sorted_shifts.erase(sorted_shifts.begin());
    return sorted_shifts;
}

```

8 Trees

8.1 Centroid Decomposition

```

/* Description: Centroid Decomposition on trees */
#include <bits/stdc++.h>
#define push_back push_back

```

```

#define all(a) a.begin(), a.end()
#define sz(a) (int)a.size()
#define x first
#define y second
using namespace std;
typedef long long ll;
typedef long double ld;
typedef pair<int, int> pii;
const int maxn = 1e5 + 100;
vector<int> adj[maxn];
int n, sub[maxn], par[maxn];
bool vis[maxn];
char ans[maxn];
void dfs(int v) {
    sub[v] = 1;
    for (auto to : adj[v]) {
        if (!vis[to] && to != par[v]) {
            par[to] = v;
            dfs(to);
            sub[v] += sub[to];
        }
    }
}
int get_centroid(int v) {
    par[v] = 0;
    dfs(v);
    int sz = sub[v];
    while (true) {
        pii mx = {0, 0};
        for (auto to : adj[v]) {
            if (!vis[to] && par[v] != to) {
                mx = max(mx, {sub[to], to});
            }
        }
        if (mx.first * 2 > sz) {
            v = mx.second;
        } else {
            return v;
        }
    }
}
void solve(int v, int level = 0) {
    v = get_centroid(v);
    vis[v] = true;
    ans[v] = (char)(level + 'A');
    for (auto to : adj[v]) {
        if (!vis[to]) {
            solve(to, level + 1);
        }
    }
}
int main() {
    ios_base::sync_with_stdio(false), cin.tie(0);
    cin >> n;
    for (int i = 0; i < n - 1; i++) {

```

```

        int a, b;
        cin >> a >> b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    solve(1);
    for (int i = 1; i <= n; i++) {
        cout << ans[i] << " ";
    }
    cout << "\n";
    return 0;
}

```

8.2 HeavyLight

```

void dfs_sz(int v = 0) {
    sz[v] = 1;
    for (auto &u: g[v]) {
        dfs_sz(u);
        sz[v] += sz[u];
        if (sz[u] > sz[g[v][0]]) {
            swap(u, g[v][0]);
        }
    }
}
void dfs_hld(int v = 0) {
    in[v] = t++;
    for (auto u: g[v]) {
        nxt[u] = (u == g[v][0] ? nxt[v] : u);
        dfs_hld(u);
    }
    out[v] = t;
}
// subtree v corresponds to segment [in_v; out_v)
// the path from v to the last vertex in ascending heavy path
// from v (which is nxt_v) will be [in_{nxt_v}; in_v] subsegment
// which gives you the opportunity to process queries on paths
// and subtrees simultaneously in the same segment tree

```

8.3 Lowest Common Ancestor

```

/* Description: Offline Lowest Common Ancestor with NlogN preprocessing and logN
   query using binary lifting */
const int maxn = 1e5 + 100;
const int maxlog = 20;
int dep[maxn], anc[maxn][maxlog];
vector<int> adj[maxn];
void dfs(int v, int p = 0) {
    dep[v] = dep[p] + 1;
    anc[v][0] = p;

```



```

    for (int i = 1; i < 18; i++) {
        anc[v][i] = anc[anc[v][i - 1]][i - 1];
    }
    for (int to : adj[v]) {
        if (to != p) {
            dfs(to, v);
        }
    }
}
int lca(int a, int b) {
    if (dep[a] < dep[b]) {
        swap(a, b);
    }
    for (int i = 17; i >= 0; i--) {
        if (dep[anc[a][i]] >= dep[b]) {
            a = anc[a][i];
        }
    }
}

```

```

    }
}
if (a == b) {
    return a;
}
for (int i = 17; i >= 0; i--) {
    if (anc[a][i] != anc[b][i]) {
        a = anc[a][i], b = anc[b][i];
    }
}
return anc[a][0];
}
int getDist(int v, int u) {
    return dep[v] + dep[u] - 2 * dep[lca(v, u)];
}

```
