

Muffix Sassif – TRD

Andrianov, Lepeshov, Shulyatev



Содержание

1	Геометрия	2
1.1	2D	2
1.2	3D	2
1.3	Выпуклая оболочка	2
1.4	Касательные из точки	3
1.5	Касательные параллельные прямой	3
2	Графы	3
2.1	Венгерский алгоритм	3
2.2	Дейкстра за квадрат	3
2.3	Диниц	3
2.4	КСС	4

2.5	Минкост (Джонсон)	4
2.6	Мосты	5
2.7	Паросочетания	5
2.8	Точки сочленения	5
2.9	Эдмондс-Карп	5
2.10	Эйлеров цикл	6
3	ДП	6
3.1	СНТ	6
3.2	Li Chao	6
3.3	SOS-dp	6
3.4	НВП	6
3.5	НОВП	7
4	Деревья	7
4.1	Centroid	7
4.2	HLD	7
4.3	Link-cut	8
5	Другое	9
5.1	attribute_packed	9
5.2	ordered_set	9
5.3	pragma	9
5.4	Аллокатор Копелиовича	9
6	Математика	9
6.1	2-SAT	9
6.2	FFT mod	9

6.3	FFT	10
6.4	Гаусс	11
6.5	Диофантовы уравнения	11
6.6	КТО	11
6.7	Код Грея	11
6.8	Линейное решето	11
6.9	Ро-Поллард	11
6.10	Символ Якоби, Лежандра	11
7	Строки	12
7.1	Z-функция	12
7.2	Ахо-Корасик	12
7.3	Префикс-функция	12
7.4	Суффиксный автомат	12
7.5	Суффиксный массив	12
8	Структуры данных	13
8.1	Disjoint Sparse Table	13
8.2	Segment Tree Beats	13
8.3	ДД по невявному	14
8.4	ДД	14
8.5	Персистентное ДД	15
8.6	Персистентное ДО	15
8.7	Спарсы	15
8.8	Фенвик (pref += x)	15
8.9	Фенвик	15

1 Геометрия

1.1 2D

```
class Pt:
    def dot(self, other):
        return self.x * other.x + self.y * other.y

    def cross(self, other):
        return self.x * other.y - self.y * other.x

    @staticmethod
    def get_straight(self, other):
        a = self.y - other.y
        b = other.x - self.x
        c = self.cross(other)
        return a, b, c

class Straight:

    def __eq__(self, other):
        if self.b != 0 or other.b != 0:
            return self.a * other.b == other.a * self.b
        and self.c * other.b == other.c * self.b
        val1 = math.sqrt(self.a ** 2 + self.b ** 2)
        val2 = math.sqrt(other.a ** 2 + other.b ** 2)
        a1, c1 = self.a / val1, self.c / val1
        a2, c2 = other.a / val2, other.c / val2
        if (a1 < 0) != (a2 < 0):
            a1, a2, c1, c2 = a1, -a2, c1, -c2
        return a1 == a2 and c1 == c2

    def perpendicular(self, point: Pt):
        return Straight(-self.b, self.a, self.b * point
            .x - self.a * point.y)

    def get_value(self, point):
        return self.a * point.x + self.b * point.y +
            self.c

    def intersection(self, other):
        d = Pt(self.a, self.b).cross(Pt(other.a, other.
            b))
        dx = Pt(self.c, self.b).cross(Pt(other.c, other
            .b))
        dy = Pt(self.a, self.c).cross(Pt(other.a, other
            .c))
        return Pt(-dx / d, -dy / d)
```

```
def dist_from_point(self, point):
    val = math.sqrt(self.a ** 2 + self.b ** 2)
    return abs(Straight(self.a / val, self.b / val,
        self.c / val).get_value(point))

def parallel(self, dist):
    val = math.sqrt(self.a ** 2 + self.b ** 2)
    return Straight(self.a, self.b, self.c - dist *
        val)

def is_parallel(self, other):
    return self.a * other.b == self.b * other.a

def is_perpendicular(self, other):
    per = Straight(-self.b, self.a, 0)
    return per.a * other.b == per.b * other.a

class Triangle:
    def intersection_medians(self):
        return (self.A + self.B + self.C) / 3

    def intersection_altitudes(self):
        st1 = Straight(self.A, self.B).perpendicular(
            self.C)
        st2 = Straight(self.A, self.C).perpendicular(
            self.B)
        return st1.intersection(st2)

    def intersection_middle_pers(self):
        st1 = Straight(self.A, self.B).perpendicular((
            self.A + self.B) / 2)
        st2 = Straight(self.A, self.C).perpendicular((
            self.A + self.C) / 2)
        return st1.intersection(st2)

class Circle:
    def intersect_straight(self, st):
        pt = st.get_point()
        A = st.a ** 2 + st.b ** 2
        B = 2 * (-st.b * (pt.x - self.center.x) + st.a
            * (pt.y - self.center.y))
        C = (pt.x - self.center.x) ** 2 + (pt.y - self.
            center.y) ** 2 - self.radius ** 2
        D = B ** 2 - 4 * A * C
        if D < 0:
            return []
        D = math.sqrt(D)
```

```
vector = Pt(-st.b, st.a)
if D == 0:
    t = -B / (2 * A)
    return [pt + t * vector]
t1 = (-B - D) / (2 * A)
t2 = (-B + D) / (2 * A)
return [pt + t1 * vector, pt + t2 * vector]

def intersect_circle(self, other):
    x1, x2 = self.center.x, other.center.x
    y1, y2 = self.center.y, other.center.y
    a = -2 * (x1 - x2)
    b = -2 * (y1 - y2)
    c = (x1 ** 2 - x2 ** 2) + (y1 ** 2 - y2 ** 2) -
        (self.radius ** 2 - other.radius ** 2)
    return self.intersect_straight(Straight(a, b, c
        ))

def is_own(self, pt):
    return (pt.x - self.center.x) ** 2 + (pt.y -
        self.center.y) ** 2 == self.radius ** 2

def tangent_pts(self, pt):
    if self.is_own(pt):
        return [pt]
    cir = Circle(pt, math.sqrt(abs(pt - self.center
        ) ** 2 - self.radius ** 2))
    return self.intersect_circle(cir)

def dist_by_circle(self, pt1, pt2):
    pt1 -= self.center
    pt2 -= self.center
    ang = (pt1.polar_angle() - pt2.polar_angle()) %
        (2 * math.pi)
    ang = min(ang, 2 * math.pi - ang)
    return self.radius * ang
```

1.2 3D

```
// TODO
```

1.3 Выпуклая оболочка

```
Pt start(0, 0);

bool comp(Pt a, Pt b) {
    ll ang = (a - start).cross(b - start);
    if (ang < 0) {
```

```

        return false;
    } else if (ang > 0) {
        return true;
    }
    return abs(start - a) < abs(start - b);
}

vector<Pt> convex_hull(vector<Pt> points) {
    int n = points.size();
    start = points[0];
    for (auto x : points) {
        start = min(start, x);
    }
    sort(points.begin(), points.end(), comp);
    vector<Pt> s = {points[0], points[1]};
    for (int i = 2; i < n; i++) {
        int k = s.size();
        while (k > 1 && (s[k - 1] - s[k - 2]).cross(
            points[i] - s[k - 1]) <= 0) {
            s.pop_back();
            k = s.size();
        }
        s.push_back(points[i]);
    }
    return s;
}

```

1.4 Касательные из точки

```
// TODO
```

1.5 Касательные параллельные прямой

```
// TODO
```

2 Графы

2.1 Венгерский алгоритм

```

vector<int> venger(vector<vector<int>> arr) {
    int n = (int) arr.size() - 1;
    vector<int> u(n + 1), v(n + 1), p(n + 1), way(n + 1);
    for (int i = 1; i <= n; i++) {
        p[0] = i;
        int ind = 0;
        vector<int> minv(n + 1, INF), used(n + 1);

```

```

        do {
            used[ind] = 1;
            int ind2 = p[ind], dlt = INF, ind3 = 0;
            for (int j = 1; j <= n; j++)
                if (!used[j]) {
                    int cur = arr[ind2][j] - u[ind2] -
                        v[j];

                    if (cur < minv[j]) {
                        minv[j] = cur;
                        way[j] = ind;
                    }
                    if (minv[j] < dlt) {
                        dlt = minv[j], ind3 = j;
                    }
                }
            for (int j = 0; j <= n; j++)
                if (used[j]) {
                    u[p[j]] += dlt;
                    v[j] -= dlt;
                } else {
                    minv[j] -= dlt;
                }
            ind = ind3;
        } while (p[ind] != 0);
        do {
            int ind3 = way[ind];
            p[ind] = p[ind3];
            ind = ind3;
        } while (ind);
    }
    vector<int> ans(n + 1);
    for (int j = 1; j <= n; j++) {
        ans[p[j]] = j;
    }
    return ans;
}

```

2.2 Дейкстра за квадрат

```
// TODO
```

2.3 Диниц

```

vector<edge> g[MAXN];
pair<int, int> pred[MAXN];
int d[MAXN];
int inds[MAXN];

```

```

bool dfs(int v, int final, int W) {
    if (v == final) {
        return true;
    }
    for (int i = inds[v]; i < (int) g[v].size(); i++) {
        auto e = g[v][i];
        if (e.f + W <= e.c && d[v] + 1 == d[e.v]) {
            pred[e.v] = {v, i};
            bool flag = dfs(e.v, final, W);
            if (flag) {
                return true;
            }
            inds[v]++;
        } else {
            inds[v]++;
        }
    }
    return false;
}

bool bfs(int start, int final, int W) {
    fill(d, d + MAXN, INF);
    d[start] = 0;
    deque<int> q = {start};
    while (!q.empty()) {
        int v = q.front();
        q.pop_front();
        for (auto e : g[v]) {
            if (e.f + W <= e.c && d[e.v] > d[v] + 1) {
                d[e.v] = d[v] + 1;
                q.push_back(e.v);
            }
        }
    }
    if (d[final] == INF) {
        return false;
    }
    fill(inds, inds + MAXN, 0);
    while (dfs(start, final, W)) {
        int v = final;
        int x = INF;
        while (v != start) {
            int ind = pred[v].second;
            v = pred[v].first;
            x = min(x, g[v][ind].c - g[v][ind].f);
        }
        v = final;
        while (v != start) {
            int ind = pred[v].second;
            v = pred[v].first;

```

```

        g[v][ind].f += x;
        g[g[v][ind].v][g[v][ind].ind].f -= x;
    }
}
return true;
}

void Dinic(int start, int final) {
    int W = (1LL << 30);
    do {
        while (bfs(start, final, W));
        W /= 2;
    } while (W >= 1);
}

signed main() {
    vector<pair<int, int>> edges;
    for (int i = 0; i < m; i++) {
        int u, v, c;
        cin >> u >> v >> c;
        edges.emplace_back(u, v);
        g[u].push_back({v, 0, c, (int) g[v].size()});
        g[v].push_back({u, 0, c, (int) g[u].size() -
1});
    }
    Dinic(1, n);
    int res = 0;
    for (auto e : g[1]) {
        res += e.f;
    }
    vector<int> ans;
    for (int i = 0; i < m; i++) {
        int u = edges[i].first, v = edges[i].second;
        if ((d[u] != INF && d[v] == INF) || (d[u] ==
INF && d[v] != INF)) {
            ans.push_back(i + 1);
        }
    }
}

```

2.4 KCC

```

void dfs1(int v) {
    vis[v] = 1;
    for (auto u : g[v]) {
        if (!vis[u]) {
            dfs1(u);
        }
    }
}

```

```

    topsort.push_back(v);
}

void dfs2(int v) {
    vis[v] = 1;
    for (auto u : rg[v]) {
        if (!vis[u]) {
            dfs2(u);
        }
    }
    comp.push_back(v);
}

signed main() {
    for (int i = 1; i <= n; i++)
        if (!vis[i])
            dfs1(i);
    reverse(topsort.begin(), topsort.end());
    for (int j = 1; j <= n; j++) {
        int vert = topsort[j - 1];
        if (!vis[vert])
            dfs2(vert);
    }
}

```

2.5 Минкост (Джонсон)

```

using cost_t = ll;
using flow_t = int;

const int MAXN = 10000;
const int MAXM = 25000 * 2;
const cost_t INFw = 1e12;
const flow_t INFf = 10;

struct Edge {
    int v, u;
    flow_t f, c;
    cost_t w;
};

Edge edg[MAXN];
int esz = 0;
vector<int> graph[MAXN];
ll dist[MAXN];
ll pot[MAXN];
int S, T;
int NUMV;

```

```

int pre[MAXN];
bitset<MAXN> inQ;

flow_t get_flow() {
    int v = T;
    if (pre[v] == -1)
        return 0;
    flow_t f = INFf;
    do {
        int ei = pre[v];
        Edge &e = edg[ei];
        f = min(f, e.c - e.f);
        if (f == 0)
            return 0;
        v = e.v;
    } while (v != S);
    v = T;
    do {
        int ei = pre[v];
        edg[ei].f += f;
        edg[ei ^ 1].f -= f;
        v = edg[ei].v;
    } while (v != S);
    return f;
}

void spfa() {
    fill(dist, dist + NUMV, INFw);
    dist[S] = 0;
    deque<int> Q = {S};
    inQ[S] = true;
    while (!Q.empty()) {
        int v = Q.front();
        Q.pop_front();
        inQ[v] = false;
        cost_t d = dist[v];
        for (int ei : graph[v]) {
            Edge &e = edg[ei];
            if (e.f == e.c)
                continue;
            cost_t w = e.w + pot[v] - pot[e.u];
            if (dist[e.u] <= d + w)
                continue;
            pre[e.u] = ei;
            dist[e.u] = d + w;
            if (!inQ[e.u]) {
                inQ[e.u] = true;
                Q.push_back(e.u);
            }
        }
    }
}

```

```

    }
    for (int i = 0; i < NUMV; ++i)
        pot[i] += dist[i];
}

cost_t mincost() {
    spfa(); // pot[i] = 0 // or ford_bellman
    flow_t f = 0;
    while (true) {
        flow_t ff = get_flow();
        if (ff == 0)
            break;
        f += ff;
        spfa(); // or dijkstra
    }
    cost_t res = 0;
    for (int i = 0; i < esz; ++i)
        res += edg[i].f * edg[i].w;
    res /= 2;
    return res;
}

void add_edge(int v, int u, int c, int w) {
    edg[esz] = {v, u, 0, c, w};
    edg[esz + 1] = {u, v, 0, 0, -w};
    graph[v].push_back(esz);
    graph[u].push_back(esz + 1);
    esz += 2;
}

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m;
    cin >> n >> m;
    S = 0;
    T = n - 1;
    NUMV = n;
    for (int i = 0; i < m; ++i) {
        int v, u, c, w;
        cin >> v >> u >> c >> w;
        v--, u--;
        add_edge(v, u, c, w);
    }
    cost_t ans = mincost();
    cout << ans;
}

```

2.6 Мосты

```

void dfs(int v, int par) {
    vis[v] = 1;
    up[v] = tin[v] = timer++;
    for (auto u : g[v]) {
        if (!vis[u]) {
            dfs(u, v);
            up[v] = min(up[v], up[u]);
        } else if (u != par) {
            up[v] = min(up[v], tin[u]);
        }
        if (up[u] > tin[v]) {
            bridges.emplace_back(v, u);
        }
    }
}

```

2.7 Паросочетания

```

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

signed main() {
    for (int i = 0; i < s; ++i) {
        ans += dfs(i, i + 1);
    }
}

```

2.8 Точки сочленения

```

void dfs(int v, int par) {
    vis[v] = 1;
    up[v] = tin[v] = timer++;
    int child = 0;

```

```

    for (auto u : g[v]) {
        if (!vis[u]) {
            dfs(u, v);
            up[v] = min(up[v], up[u]);
            if (up[u] >= tin[v] && par != -1) {
                points.insert(v);
            }
            child++;
        } else if (u != par) {
            up[v] = min(up[v], tin[u]);
        }
    }
    if (par == -1 && child >= 2) {
        points.insert(v);
    }
}

```

2.9 Эдмондс-Карп

```

struct edge {
    int v, f, c, ind;
};

vector<edge> g[MAXN];

bool bfs(int start, int final, int W) {
    vector<int> d(MAXN, INF);
    vector<pair<int, int>> pred(MAXN);
    d[start] = 0;
    deque<int> q = {start};
    while (!q.empty()) {
        int v = q.front();
        q.pop_front();
        for (int i = 0; i < (int) g[v].size(); i++) {
            auto e = g[v][i];
            if (e.f + W <= e.c && d[e.v] > d[v] + 1) {
                d[e.v] = d[v] + 1;
                pred[e.v] = {v, i};
                q.push_back(e.v);
            }
        }
    }
    if (d[final] == INF) {
        return false;
    }
    int v = final;
    int x = INF;
    while (v != start) {
        int ind = pred[v].second;

```

```

        v = pred[v].first;
        x = min(x, g[v][ind].c - g[v][ind].f);
    }
    v = final;
    while (v != start) {
        int ind = pred[v].second;
        v = pred[v].first;
        g[v][ind].f += x;
        g[g[v][ind].v][g[v][ind].ind].f -= x;
    }
    return true;
}

signed main() {
    for (int i = 0; i < m; i++) {
        int u, v, c;
        cin >> u >> v >> c;
        g[u].push_back({v, 0, c, (int) g[v].size()});
        g[v].push_back({u, 0, 0, (int) g[u].size() -
1});
    }
    int W = (1 << 30);
    do {
        while (bfs(1, n, W));
        W /= 2;
    } while (W >= 1);
    int res = 0;
    for (auto e : g[1]) {
        res += e.f;
    }
}

```

2.10 Эйлеров цикл

```
// TODO
```

3 ДП

3.1 СХТ

```

pair<ld, ld> inter(Line a, Line b) {
    ld x = (b.b - a.b) / (a.k - b.k);
    ld y = a.k * x + a.b;
    return {x, y};
}

void add_line(ld k, ld b, vector<Line> &s, vector<pair<
ld, ld>> &pts) {

```

```

    while (s.size() >= 2) {
        pair<ld, ld> x1 = inter(s.back(), s[s.size() -
2]);
        pair<ld, ld> x2 = inter(s[s.size() - 2], {k, b
});
        if (x1 > x2) {
            break;
        }
        pts.pop_back();
        s.pop_back();
    }
    if (!s.empty()) {
        pts.push_back(inter(s.back(), {k, b}));
    }
    s.push_back({k, b});
}

ld bin_search(vector<Line> &s, ld x) {
    int l = 0, r = s.size();
    while (l + 1 < r) {
        int m = (r + l) / 2;
        auto kek = inter(s[m - 1], s[m]);
        if (kek.first >= x) {
            l = m;
        } else {
            r = m;
        }
    }
    return s[l].k * x + s[l].b;
}

```

3.2 Li Chao

```

// max

struct Line {
    int k, b;

    int f(int x) {
        return k * x + b;
    }
};

```

```

struct ST {
    vector<Line> st;

    ST(int n) {
        Line ln = {0LL, -INF};
        st.resize(4 * n, ln);
    }

```

```

}

void upd(int i, int l, int r, Line ln) {
    int child = 1;
    Line ln1 = ln;
    int m = (l + r) / 2;
    if (ln.f(m) > st[i].f(m)) {
        if (ln.k < st[i].k) {
            child = 2;
        }
        ln1 = st[i];
        st[i] = ln;
    } else {
        if (st[i].k < ln.k) {
            child = 2;
        }
    }
    if (l + 1 < r) {
        if (child == 1) {
            upd(i * 2 + 1, l, m, ln1);
        } else {
            upd(i * 2 + 2, m, r, ln1);
        }
    }

    int res(int i, int l, int r, int x) {
        if (l + 1 == r) {
            return st[i].f(x);
        }
        int m = (l + r) / 2;
        int val = st[i].f(x);
        if (x < m) {
            val = max(val, res(i * 2 + 1, l, m, x));
        } else {
            val = max(val, res(i * 2 + 2, m, r, x));
        }
        return val;
    }
};

```

3.3 SOS-dp

```
// TODO
```

3.4 HBП

```
// TODO
```

3.5 НОВП

// TODO

4 Деревья

4.1 Centroid

```
void sizes(int v, int p) {
    sz[v] = 1;
    for (auto u : g[v]) {
        if (u != p && !used[u]) {
            sizes(u, v);
            sz[v] += sz[u];
        }
    }
}

int centroid(int v, int p, int n) {
    for (int u : g[v]) {
        if (sz[u] > n / 2 && u != p && !used[u]) {
            return centroid(u, v, n);
        }
    }
    return v;
}

void dfs(int v, int p) {
    .....
    for (auto u : g[v]) {
        if (u != p && !used[u]) {
            dfs(u, v);
        }
    }
}

void solve(int v) {
    sizes(v, -1);
    .....
    for (auto u : g[v]) {
        if (!used[u]) {
            .....
            dfs(u, v);
            .....
        }
    }
    .....
    used[v] = 1;
    for (int u : g[v]) {
        if (!used[u]) {
```

```
        solve(centroid(u, v, sz[u]));
    }
}

int main() {
    sizes(0, -1);
    solve(centroid(0, -1, n));
}
```

4.2 HLD

```
const int MAXN = 50500;
const int INF = (int) 1e15;
const int L = 20;
vector<int> g[MAXN];
int sz[MAXN];
int depth[MAXN];

vector<vector<int>> up(MAXN, vector<int>(L + 1));

void dfs(int v, int p) {
    up[v][0] = p;
    for (int i = 1; i <= L; i++) {
        up[v][i] = up[up[v][i - 1]][i - 1];
    }
    for (int u : g[v]) {
        if (u != p) {
            dfs(u, v);
        }
    }
}

int lca(int u, int v) {
    if (u == v) {
        return u;
    }
    int du = depth[u], dv = depth[v];
    if (du < dv) {
        swap(du, dv);
        swap(u, v);
    }
    for (int i = L; i >= 0; i--) {
        if (du - (int) pow(2, i) >= dv) {
            u = up[u][i];
            du -= (int) pow(2, i);
        }
    }
    if (u == v) {
        return u;
```

```
    }
    for (int i = L; i >= 0; i--) {
        if (up[u][i] != up[v][i]) {
            u = up[u][i];
            v = up[v][i];
        }
    }
    return up[u][0];
}

void dfs1(int v, int p) {
    sz[v] = 1;
    for (int u : g[v]) {
        if (u != p) {
            dfs1(u, v);
            sz[v] += sz[u];
        }
    }
}

int cnt = 0;
int nn[MAXN];
int pred[MAXN];
int rup[MAXN];

void dfs2(int v, int p, int root, int dep = 0) {
    depth[v] = dep;
    nn[v] = cnt++;
    pred[v] = p;
    rup[v] = root;
    int mx = 0;
    int vert = -1;
    for (int u : g[v]) {
        if (u != p) {
            if (mx < sz[u]) {
                mx = sz[u];
                vert = u;
            }
        }
    }
    if (vert != -1) {
        dfs2(vert, v, root, dep + 1);
    }
    for (int u : g[v]) {
        if (u != p && u != vert) {
            dfs2(u, v, u, dep + 1);
        }
    }
}
```

```

ST st({});
int n;

int mx_path_up(int u, int v) {
    if (depth[u] < depth[v]) {
        swap(u, v);
    }
    int res = -INF;
    while (true) {
        int root = rup[u];
        if (depth[root] <= depth[v]) {
            res = max(res, st.rmql(0, 0, n, nn[v], nn[u]
+ 1));
            break;
        }
        res = max(res, st.rmql(0, 0, n, nn[root], nn[u]
+ 1));
        u = pred[rup[u]];
    }
    return res;
}

int mx_path(int u, int v) {
    int vert = lca(u, v);
    return max(mx_path_up(u, vert), mx_path_up(v, vert)
);
}

void change(int u, int qd) {
    st.update(0, 0, n, nn[u], qd);
}

signed main() {
    cin >> n;
    vector<int> hs(n);
    for (auto &x : hs) {
        cin >> x;
    }
    for (int i = 0; i < n - 1; i++) {
        cin >> u1 >> v1;
        g[u1].push_back(v1);
        g[v1].push_back(u1);
    }
    dfs1(1, -1);
    dfs(1, 1);
    dfs2(1, -1, 1);
    vector<int> nhs(n);
    for (int i = 1; i <= n; i++) {
        nhs[nn[i]] = hs[i - 1];
    }
}

```

```

st = *new ST(nhs);
char op;
int q;
cin >> q;
while (q--) {
    cin >> op >> v1 >> u1;
    if (op == '?') {
        cout << mx_path(u1, v1) << endl;
    } else {
        change(v1, u1);
    }
}
}

```

4.3 Link-cut

```

struct Node {
    Node *ch[2];
    Node *p;
    bool rev;
    int sz;

    Node () {
        ch[0] = nullptr;
        ch[1] = nullptr;
        p = nullptr;
        rev = false;
        sz = 1;
    }
};

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

int chnum(Node *v) {
    return v->p->ch[1] == v;
}

bool isroot(Node *v) {
    return v->p == nullptr || v->p->ch[chnum(v)] != v;
}

void push(Node *v) {
    if (v->rev) {
        if (v->ch[0])
            v->ch[0]->rev ^= 1;
        if (v->ch[1])
            v->ch[1]->rev ^= 1;
    }
}

```

```

        swap(v->ch[0], v->ch[1]);
        v->rev = false;
    }
}

void pull(Node *v) {
    v->sz = size(v->ch[1]) + size(v->ch[0]) + 1;
}

void attach(Node *v, Node *p, int num) {
    if (p)
        p->ch[num] = v;
    if (v)
        v->p = p;
}

void rotate(Node *v) {
    Node *p = v->p;
    push(p);
    push(v);
    int num = chnum(v);
    Node *u = v->ch[1 - num];
    if (!isroot(v->p))
        attach(v, p->p, chnum(p));
    else
        v->p = p->p;
    attach(u, p, num);
    attach(p, v, 1 - num);
    pull(p);
    pull(v);
}

void splay(Node *v) {
    push(v);
    while (!isroot(v)) {
        if (!isroot(v->p)) {
            if (chnum(v) == chnum(v->p))
                rotate(v->p);
            else
                rotate(v);
        }
        rotate(v);
    }
}

void expose(Node *v) {
    splay(v);
    v->ch[1] = nullptr;
    pull(v);
    while (v->p != nullptr) {

```



```
Node *p = v->p;
splay(p);
attach(v, p, 1);
pull(p);
splay(v);
}
}

void makeroot(Node *v) {
    expose(v);
    v->rev ^= 1;
    push(v);
}

void link(Node *v, Node *u) {
    makeroot(v);
    makeroot(u);
    u->p = v;
}

void cut(Node *v, Node *u) {
    makeroot(u);
    makeroot(v);
    v->ch[1] = nullptr;
    u->p = nullptr;
}

int get(Node *v, Node *u) {
    makeroot(u);
    makeroot(v);
    Node *w = u;
    while (!isroot(w))
        w = w->p;
    return (w == v ? size(v) - 1 : -1);
}

const int MAXN = 100010;
Node *nodes[MAXN];

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);
    int n, q;
    cin >> n >> q;
    for (int i = 0; i < n; ++i)
        nodes[i] = new Node();
    while (q--) {
        string s;
        Int a, b;
```

```
cin >> s >> a >> b;
a--, b--;
if (s[0] == 'g')
    cout << get(nodes[a], nodes[b]) << '\n';
else if (s[0] == 'l')
    link(nodes[a], nodes[b]);
else
    cut(nodes[a], nodes[b]);
}
}
```

5 Другое

5.1 attribute_packed

```
// TODO
```

5.2 ordered_set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;

typedef tree<int, null_type, less<>, rb_tree_tag,
            tree_order_statistics_node_update> ordered_set;

//st.find_by_order(index);
//st.order_of_key(key);
```

5.3 pragma

```
#pragma GCC optimize("Ofast,fast-math,unroll-loops,no-
stack-protector,inline")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,sse4.1,
sse4.2,avx,avx2,abm,mmx,popcnt")
```

5.4 Аллокатор Копелиовича

```
// TODO
```

6 Математика

6.1 2-SAT

```
// TODO
```

6.2 FFT mod

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

using namespace std;

typedef long long ll;
#define int long long

const int MOD = 998244353; // 7*17 * 2^23 + 1
const int GEN = 3;
//const int MOD = 7340033; // 7 * 2^20 + 1
//const int MOD = 5;
//const int MOD = 469762049; // 7 * 2^26 + 1
//const int GEN = 30;

const int LOG = 20;
const int MAXN = 1 << LOG;
int tail[MAXN + 1];
int OMEGA[MAXN + 1];

int binpow(int x, int p) {
    int res = 1;
    while (p > 0) {
        if (p & 1)
            res = res * 1ll * x % MOD;
        x = x * 1ll * x % MOD;
        p >>= 1;
    }
    return res;
}

int omega(int n, int k) {
    return OMEGA[MAXN / n * k];
}

int gettail(int x, int lg) {
    return tail[x] >> (LOG - lg);
}

void calcomega() {
    ll one = binpow(GEN, (MOD - 1) / MAXN);
    OMEGA[0] = 1;
    for (int i = 1; i < MAXN; ++i) {
        OMEGA[i] = OMEGA[i - 1] * one % MOD;
    }
}
```

```

void calctail() {
    int n = MAXN;
    for (int x = 0; x < n; ++x) {
        int res = 0;
        for (int i = 0; i < LOG; ++i) {
            res += ((x >> i) & 1) << (LOG - i - 1);
        }
        tail[x] = res;
    }
}

// Without precalc, tail[], OMEGA[]
//
//ll omega(int n, int k) {
//    return binpow(GEN, (MOD - 1) / n * k);
//}
//
//int gettail(int x, int lg) {
//    int res = 0;
//    for (int i = 0; i < lg; ++i)
//        res += ((x >> i) & 1) << (lg - i - 1);
//    return res;
//}

void fft(vector<int> &A, int lg) {
    int n = 1 << lg;
    for (int i = 0; i < n; ++i) {
        int j = gettail(i, lg);
        if (i < j)
            swap(A[i], A[j]);
    }
    for (int len = 2; len <= n; len *= 2) {
        for (int i = 0; i < n; i += len) {
            for (int j = 0; j < len / 2; ++j) {
                auto v = A[i + j];
                auto u = A[i + j + len / 2] * 111 *
                    omega(len, j) % MOD;
                A[i + j] = (v + u) % MOD;
                A[i + j + len / 2] = (v - u + MOD) %
                    MOD;
            }
        }
    }
}

int inverse(int x) {
    return binpow(x, MOD - 2);
}

```

```

void invfft(vector<int> &A, int lg) {
    int n = 1 << lg;
    fft(A, lg);
    for (auto &el : A)
        el = el * 111 * inverse(n % MOD) % MOD;
    reverse(A.begin() + 1, A.end());
}

vector<int> mul(vector<int> A, vector<int> B) {
    int lg = 32 - __builtin_clz(A.size() + B.size() -
        1);
    int n = 1 << lg;
    A.resize(n, 0);
    B.resize(n, 0);
    fft(A, lg);
    fft(B, lg);
    for (int i = 0; i < n; ++i)
        A[i] = A[i] * 111 * B[i] % MOD;
    invfft(A, lg);
    return A;
}

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    calctail();
    calcomega();
    int n, m;
    cin >> n >> m;
    vector<int> A(n), B(m);
    for (int &el : A)
        cin >> el;
    for (int &el : B)
        cin >> el;
    auto C = mul(A, B);
    for (auto el : C)
        cout << el << '␣';
}

```

6.3 FFT

```

const double PI = acos(-1);
const int LOG = 19;
const int MAXN = 1 << LOG;

struct comp {
    double x, y;
    comp() : x(0), y(0) {}
    comp(double x, double y) : x(x), y(y) {}

```

```

    comp(int x) : x(x), y(0) {}
    comp operator+(const comp &o) const {
        return comp(x + o.x, y + o.y);
    }
    comp operator-(const comp &o) const {
        return comp(x - o.x, y - o.y);
    }
    comp operator*(const comp &o) const {
        return comp(x * o.x - y * o.y, x * o.y + y * o.
            x);
    }
    comp operator/(const int k) const {
        return comp(x / k, y / k);
    }
    comp conj() const {
        return comp(x, -y);
    }
};

comp OMEGA[MAXN + 10];
int tail[MAXN + 10];

comp omega(int n, int k) {
    return OMEGA[MAXN / n * k];
}

void calcomega() {
    for (int i = 0; i < MAXN; ++i) {
        double x = 2 * PI * i / MAXN;
        OMEGA[i] = {cos(x), sin(x)};
    }
}

void calctail() {
    tail[0] = 0;
    for (int i = 1; i < MAXN; ++i) {
        tail[i] = (tail[i >> 1] >> 1) | ((i & 1) << (
            LOG - 1));
    }
}

void fft(vector<comp> &A) {
    int n = A.size();
    for (int i = 0; i < n; ++i) {
        if (i < tail[i])
            swap(A[i], A[tail[i]]);
    }
    for (int len = 2; len <= n; len *= 2) {
        for (int i = 0; i < n; i += len) {
            for (int j = 0; j < len / 2; ++j) {

```

```

        auto v = A[i + j];
        auto u = A[i + j + len / 2] * omega(len
, j);

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

void fft2(vector<comp> &A, vector<comp> &B) {
    int n = A.size();
    vector<comp> C(n);
    for (int i = 0; i < n; ++i) {
        C[i].x = A[i].x;
        C[i].y = B[i].x;
    }
    fft(C);
    C.push_back(C[0]);
    for (int i = 0; i < n; ++i) {
        A[i] = (C[i] + C[n - i].conj()) / 2;
        B[i] = (C[i] - C[n - i].conj()) / 2 * comp(0,
-1);
    }
}

void invfft(vector<comp> &A) {
    fft(A);
    for (auto &el : A)
        el = el / MAXN;
    reverse(A.begin() + 1, A.end());
}

vector<int> mul(vector<int> &a, vector<int> &b) {
    vector<comp> A(MAXN, 0), B(MAXN, 0);
    for (int i = 0; i < (int)a.size(); ++i)
        A[i] = a[i];
    for (int i = 0; i < (int)b.size(); ++i)
        B[i] = b[i];
    fft2(A, B);
    for (int i = 0; i < MAXN; ++i)
        A[i] = A[i] * B[i];
    invfft(A);
    vector<int> c(MAXN);
    for (int i = 0; i < MAXN; ++i) {
        int x = round(A[i].x);
        c[i] = x;
    }
    while (!c.empty() && c.back() == 0)
        c.pop_back();
    return c;
}

```

```

}

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    calcomega();
    calctail();
    string sa, sb;
    cin >> sa >> sb;
    reverse(sa.begin(), sa.end());
    reverse(sb.begin(), sb.end());
    vector<int> a(sa.size()), b(sb.size());
    bool minus = false;
    if (sa.back() == '-') {
        minus ^= true;
        sa.pop_back();
    }
    if (sb.back() == '-') {
        minus ^= true;
        sb.pop_back();
    }
    for (int i = 0; i < (int)sa.size(); ++i)
        a[i] = sa[i] - '0';
    for (int i = 0; i < (int)sb.size(); ++i)
        b[i] = sb[i] - '0';
    auto c = mul(a, b);
    int shift = 0;
    for (int i = 0; i < (int)c.size(); ++i) {
        int x = c[i] + shift;
        c[i] = x % 10;
        shift = x / 10;
    }
    while (shift > 0) {
        c.push_back(shift % 10);
        shift /= 10;
    }
    while (!c.empty() && c.back() == 0)
        c.pop_back();
    if (c.empty()) {
        cout << 0;
        return 0;
    }
    if (minus)
        cout << '-';
    for (int i = c.size() - 1; i >= 0; --i)
        cout << c[i];
}

```

6.4 Гаусс

```

// TODO битовый
// TODO дабловый
// TODO по модулю

```

6.5 Диофантовы уравнения

```

def bezout(a, b):
    x, xx, y, yy = 1, 0, 0, 1
    while b:
        q = a // b
        a, b = b, a % b
        x, xx = xx, x - xx*q
        y, yy = yy, y - yy*q
    return (a, x, y)

```

```

a, b, c = map(int, input().split())
d, k, l = bezout(a, b)
q = c // d
x, y = q*k, q*l
if c % d == 0:
    x -= x // (b//d) * (b//d)
    y = (c-a*x) // b

```

6.6 КТО

```

// TODO

```

6.7 Код Грея

```

// TODO

```

6.8 Линейное решето

```

// TODO

```

6.9 Ро-Поллард

```

// TODO

```

6.10 Символ Якоби, Лежандра

```

// TODO

```

7 Строки

7.1 Z-функция

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

7.2 Ахо-Корасик

```
int cntv = 1;

void add(string &s) {
    static int cnt_s = 1;
    int v = 0;
    for (char el : s) {
        if (go[v][el - 'a'] == 0) {
            go[v][el - 'a'] = cntv;
            par[cntv] = v;
            par_c[cntv] = el;
            cntv++;
        }
        v = go[v][el - 'a'];
    }
    term[v].push_back(cnt_s++);
}

void bfs() {
    deque<int> q = {0};
    while (!q.empty()) {
        int v = q.front();
        q.pop_front();
        if (v > 0) {
            if (par[v] == 0) {
```

```
                suf[v] = 0;
            } else {
                suf[v] = go[suf[par[v]]][par_c[v] - 'a'
];
            }
            g[suf[v]].push_back(v);
        }
        for (int c = 0; c < 26; c++) {
            if (go[v][c] == 0) {
                go[v][c] = go[suf[v]][c];
            } else {
                q.push_back(go[v][c]);
            }
        }
    }
}
```

7.3 Префикс-функция

```
int main() {
    vector<int> pref(n, 0);
    int ans = 0;
    for (int i = 1; i < n; i++) {
        while (ans > 0 && s[ans] != s[i]) {
            ans = pref[ans - 1];
        }
        if (s[i] == s[ans]) {
            ans++;
        }
        pref[i] = ans;
    }
}
```

7.4 Суффиксный автомат

```
struct Node {
    int go[26];
    int suf;
    int len;
};
Node verts[MAXN];
int cnt_v = 1;
int max_v = 0;

void add(char c) {
    c -= 'a';
    int nv = cnt_v++;
    verts[max_v].go[c] = nv;
```

```
verts[nv].len = verts[max_v].len + 1;
int v = max_v;
while (v != 0) {
    if (verts[verts[v].suf].go[c] == 0) {
        v = verts[v].suf;
        verts[v].go[c] = nv;
        verts[nv].len = max(verts[nv].len, verts[v
].len + 1);
        continue;
    }
    int vv = verts[v].suf, uu = verts[vv].go[c];
    if (verts[vv].len + 1 == verts[uu].len) {
        verts[nv].suf = uu;
        break;
    }
    int v2 = cnt_v++;
    for (int c2 = 0; c2 < 26; c2++) {
        verts[v2].go[c2] = verts[uu].go[c2];
    }
    int to = verts[vv].go[c];
    do {
        if (verts[vv].go[c] == to) {
            verts[vv].go[c] = v2;
            verts[v2].len = max(verts[v2].len,
verts[vv].len + 1);
        } else {
            break;
        }
        vv = verts[vv].suf;
    } while (vv != 0);
    if (verts[vv].go[c] == to) {
        verts[vv].go[c] = v2;
        verts[v2].len = max(verts[v2].len, verts[vv
].len + 1);
    }
    verts[v2].suf = verts[uu].suf;
    verts[uu].suf = verts[nv].suf = v2;
    break;
}
max_v = nv;
}
```

7.5 Суффиксный массив

```
vector<int> build_suff_arr(string s) {
    s.push_back('#');
    int n = s.size();
    vector<int> suf(n), c(n);
    vector<int> cnt(MAX);
```

```

for (int i = 0; i < n; i++) {
    cnt[s[i] - '#']++;
}
vector<int> pos(MAX);
for (int i = 1; i < MAX; i++) {
    pos[i] = pos[i - 1] + cnt[i - 1];
}
for (int i = 0; i < n; i++) {
    suf[pos[s[i] - '#']++] = i;
}
int cls = -1;
for (int i = 0; i < n; i++) {
    if (i == 0 || s[suf[i]] != s[suf[i - 1]]) {
        cls++;
    }
    c[suf[i]] = cls;
}
for (int L = 1; L < n; L *= 2) {
    fill(cnt.begin(), cnt.end(), 0);
    for (int i = 0; i < n; i++) {
        cnt[c[i]]++;
    }
    pos[0] = 0;
    for (int i = 1; i < n; i++) {
        pos[i] = pos[i - 1] + cnt[i - 1];
    }
    for (int i = 0; i < n; i++) {
        suf[i] = (suf[i] - L + n) % n;
    }
    vector<int> new_suf(n), new_c(n);
    for (int i = 0; i < n; i++) {
        int where = pos[c[suf[i]]];
        new_suf[where] = suf[i];
        pos[c[suf[i]]]++;
    }
    cls = -1;
    for (int i = 0; i < n; i++) {
        if (i == 0) {
            cls++;
            new_c[new_suf[i]] = cls;
            continue;
        }
        pair<int, int> prev = {c[new_suf[i - 1]], c[
            (new_suf[i - 1] + L) % n]};
        pair<int, int> now = {c[new_suf[i]], c[
            (new_suf[i] + L) % n]};
        if (prev != now) {
            cls++;
        }
        new_c[new_suf[i]] = cls;
    }
}

```

```

    }
    swap(c, new_c);
    swap(suf, new_suf);
}
vector<int> res;
for (int i = 1; i < n; i++) {
    res.push_back(suf[i]);
}
return res;
}

vector<int> lcp_neighboring(string &s, vector<int> &suf)
{
    int n = s.size();
    vector<int> lcp(n), where(n);
    for (int i = 0; i < n; i++) {
        where[suf[i]] = i;
    }
    int k = 0;
    for (int j = 0; j < n; j++) {
        int pos = where[j];
        if (pos == n - 1) {
            k = 0;
            lcp[pos] = 0;
        } else {
            k = max(OLL, k - 1);
            while (s[j + k] == s[suf[pos + 1] + k]) {
                k++;
                if (j + k >= n || suf[pos + 1] + k >= n)
                    break;
            }
            lcp[pos] = k;
        }
    }
    return lcp;
}

int sol(int k, string s) {
    int n = s.size();
    vector<int> suf = build_suff_arr(s);
    vector<int> lcp = lcp_neighboring(s, suf);
    vector<int> where(n);
    for (int i = 0; i < n; i++) {
        where[suf[i]] = i;
    }
    Sparse_Table st(lcp);
    int ans = 0;
    for (int i = 0; i < n - k; i++) {

```

```

        ans += st.rmq(where[i], where[i + k]);
    }
    return ans;
}

```

8 Структуры данных

8.1 Disjoint Sparse Table

```
// TODO
```

8.2 Segment Tree Beats

```
// min=, sum
```

```

struct ST {
    vector<int> st, mx, mx_cnt, sec_mx;

    ST(int n) {
        st.resize(n * 4, 0);
        mx.resize(n * 4, 0);
        mx_cnt.resize(n * 4, 0);
        sec_mx.resize(n * 4, 0);
        build(0, 0, n);
    }

    void upd_from_children(int v) {
        st[v] = st[v * 2 + 1] + st[v * 2 + 2];
        mx[v] = max(mx[v * 2 + 1], mx[v * 2 + 2]);
        mx_cnt[v] = 0;
        sec_mx[v] = max(sec_mx[v * 2 + 1], sec_mx[v * 2
            + 2]);
        if (mx[v * 2 + 1] == mx[v]) {
            mx_cnt[v] += mx_cnt[v * 2 + 1];
        } else {
            sec_mx[v] = max(sec_mx[v], mx[v * 2 + 1]);
        }
        if (mx[v * 2 + 2] == mx[v]) {
            mx_cnt[v] += mx_cnt[v * 2 + 2];
        } else {
            sec_mx[v] = max(sec_mx[v], mx[v * 2 + 2]);
        }
    }

    void build(int i, int l, int r) {
        if (l + 1 == r) {
            st[i] = mx[i] = 0;
            mx_cnt[i] = 1;

```

```

        sec_mx[i] = -INF;
        return;
    }
    int m = (r + 1) / 2;
    build(i * 2 + 1, l, m);
    build(i * 2 + 2, m, r);
    upd_from_children(i);
}

void push_min_eq(int v, int val) {
    if (mx[v] > val) {
        st[v] -= (mx[v] - val) * mx_cnt[v];
        mx[v] = val;
    }
}

void push(int i) {
    push_min_eq(i * 2 + 1, mx[i]);
    push_min_eq(i * 2 + 2, mx[i]);
}

void update(int i, int l, int r, int ql, int qr,
int val) {
    if (mx[i] <= val) {
        return;
    }
    if (ql == l && qr == r && sec_mx[i] < val) {
        push_min_eq(i, val);
        return;
    }
    push(i);
    int m = (r + 1) / 2;
    if (qr <= m) {
        update(i * 2 + 1, l, m, ql, qr, val);
    } else if (ql >= m) {
        update(i * 2 + 2, m, r, ql, qr, val);
    } else {
        update(i * 2 + 1, l, m, ql, m, val);
        update(i * 2 + 2, m, r, m, qr, val);
    }
    upd_from_children(i);
}

int sum(int i, int l, int r, int ql, int qr) {
    if (l == ql && r == qr) {
        return st[i];
    }
    push(i);
    int m = (r + 1) / 2;
    if (qr <= m) {

```

```

        return sum(i * 2 + 1, l, m, ql, qr);
    }
    if (ql >= m) {
        return sum(i * 2 + 2, m, r, ql, qr);
    }
    return sum(i * 2 + 1, l, m, ql, m) + sum(i * 2
+ 2, m, r, m, qr);
}
};

```

8.3 ДД по неявному

```

pair<Node *, Node *> split(Node *now, int k) {
    if (!now) {
        return {nullptr, nullptr};
    }
    if (size(now->l) + 1 <= k) {
        auto ans = split(now->r, k - 1 - size(now->l));
        now->r = ans.first;
        update_size(now);
        return {now, ans.second};
    }
    auto ans = split(now->l, k);
    now->l = ans.second;
    update_size(now);
    return {ans.first, now};
}

Node *merge(Node *l, Node *r) {
    if (!l) {
        return r;
    }
    if (!r) {
        return l;
    }
    if (l->y <= r->y) {
        auto ans = merge(l->r, r);
        l->r = ans;
        update_size(l);
        return l;
    }
    auto ans = merge(l, r->l);
    r->l = ans;
    update_size(r);
    return r;
}

Node *insert(Node *root, int pos) {
    auto r = split(root, pos);

```

```

    Node *nn = new Node(pos);
    root = merge(r.first, nn);
    root = merge(root, r.second);
    return root;
}

Node *to_begin(Node *root, int l, int r) {
    auto a = split(root, l);
    auto b = split(a.second, r - l);
    return merge(b.first, merge(a.first, b.second));
}

```

8.4 ДД

```

pair<Node *, Node *> split(Node *now, ll x) {
    if (!now) {
        return {nullptr, nullptr};
    }
    if (now->x <= x) {
        auto ans = split(now->r, x);
        now->r = ans.first;
        update_sum(now);
        return {now, ans.second};
    }
    auto ans = split(now->l, x);
    now->l = ans.second;
    update_sum(now);
    return {ans.first, now};
}

Node *merge(Node *l, Node *r) {
    if (!l) {
        return r;
    }
    if (!r) {
        return l;
    }
    if (l->y <= r->y) {
        auto ans = merge(l->r, r);
        l->r = ans;
        update_sum(l);
        return l;
    }
    auto ans = merge(l, r->l);
    r->l = ans;
    update_sum(r);
    return r;
}

```

```

Node *insert(Node *root, ll val) {
    Node *new_v = new Node(val);
    auto ans = split(root, val);
    return merge(merge(ans.first, new_v), ans.second);
}

Node *del(Node *root, ll val) {
    auto ans = split(root, val);
    auto ans1 = split(ans.first, val - 1);
    return merge(ans1.first, ans.second);
}

ll get_sum(Node *root, ll l, ll r) {
    if (!root) {
        return 0;
    }
    auto ans = split(root, l - 1);
    auto ans2 = split(ans.second, r);
    if (!ans2.first) {
        merge(merge(ans.first, ans2.first), ans2.second);
    }
    return 0;
}
ll res = ans2.first->sum_;
merge(merge(ans.first, ans2.first), ans2.second);
return res;
}

```

8.5 Персистентное ДД

```
// TODO
```

8.6 Персистентное ДО

```

struct ST {
    vector<Node*> roots;
    int n;

    ST(vector<int> &a) {
        start = a;
        n = a.size();
        roots.push_back(nullptr);
        build(roots[0], 0, n);
    }

    void update(Node *&now, Node *old, int l, int r,
        int pos, int qd) {
        if (l + 1 == r) {

```

```

        now = new_node(qd);
        return;
    }
    now = new_node();
    int m = (l + r) / 2;
    if (pos < m) {
        now->r = old->r;
        update(now->l, old->l, l, m, pos, qd);
    } else {
        now->l = old->l;
        update(now->r, old->r, m, r, pos, qd);
    }
    now->sm = now->l->sm + now->r->sm;
}
};

```

8.7 Спарсы

```

struct Sparse_Table {
    vector<vector<int>> st;
    vector<int> max2;

    Sparse_Table(vector<int> &a) {
        int n = a.size();
        st.emplace_back();
        for (int i = 0; i < n; i++) {
            st[0].push_back(a[i]);
        }
        for (int i = 1; (1 << i) <= n; i++) {
            st.emplace_back();
            for (int p = 0; p + (1 << i) <= n; p++) {
                st[i].push_back(min(st[i - 1][p], st[i - 1][p + (1 << (i - 1))]]));
            }
        }
        max2.resize(n + 1);
        max2[1] = 0;
        for (int i = 2; i <= n; i++) {
            max2[i] = max2[i / 2] + 1;
        }
    }

    int rmq(int l, int r) {
        r++;
        int i = max2[r - 1];
        return min(st[i][l], st[i][r - (1 << i)]);
    }
};

```

8.8 Фенвик (pref += x)

```

// a[left..right] += delta; get_sum a[l..pos]
//
//void update(left, right, delta)
// T1.add(left, delta)
// T1.add(right + 1, -delta);
// T2.add(left, delta * (left - 1))
// T2.add(right + 1, -delta * right);
//
//int getSum(pos)
// return T1.sum(pos) * pos - T2.sum(pos)

```

8.9 Фенвик

```
// Нумерация с 1
```

```

struct Fenwick_tree {
    vector<vector<vector<int>>> ft;

    Fenwick_tree(int n) {
        ft.resize(n + 1, vector<vector<int>>>(n + 1,
            vector<int>(n + 1)));
    }

    void upd(int x, int y, int z, int d) {
        for (int x1 = x; x1 < ft.size(); x1 += x1 & -x1) {
            for (int y1 = y; y1 < ft[x1].size(); y1 += y1 & -y1) {
                for (int z1 = z; z1 < ft[x1][y1].size(); z1 += z1 & -z1) {
                    ft[x1][y1][z1] += d;
                }
            }
        }
    }

    int rsq(int x, int y, int z) {
        int ans = 0;
        for (int x1 = x; x1 > 0; x1 -= x1 & -x1) {
            for (int y1 = y; y1 > 0; y1 -= y1 & -y1) {
                for (int z1 = z; z1 > 0; z1 -= z1 & -z1) {
                    ans += ft[x1][y1][z1];
                }
            }
        }
        return ans;
    }
};

```

<pre> } int sum_3d(int x1, int x2, int y1, int y2, int z1, int z2) {</pre>	<pre> int ans = rsq(x2, y2, z1 - 1) + rsq(x1 - 1, y2, z2) - rsq(x1 - 1, y2, z1 - 1); ans += rsq(x2, y1 - 1, z2); ans -= rsq(x2, y1 - 1, z1 - 1) + rsq(x1 - 1, y1 - 1, z2) - rsq(x1 - 1, y1 - 1, z1 - 1);</pre>	<pre> return rsq(x2, y2, z2) - ans; } };</pre>
--	---	---