数据结构

linkCutTree

标准版

```
template<class Info>
struct linkCutTree {
    struct node {
        int s[2], p, r;
        Info v;
    };
    int n;
    vector<node> t;

    int &fa(int x) { return t[x].p; }
    int &lc(int x) { return t[x].s[0]; }
    int &rc(int x) { return t[x].s[1]; }
    // notroot
    bool pos(int x) {
        return t[t[x].p].s[0] == x || t[t[x].p].s[1] == x;
    }
    // 不能以0开头
    linkCutTree(int n) : n(n) { t.resize(n + 1); t[0].v.defaultclear(); }

    void pull(int x) {
        t[x].v.up(t[lc(x)].v, t[rc(x)].v);
    }

    void push(int x) {
        if (t[x].r) {
            swap(lc(x), rc(x));
            t[lc(x)].v.reve();
            t[rc(x)].v.reve();
            t[rc(x)].r ^= 1;
            t[lc(x)].r ^= 1;
            t[x].r = 0;
        }
    }

    // maintain
    void mt(int x) {
        if (pos(x)) mt(fa(x));
        push(x);
    }
    // rotate
    void rtt(int x) {
        int y = fa(x), z = fa(y);
        int k = rc(y) == x;
        if (pos(y))
            t[z].s[rc(z) == y] = x;
        fa(x) = z;
        t[y].s[k] = t[x].s[k ^ 1];
```

```
        fa(t[x].s[k ^ 1]) = y;
        t[x].s[k ^ 1] = y;
        fa(y) = x;
        pull(y);
    }
    void splay(int x) {
        mt(x);
        while (pos(x)) {
            int y = fa(x), z = fa(y);
            if (pos(y))
                ((rc(z) == y) ^ (rc(y) == x))
                ? rtt(x) : rtt(y);
            rtt(x);
        }
        pull(x);
    }
    // access
    void acc(int x) {
        for (int y = 0; x;) {
            splay(x);
            rc(x) = y;
            pull(x);
            y = x;
            x = fa(x);
        }
    }

    // makeroot
    void mrt(int x) {
        acc(x);
        splay(x);
        t[x].r ^= 1;
    }

    //y变成原树和辅助树的根
    const Info &split(int x, int y) {
        mrt(x);
        acc(y);
        splay(y);
        return t[y].v;
    }

    // findroot
    int find(int x) {
        acc(x);
        splay(x);
        while (lc(x))
            push(x), x = lc(x);
        splay(x);
        return x;
    }

    void link(int x, int y) {
        mrt(x);
        if (find(y) != x) fa(x) = y;
    }
```

```
    void cut(int x, int y) {
        mrt(x);
        if (find(y) == x
            && fa(y) == x && !lc(y)) {
            rc(x) = fa(y) = 0;
            pull(x);
        }
    }

    void modify(int x, const Info &val) {
        splay(x);
        t[x].v.modify(val);
        pull(x);
    }

    bool same(int x, int y) {
        mrt(x);
        return find(y) == x;
    }
    node &operator[](int x) {
        return t[x];
    }
    void DFS(int u, int dep = 0) {
        if (!u) {
            return;
        }
        push(u);
        for (auto i : {0, 1}) {
            if (i == 1) {
                cerr << string(dep, '\t');
                cerr << u << ' ' << t[u].v << endl;
            }
            DFS(t[u].s[i], dep + 1);
        }
    }
    void dfs(int u) {
# ifndef ONLINE_JUDGE
        cerr << "\nlct rooted u: " << u << ", P = " << t[u].p << '\n';
        DFS(u);
# endif
    }
};


struct Info {
    void reve() {}
    void modify(const Info& rhs) {}
    void up(const Info &lhs, const Info &rhs) {}
    // default
    void clear() {}
    friend ostream &operator<<(ostream &cout, Info x) {
        return cout;
    };
};
```

```
using Tree = linkCutTree<Info>;
```

LazyLinkCutTree

```
template<class Info, class Tag>
struct LazyLinkCutTree {
    struct node {
        int s[2], p, r;
        Info v;
        Tag t;
    };
    int n;
    vector<node> t;

    int &fa(int x) { return t[x].p; }
    int &lc(int x) { return t[x].s[0]; }
    int &rc(int x) { return t[x].s[1]; }
    bool pos(int x) {
        return t[t[x].p].s[0] == x || t[t[x].p].s[1] == x;
    }
    // 不能以0开头
    LazyLinkCutTree(int n) : n(n) {
        t.resize(n + 1);
        t[0].t.clear();
        t[0].v.clear();
    }

    void pull(int x) {
        t[x].v.up(t[lc(x)].v, t[rc(x)].v);
    }

    void apply(int x, const Tag &rhs) {
        if (x) {
            t[x].v.apply(rhs);
            t[x].t.apply(rhs);
        }
    }

    void push(int x) {
        if (t[x].r) {
            swap(lc(x), rc(x));
            t[lc(x)].v.reve();
            t[rc(x)].v.reve();
            t[rc(x)].r ^= 1;
            t[lc(x)].r ^= 1;
            t[x].r = 0;
        }
        if (bool(t[x].t)) {
            apply(lc(x), t[x].t);
            apply(rc(x), t[x].t);
            t[x].t.clear();
        }
    }

    void mt(int x) {
```

```
    if (pos(x)) mt(fa(x));
    push(x);
}

void rtt(int x) {
    int y = fa(x), z = fa(y);
    int k = rc(y) == x;
    if (pos(y))
        t[z].s[rc(z) == y] = x;
    fa(x) = z;
    t[y].s[k] = t[x].s[k ^ 1];
    fa(t[x].s[k ^ 1]) = y;
    t[x].s[k ^ 1] = y;
    fa(y) = x;
    pull(y);
}

void splay(int x) {
    mt(x);
    while (pos(x)) {
        int y = fa(x), z = fa(y);
        if (pos(y))
            ((rc(z) == y) ^ (rc(y) == x))
            ? rtt(x) : rtt(y);
        rtt(x);
    }
    pull(x);
}

void acc(int x) {
    for (int y = 0; x;) {
        splay(x);
        rc(x) = y;
        pull(x);
        y = x;
        x = fa(x);
    }
}

void mrt(int x) {
    acc(x);
    splay(x);
    t[x].r ^= 1;
}

//y变成原树和辅助树的根
const Info &split(int x, int y) {
    mrt(x);
    acc(y);
    splay(y);
    return t[y].v;
}

int find(int x) {
    acc(x);
    splay(x);
```

```
        while (lc (x))
            push(x), x = lc(x);
        splay(x);
        return x;
    }

    void link(int x, int y) {
        mrt(x);
        if (find(y) != x) fa(x) = y;
    }

    void cut(int x, int y) {
        mrt(x);
        if (find(y) == x && fa(y) == x && !lc(y)) {
            rc(x) = fa(y) = 0;
            pull(x);
        }
    }

    void modify(int x, const Info &val) {
        splay(x);
        t[x].v.modify(val);
        pull(x);
    }

    void lineModify(int u, int v, const Tag &rhs) {
        split(u, v);
        apply(v, rhs);
    }

    bool same(int x, int y) {
        mrt(x);
        return find(y) == x;
    }

    node &operator[](int x) {
        return t[x];
    }
    void DFS(int u, int dep = 0) {
        if (!u) {
            return;
        }
        push(u);
        for (auto i : {0, 1}) {
            if (i == 1) {
                cerr << string(dep, '\t');
                cerr << u << ' ' << t[u].v << ' ' << t[u].t << endl;
            }
            DFS(t[u].s[i], dep + 1);
        }
    }
    void dfs(int u) {
# ifndef ONLINE_JUDGE
        cerr << "\nlct rooted u: " << u << ", P = " << t[u].p << '\n';
        DFS(u);
# endif
```

```
    }
};

struct Tag {
    void apply(const Tag &rhs) {}
    void clear() {}
    constexpr operator bool() {
        return false;
    }
    friend ostream &operator<<(ostream &cout, Tag x) {
        return cout;
    }
};

struct Info {
    void reve() {}
    void modify(const Info& rhs) {}
    void up(const Info &lhs, const Info &rhs) {}
    void apply(const Tag &rhs) {}
    void clear() {}
    friend ostream &operator<<(ostream &cout, Info x) {
        return cout;
    }
};

using Tree = LazyLinkCutTree<Info, Tag>;
```

维护子树信息

```
template<class Info>
struct linkCutTree {
    struct node {
        int s[2], p, r;
        Info v;
    };
    int n;
    vector<node> t;

    int &fa(int x) { return t[x].p; }
    int &lc(int x) { return t[x].s[0]; }
    int &rc(int x) { return t[x].s[1]; }
    bool pos(int x) {
        return t[t[x].p].s[0] == x || t[t[x].p].s[1] == x;
    }
    // 不能以0开头
    linkCutTree(int n) : n(n) { t.resize(n + 1); t[0].v.clear(); }

    void pull(int x) {
        // debug(x);
        t[x].v.up(t[lc(x)].v, t[rc(x)].v);
    }
    void push(int x) {
        if (t[x].r) {
            swap(lc(x), rc(x));
            t[lc(x)].v.reve();
```

```
            t[rc(x)].v.reve();
            t[rc(x)].r ^= 1;
            t[lc(x)].r ^= 1;
            t[x].r = 0;
        }
    }
    void mt(int x) {
        if (pos(x)) mt(fa(x));
        push(x);
    }

    void rtt(int x) {
        int y = fa(x), z = fa(y);
        int k = rc(y) == x;
        if (pos(y))
            t[z].s[rc(z) == y] = x;
        fa(x) = z;
        t[y].s[k] = t[x].s[k ^ 1];
        fa(t[x].s[k ^ 1]) = y;
        t[x].s[k ^ 1] = y;
        fa(y) = x;
        pull(y);
    }
    void splay(int x) {
        mt(x);
        while (pos(x)) {
            int y = fa(x), z = fa(y);
            if (pos(y))
                ((rc(z) == y) ^ (rc(y) == x)) ? rtt(x) : rtt(y);
            rtt(x);
        }
        pull(x);
    }

    void acc(int x) {
        for (int y = 0; x;) {
            splay(x);
            t[x].v.vup(t[rc(x)].v);
            rc(x) = y;
            t[x].v.rv(t[rc(x)].v);
            pull(x);
            y = x;
            x = fa(x);
        }
    }

    void mrt(int x) {
        acc(x);
        splay(x);
        t[x].v.reve();
        t[x].r ^= 1;
    }

    //x变为原树的根，y变成辅助树的根
    const Info &split(int x, int y) {
        mrt(x);
```

```
        acc(y);
        splay(y);
        return t[y].v;
    }

    int find(int x) {
        acc(x);
        splay(x);
        while (lc(x))
            push(x), x = lc(x);
        splay(x);
        return x;
    }

    void link(int x, int y) {
        mrt(x);
        mrt(y);
        if (find(y) != x) {
            fa(x) = y;
            t[y].v.vup(t[x].v);
        }
    }

    void cut(int x, int y) {
        mrt(x);
        if (find(y) == x && fa(y) == x && !lc(y)) {
            rc(x) = fa(y) = 0;
            pull(x);
        }
    }

    void modify(int x, const Info &val) {
        mrt(x);
        t[x].v.modify(val);
        pull(x);
    }

    bool same(int x, int y) {
        mrt(x);
        return find(y) == x;
    }
    node &operator[](int x) {
        return t[x];
    }
    void DFS(int u, int dep = 0) {
        if (!u) {
            return;
        }
        push(u);
        for (auto i : {0, 1}) {
            if (i == 1) {
                cerr << string(dep, '\t');
                cerr << u << ' ' << t[u].v << endl;
            }
            DFS(t[u].s[i], dep + 1);
        }
```

```
    }
    void dfs(int u) {
# ifndef ONLINE_JUDGE
        cerr << "\nlct rooted u: " << u << ", P = " << t[u].p << '\n';
        DFS(u);
# endif
    }
};

struct Info {
    void reve() {}
    void modify(const Info& rhs) {}
    void vup(const Info &rhs) {}
    void rv(const Info &rhs) {}
    void up(const Info &lhs, const Info &rhs) {}
    void clear() {}
# ifndef ONLINE_JUDGE
    friend ostream &operator<<(ostream &cout, Info u) {
        return cout;
    }
# endif
};

using Tree = linkCutTree<Info>;
```

动态维护直径

```
struct Info {
    i64 x;
    i64 ans = 0;
    array<i64, 2> max{};
    i64 sum;
    multiset<i64> set;
    multiset<i64> vans;
    void reve() {
        swap(max[0], max[1]);
    }
    void modify(const Info& rhs) {
        x = rhs.x;
    }
    void vup(const Info &rhs) {
        // debug(rhs.max[0]);
        if (rhs.max[0] != 0) {
            set.insert(rhs.max[0]);
        }
        if (rhs.ans != 0) {
            vans.insert(rhs.ans);
        }
    }
    void rv(const Info &rhs) {
        if (rhs.max[0] != 0) {
            set.erase(set.find(rhs.max[0]));
        }
        if (rhs.ans != 0) {
            vans.erase(vans.find(rhs.ans));
```

```
        }
    }
    void up(const Info &lhs, const Info &rhs) {
        sum = lhs.sum + rhs.sum + x;
        i64 F = 0, S = 0;
        if (set.size()) {
            auto it = set.rbegin();
            F = *it;
            if (set.size() >= 2) {
                it ++;
                S = *it;
            }
        }
        i64 R = std::max(F, rhs.max[0]);
        i64 L = std::max(F, lhs.max[1]);
        max[0] = std::max(lhs.max[0], lhs.sum + x + R);
        max[1] = std::max(rhs.max[1], rhs.sum + x + L);
        array<i64, 4> vec{F, S, lhs.max[1], rhs.max[0]};
        sort(vec.rbegin(), vec.rend());
        ans = std::max({vec[0] + vec[1] + x, lhs.ans, rhs.ans});
        if (!vans.empty()) {
            ans = std::max(ans, *vans.rbegin());
        }
    }
    void clear() {}
# ifndef ONLINE_JUDGE
    friend ostream &operator<<(ostream &cout, Info u) {
        return cout << u.x << ' ' << u.ans << ' ' << u.set << ' ' << u.max;
    }
# endif
};
```

RMQ

catTree

```
template<typename T, class F>
struct catTree {
    static constexpr int B = 24;
    int n;
    array<vector<T>, B> a;
    F merge;
    catTree() {}
    catTree(const vector<T> &_init, F merge) {
        init(_init, merge);
    }
    void init(const vector<T> &_init, F merge) {
        this->merge = merge;
        n = _init.size();
        a[0] = _init;
        for (int k = 1, w = 4; k <= __lg(n); k += 1, w <<= 1) {
            a[k].assign(n, {});
```

```
            for (int l = 0, mid = w / 2, r = std::min(w, n); mid < n; l += w, mid
+= w, r = std::min(r + w, n)) {
                a[k][mid - 1] = a[0][mid - 1];
                for (int i = mid - 2; i >= l; i -= 1) {
                    a[k][i] = merge(a[0][i], a[k][i + 1]);
                }
                a[k][mid] = a[0][mid];
                for (int i = mid + 1; i < r; i += 1) {
                    a[k][i] = merge(a[0][i], a[k][i - 1]);
                }
            }
        }
        // debug(a);
    }
    T operator() (int l, int r) {
        if (r - l == 1) {
            return a[0][l];
        }
        int k = __lg(l ^ (r - 1));
        return merge(a[k][l], a[k][r - 1]);
    }
};
```

状压rmq

```
template<class T,
    class Cmp = std::less<T>>
struct RMQ {
    const Cmp cmp = Cmp();
    static constexpr unsigned B = 64;
    using u64 = unsigned long long;
    int n;
    std::vector<std::vector<T>> a;
    std::vector<T> pre, suf, ini;
    std::vector<u64> stk;
    RMQ() {}
    RMQ(const std::vector<T> &v) {
        init(v);
    }
    void init(const std::vector<T> &v) {
        n = v.size();
        pre = suf = ini = v;
        stk.resize(n);
        if (!n) {
            return;
        }
        const int M = (n - 1) / B + 1;
        const int lg = std::__lg(M);
        a.assign(lg + 1, std::vector<T>(M));
        for (int i = 0; i < M; i++) {
            a[0][i] = v[i * B];
            for (int j = 1; j < B && i * B + j < n; j++) {
                a[0][i] = std::min(a[0][i], v[i * B + j], cmp);
            }
        }
```

```
        for (int i = 1; i < n; i++) {
            if (i % B) {
                pre[i] = std::min(pre[i], pre[i - 1], cmp);
            }
        }
        for (int i = n - 2; i >= 0; i--) {
            if (i % B != B - 1) {
                suf[i] = std::min(suf[i], suf[i + 1], cmp);
            }
        }
        for (int j = 0; j < lg; j++) {
            for (int i = 0; i + (2 << j) <= M; i++) {
                a[j + 1][i] = std::min(a[j][i], a[j][i + (1 << j)], cmp);
            }
        }
        for (int i = 0; i < M; i++) {
            const int l = i * B;
            const int r = std::min(1U * n, l + B);
            u64 s = 0;
            for (int j = l; j < r; j++) {
                while (s && cmp(v[j], v[std::__lg(s) + l])) {
                    s ^= 1ULL << std::__lg(s);
                }
                s |= 1ULL << (j - l);
                stk[j] = s;
            }
        }
    }
    T operator()(int l, int r) {
        if (l / B != (r - 1) / B) {
            T ans = std::min(suf[l], pre[r - 1], cmp);
            l = l / B + 1;
            r = r / B;
            if (l < r) {
                int k = std::__lg(r - l);
                ans = std::min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
            }
            return ans;
        } else {
            int x = B * (l / B);
            return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + l];
        }
    }
};
```

ST表

```
template<typename T, class F>
struct SparseTable {
    int n;
    constexpr static int B = 24;
    array<vector<T>, B> a;
    F merge;
    SparseTable() {}
```

```
    SparseTable(const vector<T> &info, F merge) {
        init(info, merge);
    }
    void init(const vector<T> &info, F merge) {
        this->merge = merge;
        n = info.size();
        for (int i = 0; i < B; i += 1) {
            a[i].assign(n, {});
        }
        a[0] = info;
        for (int k = 1; k <= __lg(n); k += 1) {
            for (int i = n - (1 << k); i >= 0; i -= 1) {
                a[k][i] = merge(a[k - 1][i], a[k - 1][i + (1 << k - 1)]);
            }
        }
    }
    T operator() (int l, int r) {
        int k = __lg(r - l);
        return merge(a[k][l], a[k][r - (1 << k)]);
    }
};
```

并查集

标准

```
struct DSU {
    std::vector<int> f, siz;

    DSU() {}
    DSU(int n) {
        init(n);
    }

    void init(int n) {
        f.resize(n);
        std::iota(f.begin(), f.end(), 0);
        siz.assign(n, 1);
    }

    int find(int x) {
        while (x != f[x]) {
            x = f[x] = f[f[x]];
        }
        return x;
    }

    bool same(int x, int y) {
        return find(x) == find(y);
    }

    bool merge(int x, int y) {
        x = find(x);
```

```
        y = find(y);
        if (x == y) {
            return false;
        }
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }

    int size(int x) {
        return siz[find(x)];
    }
};
```

可持久化

```
struct PDSU {
    int n;
    struct node;
    using Tp = Base<node>;
    struct node {
        int f, siz;
        Tp ch[2];
    };
    Tp news() {
        Tp t = Tp::news();
        return t;
    }
    vector<Tp> root;
    PDSU(): n(0) {}
    PDSU(int _n, int _m = 0) {
        init(_n, _m);
    }
    void build(Tp t, int l, int r) {
        if (r - l == 1) {
            t->f = l;
            t->siz = 1;
            return;
        }
        int m = (l + r) / 2;
        t->ch[0] = news(), t->ch[1] = news();
        build(t->ch[0], l, m), build(t->ch[1], m, r);
    }
    void init(int _n, int m = 0) {
        n = _n;
        root.reserve(m + 1);
        root.push_back(news());
        build(root.back(), 0, n);
    }
    void modify0(Tp &t0, Tp &t1, Tp v, int l, int r, int x) {
        if (r - l == 1) {
            t1->f = v->f;
            t1->siz = t0->siz;
```

```
            return;
        }
        int m = (l + r) >> 1;
        if (m > x) {
            t1->ch[0] = news();
            t1->ch[1] = t0->ch[1];
            modify0(t0->ch[0], t1->ch[0], v, l, m, x);
        } else {
            t1->ch[0] = t0->ch[0];
            t1->ch[1] = news();
            modify0(t0->ch[1], t1->ch[1], v, m, r, x);
        }
    }
    void modify0(int x, Tp v, Tp t0, Tp t1) {
        modify0(t0, t1, v, 0, n, x);
    }
    void modify1(Tp &t0, Tp &t1, Tp v, int l, int r, int x) {
        if (r - l == 1) {
            t1->f = t0->f;
            t1->siz = t0->siz + v->siz;
            return;
        }
        int m = (l + r) >> 1;
        if (m > x) {
            t1->ch[0] = news();
            t1->ch[1] = t0->ch[1];
            modify1(t0->ch[0], t1->ch[0], v, l, m, x);
        } else {
            t1->ch[0] = t0->ch[0];
            t1->ch[1] = news();
            modify1(t0->ch[1], t1->ch[1], v, m, r, x);
        }
    }
    void modify1(int x, Tp v, Tp t0, Tp t1) {
        modify1(t0, t1, v, 0, n, x);
    }
    void dfs(Tp t, int l, int r) {
        if (r - l == 1) {
            cerr << "(" << t->f << ", " << t->siz << "), ";
            return;
        }
        int m = (l + r) >> 1;
        dfs(t->ch[0], l, m);
        dfs(t->ch[1], m, r);
    }
    void dfs(int time) {
        dfs(root[time], 0, n);
        cerr << endl;
    }
    Tp Query(Tp t, int l, int r, int x) {
        while (r - l != 1) {
            int m = (l + r) / 2;
            if (m > x)
                t = t->ch[0], r = m;
            else
                t = t->ch[1], l = m;
```

```
        }
        return t;
    }
    Tp Query(int x, Tp t) {
        return Query(t, 0, n, x);
    }
    Tp find(int x, Tp t) {
        Tp fa = Query(x, t);
        return fa->f == x ?
            fa : find(fa->f, t);
    }
    bool same(int u, int v, int t = -1) {
        t = t == -1 ? int(root.size()) - 1 : t;
        root.push_back(root[t]);
        Tp lhs = find(u, root[t]), rhs = find(v, root[t]);
        return lhs->f == rhs->f;
    }
    void merge(int u, int v, int t = -1) {
        t = t == -1 ? int(root.size()) - 1 : t;
        Tp lhs = find(u, root[t]), rhs = find(v, root[t]);
        if (lhs->f == rhs->f) {
            root.push_back(root[t]);
            return;
        }
        if (lhs->siz < rhs->siz) {
            swap(lhs, rhs);
        }
        Tp cur0 = news();
        modify0(rhs->f, lhs, root[t], cur0);
        Tp cur1 = news();
        modify1(lhs->f, rhs, cur0, cur1);
        root.push_back(cur1);
    }
    void roll(int t) {
        root.push_back(root[t]);
    }
};
using DSU = PDSU;
```

可撤回

```
struct DSU {
    vector<int> fa, siz;
    vector<array<int, 4>> h;
    vector<i64> lazy;

    DSU() {}

    DSU(int n) {
        init(n);
    }

    void init(int n) {
        fa.resize(n);
        iota(fa.begin(), fa.end(), 0);
```

```
        siz.assign(n, 1);
        lazy.assign(n, 0);
    }

    int find(int x) {
        while (x != fa[x]) {
            x = fa[x];
        }
        return x;
    }

    int size(int x) {
        return siz[find(x)];
    }

    bool same(int u, int v) {
        return find(u) == find(v);
    }

    void merge(int u, int v) {
        int x = find(u);
        int y = find(v);
        if (x == y) return;
        if (siz[x] < siz[y]) std::swap(x, y);
        h.push_back({x, y, siz[x], fa[y]});
        siz[x] = siz[x] + siz[y];
        fa[y] = x;
        int p = y;
        lazy[y] -= lazy[x];
    }

    int clock() {
        return h.size();
    }

    void roll(int to) {
        while (h.size() > to) {
            auto [u, v, sizu, fav] = h.back();
            siz[u] = sizu;
            fa[v] = fav;
            h.pop_back();
            lazy[v] += lazy[u];
        }
    }
};
```

平衡树

set

FHQtreap

```cpp
/**
 * FHQ_treap set卡常:
 * 1.递归改非递归       o
 * 2.insert split优化   o
 */
# include <ext/random>
__gnu_cxx::sfmt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

template<typename Info>
struct FHQ_treap {
    struct Node;
    using Tp = Base<Node>;
    struct Node {
        Tp ch[2];
        Info val;
        int siz, key;
    };

    Tp root;

    void pull(Tp t) {
        t->siz = t->ch[0]->siz + 1 + t->ch[1]->siz;
    }
    // by val
    pair<Tp, Tp> split(Tp t, Info val) {
        if (!t) {
            return {t, t};
        }
        Tp ohs;
        if (t->val < val) {
            tie(t->ch[1], ohs) = split(t->ch[1], val);
            pull(t);
            return {t, ohs};
        } else {
            tie(ohs, t->ch[0]) = split(t->ch[0], val);
            pull(t);
            return {ohs, t};
        }
    }

    Tp merge(Tp u, Tp v) {
        if (!u | !v) return u.x | v.x;
        if (u->key < v->key) {
            u->ch[1] = merge(u->ch[1], v);
            pull(u);
            return u;
        } else {
            v->ch[0] = merge(u, v->ch[0]);
            pull(v);
            return v;
        }
    }
```

```
// set operator
    void insert(Tp &t, Tp v) {
        if (!t) {
            t = v;
            // ps;
            return;
        }
        if (t->key < v->key) {
            tie(v->ch[0], v->ch[1]) = split(t, v->val);
            t = v;
            pull(t);
            return;
        }
        t->siz += 1;
        insert(t->ch[v->val > t->val ||
            (t->val == v->val && int(rng()) >= 0)], v);
        pull(t);
    }

    void insert(Info v) {
        Tp t = Tp::__new();
        t->key = rng();
        t->val = v;
        t->siz = 1;
        insert(root, t);
    }

    void erase(Tp &t, Info v) {
        if (t->val == v) {
            t = merge(t->ch[0], t->ch[1]);
            return;
        } else {
            // t->siz -= 1;
            erase(t->ch[v > t->val], v);
            pull(t);
        }
    }

    void erase(Info v) {
        erase(root, v);
    }
    // by val
    int less(Info v) {
        Tp t = root;
        int less_siz = 0;
        while (t) {
            if (t->val >= v) {
                t = t->ch[0];
            } else {
                less_siz += t->ch[0]->siz + 1;
                t = t->ch[1];
            }
        }
        return less_siz;
    }
    // from zero
```

```
    Tp rank(Tp t, int k) {
        k += 1;
        while (true) {
            if (t->ch[0]->siz >= k) {
                t = t->ch[0];
            } else if (t->ch[0]->siz + 1 < k) {
                k -= t->ch[0]->siz + 1;
                t = t->ch[1];
            } else
                break;
        }
        return t;
    }
    // from zero
    Tp operator[] (int k) {
        return rank(root, k);
    }
    // by val
    static constexpr int inf = std::numeric_limits<int>::max();
    Info prev(Info v) {
        Tp t = root, p;
        while (t) {
            if (t->val < v) {
                p = t;
                t = t->ch[1];
            } else {
                t = t->ch[0];
            }
        }
        return p ? p->val : -inf;
    }
    // by val
    Info next(Info v) {
        Tp t = root, p;
        while (t) {
            if (t->val <= v) {
                t = t->ch[1];
            } else {
                p = t;
                t = t->ch[0];
            }
        }
        return p ? p->val : inf;
    }
    void dfs(Tp t, int dep = 0) {
        if (!t) {
            return;
        }
        dfs(t->ch[0], dep + 1);
        for (int i = 0; i < dep; i += 1) cerr << '\t';
        cerr << t->val << ' ' << t->key << '\n';
        dfs(t->ch[1], dep + 1);
    }
    void dfs() {return dfs(root);}
};
```

替罪羊树

```
constexpr double alpha = 0.75;
template<typename Info>
struct scapegoat_tree {
    struct node;
    using Tp = Base<node>;
    struct node {
        Tp ch[2];
        Info val;
        int siz, fac;
        bool exist;
    };

    Tp root = 0;

    Tp __new() {
        return Tp::__new();
    }

    void reset(Tp &t) {
        t->siz = t->fac = 1;
        t->exist = true;
        t->ch[0] = t->ch[1] = 0;
    }

    void reset(Tp &t, Info val) {
        t->siz = t->fac = 1;
        t->exist = true;
        t->ch[0] = t->ch[1] = 0;
        t->val = val;
    }

    Tp __new(Info val) {
        Tp t = __new();
        reset(t, val);
        return t;
    }

    scapegoat_tree() {}

    bool imbalance(Tp t) {
        return max({t->ch[0]->siz, t->ch[1]->siz})
                    > t->siz * alpha
        || t->siz * alpha > t->fac;
    }

    vector<Tp> v;
    void collect(Tp t) {
        if (!t) return;
        collect(t->ch[0]);
        if (t->exist)
            v.push_back(t);
        collect(t->ch[1]);
    }
```

```
    void pull(Tp t) {
        t->siz = t->ch[0]->siz + 1 + t->ch[1]->siz;
        t->fac = t->ch[0]->fac + t->exist + t->ch[1]->fac;
    }
    void lift(int l, int r, Tp &t) {
        if (l == r) {
            t = v[l];
            reset(t);
            return;
        }
        int m = l + r >> 1;
        while (l < m && v[m]->val == v[m - 1]->val) {
            -- m;
        }
        t = v[m];
        if (l != m) lift(l, m - 1, t->ch[0]);
        else t->ch[0] = 0;
        lift(m + 1, r, t->ch[1]);
        pull(t);
    }
    void rebuild(Tp &t) {
        v.clear();
        collect(t);
        if (v.empty()) {
            t = 0;
            return;
        }
        lift(0, v.size() - 1, t);
    }

    void check(Tp &t, Tp E) {
        if (t == E) return;
        if (imbalance(t)) {
            rebuild(t);
            return;
        }
        check(t->ch[E->val >= t->val], E);
    }

    void insert(Tp &t, Info val) {
        if (!t) {
            t = __new(val);
            // dfs();
            check(root, t);
            return;
        }
        t->siz ++;
        t->fac ++;
        insert(t->ch[val >= t->val], val);
    }
    void insert(Info val) {
        insert(root, val);
    }
    void erase(Tp &t, Info val) {
        if (t->exist && t->val == val) {
            t->exist = false;
```

```
            t->fac --;
            check(root, t);
            return;
        }
        t->fac--;
        erase(t->ch[val >= t->val], val);
    }
    void erase(Info val) {
        erase(root, val);
    }
    int less(Info val) {
        Tp t = root;
        int less = 0;
        while (t) {
            if (val <= t->val) {
                t = t->ch[0];
            } else {
                less += t->exist + t->ch[0]->fac;
                t = t->ch[1];
            }
        }
        return less;
    }
    // from zero
    Tp operator[](int k) {
        k += 1;
        Tp t = root;
        while (t) {
            if (t->ch[0]->fac >= k) {
                t = t->ch[0];
            } else if (t->ch[0]->fac + t->exist < k) {
                k -= t->ch[0]->fac + t->exist;
                t = t->ch[1];
            } else
                break;
        }
        return t;
    }
    void dfs(Tp t, int dep = 0) {
        if (!t) return;
        dfs(t->ch[0], dep + 1);
        for (int i = 0; i < dep; i += 1) cerr << '\t';
        cerr << t->val << ' ' << t->siz << ' ' << t->fac << endl;
        dfs(t->ch[1], dep + 1);
    }
    void dfs() { return dfs(root); }
}; //scapegoat_tree

using scet = scapegoat_tree<int>;
```

区间操作

FhqTreap

```
# include <ext/random>
__gnu_cxx::sfmt19937
rng(std::chrono::steady_clock::now().time_since_epoch().count());

struct node;
using Tp = Base<node>;

struct node {
    Tp ch[2];
    int siz, k;
    i64 val;
    i64 sum;
};

Tp news() {
    Tp t = Tp::news();
    t->k = rng();
    return t;
}

Tp news(auto val) {
    Tp t = news();
    t->val = val;
    t->siz = 1;
    t->sum = val;
    return t;
}

void pull(Tp t) {
    t->siz = t->ch[0]->siz + 1 + t->ch[1]->siz;
    t->sum = t->ch[0]->sum + t->val + t->ch[1]->sum;
}

// to [-inf, val) and [val, inf]
pair<Tp, Tp> split1(Tp t, auto val) {
    if (!t) {
        return {t, t};
    }
    Tp u;
    if (t->val < val) {
        tie(t->ch[1], u) = split1(t->ch[1], val);
        pull(t);
        return {t, u};
    } else {
        tie(u, t->ch[0]) = split1(t->ch[0], val);
        pull(t);
        return {u, t};
    }
}
```

```
// to [1, rk) and [rk, n]
pair<Tp, Tp> split2(Tp t, int rk) {
    if (!t) {
        return {t, t};
    }
    Tp u;
    if (rk <= t->ch[0]->siz) {
        tie(u, t->ch[0]) = split2(t->ch[0], rk);
        pull(t);
        return {u, t};
    } else if (rk > t->ch[0]->siz + 1) {
        tie(t->ch[1], u) = split2(t->ch[1], rk - 1 - t->ch[0]->siz);
        pull(t);
        return {t, u};
    } else {
        u = t->ch[0];
        t->ch[0] = 0;
        pull(t);
        return {u, t};
    }
}

Tp merge(Tp u, Tp v) {
    if (!u | !v) return u.x | v.x;
    if (u->k < v->k) {
        u->ch[1] = merge(u->ch[1], v);
        pull(u);
        return u;
    } else {
        v->ch[0] = merge(u, v->ch[0]);
        pull(v);
        return v;
    }
}

// 2056

void dfs(Tp t, int dep = 0) {
    if (!t) {
        return;
    }
    dfs(t->ch[0], dep + 1);
    for (int i = 0; i < dep; i += 1) cerr << '\t';
    cerr << t->val << ' ' << t->sum << ' ' << t->siz << '\n';
    dfs(t->ch[1], dep + 1);
}

// less_to_val_siz
int less_to_val(Tp t, auto val) {
    int less_siz = 0;
    while (t) {
        if (t->val >= val) {
            t = t->ch[0];
        } else {
            less_siz += t->ch[0]->siz + 1;
            t = t->ch[1];
```

```
        }
    }
    return less_siz;
}


Tp rank(Tp t, int rk) {
    while (true) {
        if (t->ch[0]->siz >= rk) {
            t = t->ch[0];
        } else if (t->ch[0]->siz + 1 < rk) {
            rk -= t->ch[0]->siz + 1;
            t = t->ch[1];
        } else
            break;
    }
    return t;
}

// prev_to_val
Tp prev(Tp t, auto val) {
    Tp p;
    while (t) {
        if (t->val < val) {
            p = t;
            t = t->ch[1];
        } else {
            t = t->ch[0];
        }
    }
    return p;
}
// next_to_val
Tp next(Tp t, auto val) {
    Tp p;
    while (t) {
        if (t->val <= val) {
            t = t->ch[1];
        } else {
            p = t;
            t = t->ch[0];
        }
    }
    return p;
}
```

FhqTreap带懒标

```
# include <ext/random>
__gnu_cxx::sfmt19937
rng(std::chrono::steady_clock::now().time_since_epoch().count());

struct node;
using Tp = Base<node>;
```

```
struct node {
    Tp ch[2];
    int siz, k;
    i64 val;
    i64 tag;
};

Tp news() {
    Tp t = Tp::news();
    t->k = rng();
    return t;
}

Tp news(auto val) {
    Tp t = news();
    t->val = val;
    t->siz = 1;
    t->tag = 0;
    return t;
}

void ap(Tp t, auto tag) {
    if (t) {
        t->val += tag;
        t->tag += tag;
    }
}

void push(Tp t) {
    if (t->tag) {
        ap(t->ch[0], t->tag);
        ap(t->ch[1], t->tag);
        t->tag = 0;
    }
}

void pull(Tp t) {
    t->siz = t->ch[0]->siz + 1 + t->ch[1]->siz;
}

// to [-inf, val) and [val, inf]
pair<Tp, Tp> split1(Tp t, auto val) {
    if (!t) {
        return {t, t};
    }
    push(t);
    Tp u;
    if (t->val < val) {
        tie(t->ch[1], u) = split1(t->ch[1], val);
        pull(t);
        return {t, u};
    } else {
        tie(u, t->ch[0]) = split1(t->ch[0], val);
        pull(t);
        return {u, t};
    }
```

```
}


// to [1, rk) and [rk, n]
pair<Tp, Tp> split2(Tp t, int rk) {
    if (!t) {
        return {t, t};
    }
    push(t);
    Tp u;
    if (rk <= t->ch[0]->siz) {
        tie(u, t->ch[0]) = split2(t->ch[0], rk);
        pull(t);
        return {u, t};
    } else if (rk > t->ch[0]->siz + 1) {
        tie(t->ch[1], u) = split2(t->ch[1], rk - 1 - t->ch[0]->siz);
        pull(t);
        return {t, u};
    } else {
        u = t->ch[0];
        t->ch[0] = 0;
        pull(t);
        return {u, t};
    }
}

Tp merge(Tp u, Tp v) {
    if (!u | !v) return u.x | v.x;
    if (u->k < v->k) {
        push(u);
        u->ch[1] = merge(u->ch[1], v);
        pull(u);
        return u;
    } else {
        push(v);
        v->ch[0] = merge(u, v->ch[0]);
        pull(v);
        return v;
    }
}

// 2056

void dfs(Tp t, int dep = 0) {
    if (!t) {
        return;
    }
    dfs(t->ch[0], dep + 1);
    for (int i = 0; i < dep; i += 1) cerr << '\t';
    cerr << t->val << ' ' << t->tag << '\n';
    dfs(t->ch[1], dep + 1);
}

// less_to_val_siz
int less_to_val(Tp t, auto val) {
    int less_siz = 0;
```

```
    while (t) {
        push(t);
        if (t->val >= val) {
            t = t->ch[0];
        } else {
            less_siz += t->ch[0]->siz + 1;
            t = t->ch[1];
        }
    }
    return less_siz;
}


Tp rank(Tp t, int rk) {
    while (true) {
        push(t);
        if (t->ch[0]->siz >= rk) {
            t = t->ch[0];
        } else if (t->ch[0]->siz + 1 < rk) {
            rk -= t->ch[0]->siz + 1;
            t = t->ch[1];
        } else
            break;
    }
    return t;
}

// prev_to_val
Tp prev(Tp t, auto val) {
    Tp p;
    while (t) {
        push(t);
        if (t->val < val) {
            p = t;
            t = t->ch[1];
        } else {
            t = t->ch[0];
        }
    }
    return p;
}
// next_to_val
Tp next(Tp t, auto val) {
    Tp p;
    while (t) {
        push(t);
        if (t->val <= val) {
            t = t->ch[1];
        } else {
            p = t;
            t = t->ch[0];
        }
    }
    return p;
}
```

splay

```
struct node;
using Tp = Base<node>;
struct node {
    Tp ch[2], p;
    i64 val;
    i64 tag;
    i64 sum;
    int siz;
};

Tp news() {
    return Tp::news();
}

Tp news(auto val) {
    Tp t = news();
    t->val = val;
    t->siz = 1;
    t->sum = val;
    t->tag = 0;
}

bool pos(Tp t) {
    return t->p->ch[1] == t;
}

void apply(Tp t, auto tag) {
    if (t) {
        t->val += tag;
        t->sum += 1ll * siz * tag;
        t->tag += tag;
    }
};

void push(Tp t) {
    if (t->tag) {
        apply(t->ch[0], t->tag);
        apply(t->ch[1], t->tag);
        t->tag = decltype(t->tag)();
    }
}

void pull(Tp t) {
    t->siz = t->ch[0]->siz + 1 + t->ch[1]->siz;
    t->sum = t->ch[0]->sum + 1 + t->ch[1]->sum;
}

void rotate(Tp t) {
    Tp q = t->p;
    int x = !pos(t);
    q->ch[!x] = t->ch[x];
    if (t->ch[x]) t->ch[x]->p = q;
```

```
    t->p = q->p;
    if (q->p) q->p->ch[pos(q)] = t;
    t->ch[x] = q;
    q->p = t;
    pull(q);
}

void pushall(Tp t) {
    if (t->p) pushall(t->p);
    push(t);
}

void splay(Tp t, Tp top = 0) {
    pushall(t);
    while (t->p != top) {
        if (t->p->p != top)
            rotate(pos(t) ^ pos(t->p) ? t : t->p);
        rotate(t);
    }
    pull(t);
}

pair<Tp, Tp> split1(Tp t, auto x) {
    if (!t) {
        return {t, t};
    }
    Tp v = 0;
    Tp j = t;
    for (Tp i = t; i; ) {
        push(i);
        j = i;
        if (i->val >= x) {
            v = i;
            i = i->ch[0];
        } else {
            i = i->ch[1];
        }
    }

    splay(j);
    if (!v) {
        return {j, 0};
    }

    splay(v);

    Tp u = v->ch[0];
    if (u) {
        v->ch[0] = u->p = 0;
        pull(v);
    }
    return {u, v};
}

// 从 1 开始
Tp findK(Tp &t, int k) {
```

```
    int mid = k;
    while (true) {
        push(t);
        if (k > t->ch[0]->siz + 1) {
            k -= t->ch[0]->siz + 1;
            t = t->ch[1];
        } else if (k <= t->ch[0]->siz) {
            t = t->ch[0];
        } else {
            break;
        }
    }
    splay(t);
    return t;
}

// [1, x) and [x, n]
pair<Tp, Tp> split2(Tp t, int x) {
    if (t->siz < x) {
        return {t, 0};
    }

    findK(t, x);

    Tp u = t->ch[0];
    if (u) {
        t->ch[0] = u->p = 0;
        pull(t);
    }
    return {u, t};
}

Tp merge(Tp l, Tp r) {
    if (!l || !r) {
        return l.x | r.x;
    }
    Tp i = l;
    push(i);
    for (; i->ch[1]; i = i->ch[1], push(i));
    splay(i);
    i->ch[1] = r;
    r->p = i;
    pull(i);
    return i;
}

void dfs(Tp t, int dep = 0) {
    if (!t) {
        return;
    }
    push(t);
    dfs(t->ch[0], dep + 1);
    for (int i = 0; i < dep; i += 1) cerr << '\t';
    cerr << t->val << "\n";
    dfs(t->ch[1], dep + 1);
}
```

```
// 写出来之后感觉不太会用到，所以没测，谨慎使用
```

带父亲的FhqTreap

```
# include <ext/random>
__gnu_cxx::sfmt19937
rng(std::chrono::steady_clock::now().time_since_epoch().count());

struct node;
using Tp = Base<node>;

struct node {
    Tp ch[2];
    Tp p;
    int siz, k;
    i64 val;
    i64 tag;
    int i;
};

Tp news() {
    Tp t = Tp::news();
    t->k = rng();
    return t;
}

Tp news(auto val) {
    Tp t = news();
    t->val = val;
    t->siz = 1;
    t->tag = 0;
    return t;
}

void apply(Tp t, auto tag) {
    if (t) {
        t->val += tag;
        t->tag += tag;
    }
}

void push(Tp t) {
    if (t->tag) {
        apply(t->ch[0], t->tag);
        apply(t->ch[1], t->tag);
        t->tag = 0;
    }
}

void pull(Tp t) {
    t->siz = t->ch[0]->siz + 1 + t->ch[1]->siz;
}
```

```
// to [-inf, val) and [val, inf]
pair<Tp, Tp> split1(Tp t, auto val) {
    if (!t) {
        return {t, t};
    }
    push(t);
    Tp u;
    if (t->val < val) {
        t->ch[1]->p = 0;
        tie(t->ch[1], u) = split1(t->ch[1], val);
        t->ch[1]->p = t;
        pull(t);
        return {t, u};
    } else {
        t->ch[0]->p = 0;
        tie(u, t->ch[0]) = split1(t->ch[0], val);
        t->ch[0]->p = t;
        pull(t);
        return {u, t};
    }
}


// to [1, rk) and [rk, n]
pair<Tp, Tp> split2(Tp t, int rk) {
    if (!t) {
        return {t, t};
    }
    push(t);
    Tp u;
    if (rk <= t->ch[0]->siz) {
        t->ch[0]->p = 0;
        tie(u, t->ch[0]) = split2(t->ch[0], rk);
        t->ch[0]->p = t;
        pull(t);
        return {u, t};
    } else if (rk > t->ch[0]->siz + 1) {
        t->ch[1]->p = 0;
        tie(t->ch[1], u) = split2(t->ch[1], rk - 1 - t->ch[0]->siz);
        t->ch[1]->p = t;
        pull(t);
        return {t, u};
    } else {
        u = t->ch[0];
        u->p = 0;
        t->ch[0] = 0;
        pull(t);
        return {u, t};
    }
}

Tp merge(Tp u, Tp v) {
    if (!u | !v) return u.x | v.x;
    if (u->k < v->k) {
        push(u);
        u->ch[1]->p = 0;
```

```
        u->ch[1] = merge(u->ch[1], v);
        u->ch[1]->p = u;
        pull(u);
        return u;
    } else {
        push(v);
        v->ch[0]->p = 0;
        v->ch[0] = merge(u, v->ch[0]);
        v->ch[0]->p = v;
        pull(v);
        return v;
    }
}

void pushAll(Tp t) {
    if (t->p) {
        pushAll(t->p);
    }
    push(t);
}

// 2056

void dfs(Tp t, int dep = 0) {
    if (!t) {
        return;
    }
    push(t);
    dfs(t->ch[0], dep + 1);
    for (int i = 0; i < dep; i += 1) cerr << '\t';
    cerr << t->val << ' ' << t->i << ' ' << t->tag << '\n';
    dfs(t->ch[1], dep + 1);
}

// less_to_val_siz
int less_to_val(Tp t, auto val) {
    int less_siz = 0;
    while (t) {
        push(t);
        if (t->val >= val) {
            t = t->ch[0];
        } else {
            less_siz += t->ch[0]->siz + 1;
            t = t->ch[1];
        }
    }
    return less_siz;
}


Tp rank(Tp t, int rk) {
    while (true) {
        push(t);
        if (t->ch[0]->siz >= rk) {
            t = t->ch[0];
        } else if (t->ch[0]->siz + 1 < rk) {
```

```
            rk -= t->ch[0]->siz + 1;
            t = t->ch[1];
        } else
            break;
    }
    return t;
}

// prev_to_val
Tp prev(Tp t, auto val) {
    Tp p;
    while (t) {
        push(t);
        if (t->val < val) {
            p = t;
            t = t->ch[1];
        } else {
            t = t->ch[0];
        }
    }
    return p;
}
// next_to_val
Tp next(Tp t, auto val) {
    Tp p;
    while (t) {
        push(t);
        if (t->val <= val) {
            t = t->ch[1];
        } else {
            p = t;
            t = t->ch[0];
        }
    }
    return p;
}
```

可持久化

```
# include <ext/random>
__gnu_cxx::sfmt19937
rng(std::chrono::steady_clock::now().time_since_epoch().count());

struct node;
using Tp = Base<node>;

struct node {
    Tp ch[2];
    int siz, k;
    i64 val;
    i64 tag;
};
```

```
Tp news() {
    Tp t = Tp::news();
    t->k = rng();
    return t;
}

Tp news(Tp u) {
    if (!u) {
        return u;
    }
    Tp p = Tp::news();
    *p = *u;
    return p;
}

void ap(Tp t, auto tag) {
    if (!t) {
        return;
    }
    t->val += tag;
    t->tag += tag;
}

void push(Tp t) {
    if (t->tag) {
        t->ch[0] = news(t->ch[0]);
        t->ch[1] = news(t->ch[1]);
        ap(t->ch[0], t->tag);
        ap(t->ch[1], t->tag);
        t->tag = decltype(t->tag)();
    }
}

void pull(Tp t) {
    t->siz = t->ch[0]->siz + 1 + t->ch[1]->siz;
}


pair<Tp, Tp> split1(Tp &t, auto val) {
    if (!t) {
        return {0, 0};
    }
    t = news(t);
    push(t);
    Tp u;
    if (t->val < val) {
        tie(t->ch[1], u) = split1(t->ch[1], val);
        pull(t);
        return {t, u};
    } else {
        tie(u, t->ch[0]) = split1(t->ch[0], val);
        pull(t);
        return {u, t};
    }
}
```

```
pair<Tp, Tp> split2(Tp t, int rk) {
    if (!t) {
        return {t, t};
    }
    push(t);
    t = news(t);
    Tp u;
    if (rk <= t->ch[0]->siz) {
        tie(u, t->ch[0]) = split2(t->ch[0], rk);
        pull(t);
        return {u, t};
    } else {
        tie(t->ch[1], u) = split2(t->ch[1], rk - 1 - t->ch[0]->siz);
        pull(t);
        return {t, u};
    }
}

template<bool isNew = false>
Tp merge(Tp u, Tp v) {
    if (!u | !v) return u.x | v.x;
    if (u->key < v->key) {
        push(u);
        if (isNew) {
            u = __new(u);
        }
        u->ch[1] = merge<isNew>(u->ch[1], v);
        pull(u);
        return u;
    } else {
        push(v);
        if (isNew) {
            v = __new(v);
        }
        v->ch[0] = merge<isNew>(u, v->ch[0]);
        pull(v);
        return v;
    }
}
```

参考旧版

FHQtreap

```
/**
 * FHQ_treap 卡常:
 * 1.递归改非递归       x
 * 2.insert split优化   o
 * 3.build 优化         o
 */

__gnu_cxx::sfmt19937
rng(std::chrono::steady_clock::now().time_since_epoch().count());

template<typename Info, typename Tag>
```

```
struct FHQ_treap {
    struct Node;
    using Tp = u32_p<Node>;

    using T = typename Info::T;
    struct Node {
        Tp ch[2];
        Info info;
        int key;
        Tag tag;
        bool rev;
    };

    Tp __new() {
        Tp t = Tp::__new();
        t->key = rng();
        return t;
    }

    void apply(Tp t, const Tag &tag) {
        if (t) {
            t->info.apply(tag);
            t->tag.apply(tag);
        }
    }

    void push(Tp t) {
        if (t->rev) {
            swap(t->ch[0], t->ch[1]);
            t->ch[0]->rev ^= 1;
            t->ch[0]->info.reve();
            t->ch[1]->rev ^= 1;
            t->ch[1]->info.reve();
            t->rev = 0;
        }
        if (t->tag) {
            apply(t->ch[0], t->tag);
            apply(t->ch[1], t->tag);
            t->tag = Tag();
        }
    }

    void pull(Tp t) {
        t->info.up(t->ch[0]->info, t->ch[1]->info);
    }

    pair<Tp, Tp> split_by_val(Tp t, T val) {
        if (!t) {
            return {t, t};
        }
        // push(t);
        Tp ohs;
        if (t->info.val < val) {
            tie(t->ch[1], ohs) = split_by_val(t->ch[1], val);
            pull(t);
            return {t, ohs};
```

```
        } else {
            tie(ohs, t->ch[0]) = split_by_val(t->ch[0], val);
            pull(t);
            return {ohs, t};
        }
    }

    pair<Tp, Tp> split_by_rank(Tp t, int rank) {
        if (!t) {
            return {t, t};
        }
        push(t);
        Tp ohs;
        if (rank <= t->ch[0]->info.siz) {
            tie(ohs, t->ch[0]) = split_by_rank(t->ch[0], rank);
            pull(t);
            return {ohs, t};
        } else if (rank > t->ch[0]->info.siz + 1) {
            tie(t->ch[1], ohs) = split_by_rank(t->ch[1], rank - 1 - t->ch[0]-
>info.siz);
            pull(t);
            return {t, ohs};
        } else {
            ohs = t->ch[0];
            t->ch[0] = 0;
            pull(t);
            return {ohs, t};
        }
    }

    Tp merge(Tp u, Tp v) {
        if (!u | !v) return u.x | v.x;
        if (u->key < v->key) {
            push(u);
            u->ch[1] = merge(u->ch[1], v);
            pull(u);
            return u;
        } else {
            push(v);
            v->ch[0] = merge(u, v->ch[0]);
            pull(v);
            return v;
        }
    }

    void rangeReverse(Tp &t, int x, int y) {
        // debug(x, y);
        auto [tmp, r] = split_by_rank(t, y);
        auto [l, m] = split_by_rank(tmp, x);
        m->rev ^= 1;
        m->info.reve();
        t = merge(l, merge(m, r));
    }

    void rangeApply(Tp &t, int x, int y, const Tag &tag) {
        auto [tmp, r] = split_by_rank(t, y);
```

```
        auto [l, m] = split_by_rank(tmp, x);
        apply(m, tag);
        t = merge(l, merge(m, r));
    }

    Tp build(int l, int r) {
        if (r - l == 1) {
            Tp t = __new();
            t->info.init(l);
            return t;
        }
        int m = l + r >> 1;
        return merge(build(l, m), build(m, r));
    }


    void insert(Tp &t, Tp v) {
        if (!t) {
            t = v;
            return;
        }
        if (t->key < v->key) {
            tie(v->ch[0], v->ch[1]) = split_by_val(t, v->info.val);
            t = v;
            pull(t);
            return;
        }
        // t->info.siz += 1;
        insert(t->ch[v->info.val > t->info.val ||
            (t->info.val == v->info.val && int(rng()) >= 0)], v);
        pull(t);
    }

    void erase(Tp &t, T v) {
        if (t->info.val == v) {
            t = merge(t->ch[0], t->ch[1]);
            return;
        } else {
            // t->info.siz -= 1;
            erase(t->ch[v > t->info.val], v);
            pull(t);
        }
    }

    int less_to_val(Tp t, Info val) {
        int less_siz = 0;
        while (t) {
            if (t->info.val >= val.val) {
                t = t->ch[0];
            } else {
                less_siz += t->ch[0]->info.siz + 1;
                t = t->ch[1];
            }
        }
        return less_siz;
    }
```

```
    Tp rank(Tp t, int rank) {
        while (true) {
            if (t->ch[0]->info.siz >= rank) {
                t = t->ch[0];
            } else if (t->ch[0]->info.siz + 1 < rank) {
                rank -= t->ch[0]->info.siz + 1;
                t = t->ch[1];
            } else
                break;
        }
        return t;
    }
    Tp prev_to_val(Tp t, Info val) {
        Tp p;
        while (t) {
            if (t->info.val < val.val) {
                p = t;
                t = t->ch[1];
            } else {
                t = t->ch[0];
            }
        }
        return p;
    }
    Tp next_to_val(Tp t, Info val) {
        Tp p;
        while (t) {
            if (t->info.val <= val.val) {
                t = t->ch[1];
            } else {
                p = t;
                t = t->ch[0];
            }
        }
        return p;
    }
    void dfs(Tp t, int dep = 0) {
        if (!t) {
            return;
        }
        push(t);
        dfs(t->ch[0], dep + 1);
        cout << t->info.val << ' ';
        // for (int i = 0; i < dep; i += 1) cerr << '\t';
        // cerr << t->info << ' ' << t->key << ' ' << t->rev << '\n';
        dfs(t->ch[1], dep + 1);
    }
};

struct Tag {
    constexpr operator bool() {
        return false;
    }
    void apply(const Tag &t) {}
};
struct Info {
```

```
    using T = int;
    int val, siz;
    void reve() {}
    void up(const Info &lhs, const Info &rhs) {
        siz = lhs.siz + 1 + rhs.siz;
    }
    void init(int val) {
        this->val = val;
        siz = 1;
    }
    void apply(const Tag &t) {}
    friend ostream &operator<<(ostream &cout, Info rhs) {
        return cout << "Info: " << rhs.val << ' ' << rhs.siz;
    }
};


using treap = FHQ_treap<Info, Tag>;
using Tp = treap::Tp;
treap T;
```

splay

```
constexpr int max_size = 262144000;
uint8_t buf[max_size];
uint8_t *head = buf;

using u32 = uint32_t;

template <class T>
struct u32_p {
    u32 x;
    u32_p(u32 x = 0) : x(x) {}
    T *operator->() {
        return (T *)(buf + x);
    }
    operator bool() {
        return x;
    }
    operator u32() {
        return x;
    }
    bool operator==(u32_p rhs) const {
        return x == rhs.x;
    }
    static u32_p __new() {
        // assert(x < max_size);
        return (head += sizeof(T)) - buf;
    }
};

template<class Info, class Tag>
struct Balance_Tree {
    struct Tree;
```

```
    using Tp = u32_p<Tree>;

    struct Tree {
        Tp ch[2], p;
        Info info;
        bool rev;
        Tag tag;
    };

    // build operator
    Balance_Tree() {
        Tp()->info.Null();
    }
    Tp __new () {
        return Tp::__new();
    }

    Tp build (int l, int r) {
        if (l > r) return 0;
        int m = l + r >> 1;
        Tp p = __new();
        p->ch[0] = build(l, m - 1);
        if (p->ch[0]) p->ch[0]->p = p;
        {
            // fun
        }
        p->ch[1] = build(m + 1, r);
        if (p->ch[1]) p->ch[1]->p = p;
        pull(p);
        return p;
    }
    template<typename F>
    Tp build (int l, int r, F fun) {
        if (l > r) return 0;
        int m = l + r >> 1;
        Tp p = __new();
        p->ch[0] = build(l, m - 1, fun);
        if (p->ch[0]) p->ch[0]->p = p;
        fun(p, m);
        p->ch[1] = build(m + 1, r, fun);
        if (p->ch[1]) p->ch[1]->p = p;
        pull(p);
        return p;
    }
    // build operator

    // basic operator
    bool pos(Tp t) {
        return t->p->ch[1] == t;
    }

    void apply(Tp t, const Tag &v) {
        if (t) {
            t->info.apply(v);
            t->tag.apply(v);
        }
```

```
    }

    void push(Tp t) {
        if (t->rev) {
            t->ch[0]->rev ^= 1;
            t->ch[1]->rev ^= 1;
            swap(t->ch[0], t->ch[1]);
            t->rev = 0;
        }
        if (t->tag) {
            apply(t->ch[0], t->tag);
            apply(t->ch[1], t->tag);
            t->tag = Tag();
        }
    }

    void pull(Tp t) {
        t->info.up(t->ch[0]->info, t->ch[1]->info);
    }

    void rotate(Tp t) {
        Tp q = t->p;
        int x = !pos(t);
        q->ch[!x] = t->ch[x];
        if (t->ch[x]) t->ch[x]->p = q;
        t->p = q->p;
        if (q->p) q->p->ch[pos(q)] = t;
        t->ch[x] = q;
        q->p = t;
        pull(q);
    }

    void pushall(Tp t) {
        if (t->p) pushall(t->p);
        push(t);
    }

    void splay(Tp t, Tp top = 0) {
        pushall(t);
        while (t->p != top) {
            if (t->p->p != top)
                rotate(pos(t) ^ pos(t->p) ? t : t->p);
            rotate(t);
        }
        pull(t);
    }
    // basic operator

    // shrink operator
    Tp rank(Tp &t, int k) {
        int mid = k;
        while (true) {
            push(t);
            if (k > t->ch[0]->info.siz + t->info.rep_cnt) {
                k -= t->ch[0]->info.siz + t->info.rep_cnt;
                t = t->ch[1];
```

```
            } else if (k <= t->ch[0]->info.siz) {
                t = t->ch[0];
            } else break;
        }
        splay(t);
        return t;
    }

    template<bool isRight>
    void split_by_range(Tp &t, int k) { // split range, but not really split
        rank(t, k);
        if constexpr(!isRight) {
            if (k > t->info.l) {
                Tp l = __new();
                (l->ch[0] = t->ch[0])->p = l;
                (l->p = t)->ch[0] = l;
                l->info.init(t->info.l, k - 1, t->info);
                t->info.init(k, t->info.r, t->info);
                pull(l), pull(t);
            }
        } else {
            if (k < t->info.r) {
                Tp r = __new();
                (r->ch[1] = t->ch[1])->p = r;
                (r->p = t)->ch[1] = r;
                r->info.init(k + 1, t->info.r, t->info);
                t->info.init(t->info.l, k, t->info);
                pull(r), pull(t);
            }
        }
    }

    Tp shrink_by_split_range(Tp &t, int l, int r) {
        if (r == t->info.siz && l == 1) {
            return t;
        } else if (r == t->info.siz) {
            split_by_range<1>(t, l - 1);
            return t->ch[1];
        } else if (l == 1) {
            split_by_range<0>(t, r + 1);
            return t->ch[0];
        } else {
            split_by_range<1>(t, l - 1);
            Tp lhs = t;
            split_by_range<0>(t, r + 1);
            splay(lhs, t);
            return lhs->ch[1];
        }
    }

    Tp shrink(Tp &t, int l, int r) {
        if (r == t->info.siz && l == 1) {
            return t;
        } else if (r == t->info.siz) {
            rank(t, l - 1);
            return t->ch[1];
```

```
        } else if (l == 1) {
            rank(t, r + 1);
            return t->ch[0];
        } else {
            Tp lhs = rank(t, l - 1);
            rank(t, r + 1);
            splay(lhs, t);
            return lhs->ch[1];
        }
    }

    void pullall(Tp t) {
        for (t = t->p; t; t = t->p)
            pull(t);
    }
    // shrink operator

    // split and merge
    std::pair<Tp, Tp> split_by_val(Tp t, int x) {
        if (!t) {
            return {t, t};
        }
        Tp v = 0;
        Tp j = t;
        for (Tp i = t; i; ) {
            push(i);
            j = i;
            if (i->info >= x) {
                v = i;
                i = i->ch[0];
            } else {
                i = i->ch[1];
            }
        }

        splay(j);
        if (!v) {
            return {j, 0};
        }

        splay(v);

        Tp u = v->ch[0];
        if (u) {
            v->ch[0] = u->p = 0;
            pull(v);
        }
        return {u, v};
    }

    std::pair<Tp, Tp> split_by_rank(Tp t, int x) {
        if (t->info.siz < x) {
            return {t, 0};
        }

        rank(t, x);
```

```
    Tp u = t->ch[0];
    if (u) {
        t->ch[0] = u->p = 0;
        pull(t);
    }
    return {u, t};
}

Tp merge(Tp l, Tp r) {
    if (l.x * r.x == 0) {
        return l.x | r.x;
    }
    Tp i = l;
    push(i);
    for (; i->ch[1]; i = i->ch[1], push(i));
    splay(i);
    i->ch[1] = r;
    r->p = i;
    pull(i);
    return i;
}
// split and merge

// set operator
void insert(Tp &t, Tp x) {
    Tp p = 0;

    while (t && t->info.x != x->info.x) {
        push(t);
        p = t;
        t = t->ch[x->info.x > t->info.x];
    }

    if (!t) {
        t = x;
        t->p = p;
        if (p) p->ch[t->info.x > p->info.x] = t;
    } else {
        t->info.apply(x->info);
    }
    splay(t);
}

void find(Tp &t, const Info &rhs) {
    // if (!t) {
    //     return;
    // }
    while (t->info.x != rhs.x && t->ch[rhs.x > t->info.x]) {
        t = t->ch[rhs.x > t->info.x];
    }
    splay(t);
}

Tp prev_by_val(Tp &t, const Info &rhs) {
    Tp p;
```

```
        while (t) {
            if (t->info.x >= rhs.x) {
                t = t->ch[0];
            } else {
                p = t;
                t = t->ch[1];
            }
        }
        splay(t = p);
        return p;
    }

    Tp next_by_val(Tp &t, const Info &rhs) {
        Tp p;
        while (t) {
            if (t->info.x <= rhs.x) {
                t = t->ch[1];
            } else {
                p = t;
                t = t->ch[0];
            }
        }
        splay(t = p);
        return p;
    }

    void erase(Tp &t, const Info &rhs) {
        find(t, rhs);
        if (t->info == rhs && t->info.erase()) {
            Tp lhs = t->ch[0], rhs = t->ch[1];
            lhs->p = 0, rhs->p = 0;
            t = merge(lhs, rhs);
        }
        splay(t);
    }
    // set operator

    void dfs(Tp t, int dep = 0) {
        if (!t) {
            return;
        }
        push(t);
        dfs(t->ch[0], dep + 1);
        for (int i = 0; i < dep; i += 1) cerr << '\t';
        std::cerr << t->info << "\n";
        dfs(t->ch[1], dep + 1);
    }

};

struct Tag {
    int set = 0;
    void apply(const Tag &t) {
        set = t.set;
    }
    operator bool() {
```

```
        return set;
    }
};

struct Info {
    int x = 1, rep_cnt = 1, siz = 1;
    int l = 0, r = 0;
    int sum = 0;
    void up(const Info &lhs, const Info &rhs) {
        siz = lhs.siz + rep_cnt + rhs.siz;
        sum = lhs.sum + x * rep_cnt + rhs.sum;
    }
    void apply(const Tag &t) {
        x = t.set - 1;
        sum = siz * x;
    }
    void apply(const Info &t) {}
    friend ostream &operator<<(ostream &cout, Info rhs) {
        return cout << rhs.x << ' ' << rhs.rep_cnt << ' ' << rhs.siz << ' ' <<
rhs.l << ' ' << rhs.r << ' ' << rhs.sum;
    }
    void init(int L, int R, Info from) {
        l = L, r = R; rep_cnt = r - l + 1; x = from.x;
    }
    void Null() {}
};

using BT = Balance_Tree<Info, Tag>;
using Tp = BT::Tp;
BT tree;
```

treap

```
constexpr int max_size = 262144000;
uint8_t buf[max_size];
uint8_t *head = buf;

using u32 = uint32_t;

template <class T>
struct u32_p {
    u32 x;
    u32_p(u32 x = 0) : x(x) {}
    T *operator->() {
        return (T *)(buf + x);
    }
    operator bool() {
        return x;
    }
    operator u32() {
        return x;
    }
    bool operator==(u32_p rhs) const {
        return x == rhs.x;
    }
```

```
    static u32_p __new() {
        // assert(x < max_size);
        return (head += sizeof(T)) - buf;
    }
};


/**
 * FHQ_treap 卡常:
 * 1.递归改非递归       x
 * 2.insert split优化   o
 * 3.build 优化         o
 */

__gnu_cxx::sfmt19937
rng(std::chrono::steady_clock::now().time_since_epoch().count());

template<typename Info, typename Tag>
struct FHQ_treap {
    struct Node;
    using Tp = u32_p<Node>;

    using T = typename Info::T;
    struct Node {
        Tp ch[2];
        Info info;
        int key;
        Tag tag;
        bool rev;
    };

    Tp __new() {
        Tp t = Tp::__new();
        t->key = rng();
        return t;
    }

    void apply(Tp t, const Tag &tag) {
        if (t) {
            t->info.apply(tag);
            t->tag.apply(tag);
        }
    }

    void push(Tp t) {
        if (t->rev) {
            swap(t->ch[0], t->ch[1]);
            t->ch[0]->rev ^= 1;
            t->ch[0]->info.reve();
            t->ch[1]->rev ^= 1;
            t->ch[1]->info.reve();
            t->rev = 0;
        }
        if (t->tag) {
            apply(t->ch[0], t->tag);
            apply(t->ch[1], t->tag);
```

```
            t->tag = Tag();
        }
    }

    void pull(Tp t) {
        t->info.up(t->ch[0]->info, t->ch[1]->info);
    }

    pair<Tp, Tp> split_by_val(Tp t, T val) {
        if (!t) {
            return {t, t};
        }
        // push(t);
        Tp ohs;
        if (t->info.val < val) {
            tie(t->ch[1], ohs) = split_by_val(t->ch[1], val);
            pull(t);
            return {t, ohs};
        } else {
            tie(ohs, t->ch[0]) = split_by_val(t->ch[0], val);
            pull(t);
            return {ohs, t};
        }
    }

    pair<Tp, Tp> split_by_rank(Tp t, int rank) {
        if (!t) {
            return {t, t};
        }
        push(t);
        Tp ohs;
        if (rank <= t->ch[0]->info.siz) {
            tie(ohs, t->ch[0]) = split_by_rank(t->ch[0], rank);
            pull(t);
            return {ohs, t};
        } else if (rank > t->ch[0]->info.siz + 1) {
            tie(t->ch[1], ohs) = split_by_rank(t->ch[1], rank - 1 - t->ch[0]-
>info.siz);
            pull(t);
            return {t, ohs};
        } else {
            ohs = t->ch[0];
            t->ch[0] = 0;
            pull(t);
            return {ohs, t};
        }
    }

    Tp merge(Tp u, Tp v) {
        if (!u | !v) return u.x | v.x;
        if (u->key < v->key) {
            push(u);
            u->ch[1] = merge(u->ch[1], v);
            pull(u);
            return u;
        } else {
```

```
        push(v);
        v->ch[0] = merge(u, v->ch[0]);
        pull(v);
        return v;
    }
}

void rangeReverse(Tp &t, int x, int y) {
    // debug(x, y);
    auto [tmp, r] = split_by_rank(t, y);
    auto [l, m] = split_by_rank(tmp, x);
    m->rev ^= 1;
    m->info.reve();
    t = merge(l, merge(m, r));
}

void rangeApply(Tp &t, int x, int y, const Tag &tag) {
    auto [tmp, r] = split_by_rank(t, y);
    auto [l, m] = split_by_rank(tmp, x);
    apply(m, tag);
    t = merge(l, merge(m, r));
}

Tp build(int l, int r) {
    if (r - l == 1) {
        Tp t = __new();
        t->info.init(l);
        return t;
    }
    int m = l + r >> 1;
    return merge(build(l, m), build(m, r));
}


void insert(Tp &t, Tp v) {
    if (!t) {
        t = v;
        return;
    }
    if (t->key < v->key) {
        tie(v->ch[0], v->ch[1]) = split_by_val(t, v->info.val);
        t = v;
        pull(t);
        return;
    }
    // t->info.siz += 1;
    insert(t->ch[v->info.val > t->info.val ||
        (t->info.val == v->info.val && int(rng()) >= 0)], v);
    pull(t);
}

void erase(Tp &t, T v) {
    if (t->info.val == v) {
        t = merge(t->ch[0], t->ch[1]);
        return;
    } else {
```

```
            // t->info.siz -= 1;
            erase(t->ch[v > t->info.val], v);
            pull(t);
        }
    }

    int less_to_val(Tp t, Info val) {
        int less_siz = 0;
        while (t) {
            if (t->info.val >= val.val) {
                t = t->ch[0];
            } else {
                less_siz += t->ch[0]->info.siz + 1;
                t = t->ch[1];
            }
        }
        return less_siz;
    }
    Tp rank(Tp t, int rank) {
        while (true) {
            if (t->ch[0]->info.siz >= rank) {
                t = t->ch[0];
            } else if (t->ch[0]->info.siz + 1 < rank) {
                rank -= t->ch[0]->info.siz + 1;
                t = t->ch[1];
            } else
                break;
        }
        return t;
    }
    Tp prev_to_val(Tp t, Info val) {
        Tp p;
        while (t) {
            if (t->info.val < val.val) {
                p = t;
                t = t->ch[1];
            } else {
                t = t->ch[0];
            }
        }
        return p;
    }
    Tp next_to_val(Tp t, Info val) {
        Tp p;
        while (t) {
            if (t->info.val <= val.val) {
                t = t->ch[1];
            } else {
                p = t;
                t = t->ch[0];
            }
        }
        return p;
    }
    void dfs(Tp t, int dep = 0) {
        if (!t) {
```

```
            return;
        }
        push(t);
        dfs(t->ch[0], dep + 1);
        cout << t->info.val << ' ';
        // for (int i = 0; i < dep; i += 1) cerr << '\t';
        // cerr << t->info << ' ' << t->key << ' ' << t->rev << '\n';
        dfs(t->ch[1], dep + 1);
    }
};

struct Tag {
    constexpr operator bool() {
        return false;
    }
    void apply(const Tag &t) {}
};
struct Info {
    using T = int;
    int val, siz;
    void reve() {}
    void up(const Info &lhs, const Info &rhs) {
        siz = lhs.siz + 1 + rhs.siz;
    }
    void init(int val) {
        this->val = val;
        siz = 1;
    }
    void apply(const Tag &t) {}
    friend ostream &operator<<(ostream &cout, Info rhs) {
        return cout << "Info: " << rhs.val << ' ' << rhs.siz;
    }
};


using treap = FHQ_treap<Info, Tag>;
using Tp = treap::Tp;
treap T;
```

可持久化文艺平衡树

```
__gnu_cxx::sfmt19937
rng(std::chrono::steady_clock::now().time_since_epoch().count());

u32 stk[200];

template<typename Info, typename Tag>
struct PersistentBalanceTree {
    struct Node;
    using Tp = u32_p<Node>;

    using T = Info::T;
    struct Node {
        Tp ch[2];
        Info info;
```

```
    int key;
    bool rev;
    Tag tag;
};

Tp __new() {
    Tp t = Tp::__new();
    t->key = rng();
    return t;
}

Tp __new(Tp t) {
    if (!t) return t;
    Tp p = Tp::__new();
    p->ch[0] = t->ch[0];
    p->ch[1] = t->ch[1];
    p->info = t->info;
    p->key = t->key;
    p->rev = t->rev;
    p->tag = t->tag;
    return p;
}

void apply(Tp t, const Tag &tag) {
    if (t) {
        t->info.apply(tag);
        t->tag.apply(tag);
    }
}

void push(Tp t) {
    if (t->rev || t->tag) {
        t->ch[0] = __new(t->ch[0]);
        t->ch[1] = __new(t->ch[1]);
        if (t->rev) {
            swap(t->ch[0], t->ch[1]);
            t->ch[0]->rev ^= 1;
            t->ch[0]->info.reve();
            t->ch[1]->rev ^= 1;
            t->ch[1]->info.reve();
            t->rev = 0;
        }
        if (t->tag) {
            apply(t->ch[0], t->tag);
            apply(t->ch[1], t->tag);
            t->tag = Tag();
        }
    }
}

void pull(Tp t) {
    t->info.up(t->ch[0]->info, t->ch[1]->info);
}

void rangeReverse(Tp &t, int x, int y) {
    // debug(x, y);
```

```
        auto [tmp, r] = split_by_rank(t, y);
        auto [l, m] = split_by_rank(tmp, x);
        m->rev ^= 1;
        m->info.reve();
        t = merge(l, merge(m, r));
    }

    void rangeApply(Tp &t, int x, int y, const Tag &tag) {
        auto [tmp, r] = split_by_rank(t, y);
        auto [l, m] = split_by_rank(tmp, x);
        apply(m, tag);
        t = merge(l, merge(m, r));
    }

    Info rangeQuery(Tp &t, int x, int y) {
        // debug(x, y);
        auto [tmp, r] = split_by_rank(t, y);
        auto [l, m] = split_by_rank(tmp, x);
        Info ans = m->info;
        t = merge(l, merge(m, r));
        return ans;
    }
//  split and merge
    pair<Tp, Tp> split_by_val(Tp &t, T val) {
        if (!t) {
            return {0, 0};
        }
        t = __new(t);
        push(t);
        Tp ohs;
        if (t->info.val < val) {
            tie(t->ch[1], ohs) = split_by_val(t->ch[1], val);
            pull(t);
            return {t, ohs};
        } else {
            tie(ohs, t->ch[0]) = split_by_val(t->ch[0], val);
            pull(t);
            return {ohs, t};
        }
    }

    pair<Tp, Tp> split_by_rank(Tp t, int rank) {
        if (!t) {
            return {t, t};
        }
        push(t);
        t = __new(t);
        Tp ohs;
        if (rank <= t->ch[0]->info.siz) {
            tie(ohs, t->ch[0]) = split_by_rank(t->ch[0], rank);
            pull(t);
            return {ohs, t};
        } else {
            tie(t->ch[1], ohs) = split_by_rank(t->ch[1], rank - 1 - t->ch[0]-
>info.siz);
            pull(t);
```

```
            return {t, ohs};
        }
    }

    template<bool isNew = false>
    Tp merge(Tp u, Tp v) {
        if (!u | !v) return u.x | v.x;
        if (u->key < v->key) {
            push(u);
            if (isNew) {
                u = __new(u);
            }
            u->ch[1] = merge<isNew>(u->ch[1], v);
            pull(u);
            return u;
        } else {
            push(v);
            if (isNew) {
                v = __new(v);
            }
            v->ch[0] = merge<isNew>(u, v->ch[0]);
            pull(v);
            return v;
        }
    }
//  split and merge


//  set operator

    // void insert_by_rank(Tp &t, int rank, Tp v) {
    //     auto [l, r] = split_by_rank(t, rank);
    //     t = merge(l, merge(v, r));
    // }

    void insert_by_rank(Tp &t, int rank, Tp v) {
        if (!t) {
            t = v;
            return;
        }
        push(t);
        t = __new(t);
        if (v->key < t->key) {
            tie(v->ch[0], v->ch[1]) = split_by_rank(t, rank);
            t = v;
            pull(t);
            return;
        }
        // debug(rank, t->ch[0]->info.siz);
        if (rank <= t->ch[0]->info.siz) {
            insert_by_rank(t->ch[0], rank, v);
        } else {
            insert_by_rank(t->ch[1], rank - 1 - t->ch[0]->info.siz, v);
        }
        pull(t);
    }
```

```
    // void erase_by_rank(Tp &t, int rank) {
    //     auto [tmp, r] = split_by_rank(t, rank);
    //     auto [l, m] = split_by_rank(tmp, rank - 1);
    //     t = merge(l, r);
    // }

    void erase_by_rank(Tp &t, int rank) {
        if (!t) return;
        push(t);
        t = __new(t);
        if (rank <= t->ch[0]->info.siz) {
            erase_by_rank(t->ch[0], rank);
            pull(t);
        } else if (rank > t->ch[0]->info.siz + 1) {
            erase_by_rank(t->ch[1], rank - 1 - t->ch[0]->info.siz);
            pull(t);
        } else {
            t = merge<true>(t->ch[0], t->ch[1]);
        }
    }

    void insert_by_val(Tp &t, Tp v) {
        t = __new(t);
        if (!t) {
            t = v;
            return;
        }
        if (t->key < v->key) {
            // push(t);
            tie(v->ch[0], v->ch[1]) = split_by_val(t, v->info.val);
            t = v;
            pull(t);
            return;
        }
        // t->info.siz += 1;
        insert_by_val(t->ch[v->info.val > t->info.val || (t->info.val == v-
>info.val && int(rng()) >= 0)], v);
        pull(t);
    }

    void erase_by_val(Tp &t, T v) {
        if (!t) return;
        t = __new(t);
        if (t->info.val == v) {
            t = merge(t->ch[0], t->ch[1]);
            return;
        } else {
            // t->info.siz -= 1;
            erase_by_val(t->ch[v > t->info.val], v);
            pull(t);
        }
    }
// not back
    void __insert_by_val(Tp &t, Tp v) {
        int Top = -1;
```

```
        Tp *p = &t;
        while (*p && v->key <= (*p)->key) {
            *p = __new(*p);
            stk[++ Top] = *p;
            p = &((*p)->ch[v->info.val > (*p)->info.val || ((*p)->info.val == v->info.val && int(rng()) >= 0)]);
        }
        if (*p) {
            tie(v->ch[0], v->ch[1]) = split_by_val(*p, v->info.val);
            pull(v);
        }
        *p = v;
        if (Top != -1) t = stk[0];
        while (Top != -1) {
            pull(stk[Top --]);
        }
    }

    void __erase_by_val(Tp &t, T v) {
        int Top = -1;
        Tp *p = &t;
        while (*p && (*p)->info.val != v) {
            *p = __new(*p);
            stk[++ Top] = *p;
            p = &((*p)->ch[v > (*p)->info.val]);
        }
        if (*p) {
            *p = merge((*p)->ch[0], (*p)->ch[1]);
        }
        if (Top != -1) t = stk[0];
        while (Top != -1) {
            pull(stk[Top --]);
        }
    }
// not back
    int less_to_val(Tp t, T val) {
        int less_siz = 0;
        while (t) {
            if (t->info.val >= val) {
                t = t->ch[0];
            } else {
                less_siz += t->ch[0]->info.siz + 1;
                t = t->ch[1];
            }
        }
        return less_siz;
    }
    Tp rank(Tp t, int rank) {
        while (true) {
            if (t->ch[0]->info.siz >= rank) {
                t = t->ch[0];
            } else if (t->ch[0]->info.siz + 1 < rank) {
                rank -= t->ch[0]->info.siz + 1;
                t = t->ch[1];
            } else
                break;
```

```
        }
        return t;
    }
    Tp prev_to_val(Tp t, T val) {
        Tp p;
        while (t) {
            if (t->info.val < val) {
                p = t;
                t = t->ch[1];
            } else {
                t = t->ch[0];
            }
        }
        return p;
    }
    Tp next_to_val(Tp t, T val) {
        Tp p;
        while (t) {
            if (t->info.val <= val) {
                t = t->ch[1];
            } else {
                p = t;
                t = t->ch[0];
            }
        }
        return p;
    }
    void dfs(Tp t, int dep = 0) {
        if (!t) {
            return;
        }
        dfs(t->ch[0], dep + 1);
        for (int i = 0; i < dep; i += 1) cerr << '\t';
        cerr << t->info << ' ' << t->key << ' ' << t->rev << '\n';
        dfs(t->ch[1], dep + 1);
    }
};

struct Tag {
    constexpr operator bool() {
        return false;
    }
    void apply(const Tag &t) {}
};
struct Info {
    using T = int;
    int val, siz;
    i64 sum;
    void reve() {}
    void up(const Info &lhs, const Info &rhs) {
        siz = lhs.siz + 1 + rhs.siz;
        sum = lhs.sum + val + rhs.sum;
    }
    void init(int val) {
        this->val = val;
        this->sum = val;
```

```
        siz = 1;
    }
    void apply(const Tag &t) {}
    friend ostream &operator<<(ostream &cout, Info rhs) {
        return cout << "Info: " << rhs.val << ' ' << rhs.sum << ' ' << rhs.siz;
    }
};

using treap = PersistentBalanceTree<Info, Tag>;
using Tp = treap::Tp;

treap T;
```

jls splay

```
struct Tree {
    int add = 0;
    int val = 0;
    int id = 0;
    u32_p<Tree> ch[2], p;
};

using Tp = u32_p<Tree>;

Tp __new() {
    return Tp::__new();
}

int pos(Tp t) {
    return t->p->ch[1] == t;
}

void add(Tp t, int v) {
    t->val += v;
    t->add += v;
}

void push(Tp t) {
    if (t->ch[0]) {
        add(t->ch[0], t->add);
    }
    if (t->ch[1]) {
        add(t->ch[1], t->add);
    }
    t->add = 0;
}

void rotate(Tp t) {
    Tp q = t->p;
    int x = !pos(t);
    q->ch[!x] = t->ch[x];
    if (t->ch[x]) t->ch[x]->p = q;
    t->p = q->p;
    if (q->p) q->p->ch[pos(q)] = t;
    t->ch[x] = q;
```

```
    q->p = t;
}

void splay(Tp t) {
    std::vector<Tp > s;
    for (Tp i = t; i->p; i = i->p) s.push_back(i->p);
    while (!s.empty()) {
        push(s.back());
        s.pop_back();
    }
    push(t);
    while (t->p) {
        if (t->p->p) {
            if (pos(t) == pos(t->p)) rotate(t->p);
            else rotate(t);
        }
        rotate(t);
    }
}

void insert(Tp &t, Tp x, Tp p = 0) {
    if (!t) {
        t = x;
        x->p = p;
        return;
    }

    push(t);
    if (x->val < t->val) {
        insert(t->ch[0], x, t);
    } else {
        insert(t->ch[1], x, t);
    }
}

void dfs(Tp t) {
    if (!t) {
        return;
    }
    push(t);
    dfs(t->ch[0]);
    std::cerr << t->val << " ";
    dfs(t->ch[1]);
}

std::pair<Tp , Tp > split(Tp t, int x) {
    if (!t) {
        return {t, t};
    }
    Tp v = 0;
    Tp j = t;
    for (Tp i = t; i; ) {
        push(i);
        j = i;
        if (i->val >= x) {
            v = i;
```

```
            i = i->ch[0];
        } else {
            i = i->ch[1];
        }
    }

    splay(j);
    if (!v) {
        return {j, 0};
    }

    splay(v);

    Tp u = v->ch[0];
    if (u) {
        v->ch[0] = u->p = 0;
    }
    return {u, v};
}

Tp merge(Tp l, Tp r) {
    if (!l) {
        return r;
    }
    if (!r) {
        return l;
    }
    Tp i = l;
    while (i->ch[1]) {
        i = i->ch[1];
    }
    splay(i);
    i->ch[1] = r;
    r->p = i;
    return i;
}
```

线段树套平衡树

```
constexpr int max_size = 262144000;
uint8_t buf[max_size];
uint8_t *head = buf;

using u32 = uint32_t;

template <class T>
struct u32_p {
    u32 x;
    u32_p(u32 x = 0) : x(x) {}
    T *operator->() {
        return (T *)(buf + x);
    }
    operator bool() {
        return x;
    }
```

```
    operator u32() {
        return x;
    }
    bool operator==(u32_p rhs) const {
        return x == rhs.x;
    }
    static u32_p __new() {
        // assert(x < max_size);
        return (head += sizeof(T)) - buf;
    }
};

/**
 * FHQ_treap set卡常:
 * 1.递归改非递归       x
 * 2.insert split优化   o
 */

__gnu_cxx::sfmt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

template<typename Info>
struct FHQ_treap {
    struct Node;
    using Tp = u32_p<Node>;
    struct Node {
        Tp ch[2];
        Info val;
        int siz, key;
    };

    Tp root;

    void pull(Tp t) {
        t->siz = t->ch[0]->siz + 1 + t->ch[1]->siz;
    }
    // by val
    pair<Tp, Tp> split(Tp t, Info val) {
        if (!t) {
            return {t, t};
        }
        Tp ohs;
        if (t->val < val) {
            tie(t->ch[1], ohs) = split(t->ch[1], val);
            pull(t);
            return {t, ohs};
        } else {
            tie(ohs, t->ch[0]) = split(t->ch[0], val);
            pull(t);
            return {ohs, t};
        }
    }

    Tp merge(Tp u, Tp v) {
        if (!u | !v) return u.x | v.x;
        if (u->key < v->key) {
            u->ch[1] = merge(u->ch[1], v);
```

```
            pull(u);
            return u;
        } else {
            v->ch[0] = merge(u, v->ch[0]);
            pull(v);
            return v;
        }
    }

// set operator
    void insert(Tp &t, Tp v) {
        if (!t) {
            t = v;
            // ps;
            return;
        }
        if (t->key < v->key) {
            tie(v->ch[0], v->ch[1]) = split(t, v->val);
            t = v;
            pull(t);
            return;
        }
        // t->siz += 1;
        insert(t->ch[v->val > t->val ||
            (t->val == v->val && int(rng()) >= 0)], v);
        pull(t);
    }

    void insert(Info v) {
        Tp t = Tp::__new();
        t->key = rng();
        t->val = v;
        t->siz = 1;
        insert(root, t);
    }

    void erase(Tp &t, Info v) {
        if (t->val == v) {
            t = merge(t->ch[0], t->ch[1]);
            return;
        } else {
            // t->siz -= 1;
            erase(t->ch[v > t->val], v);
            pull(t);
        }
    }

    void erase(Info v) {
        erase(root, v);
    }
    // by val
    int less(Info v) {
        Tp t = root;
        int less_siz = 0;
        while (t) {
            if (t->val >= v) {
```

```
            t = t->ch[0];
        } else {
            less_siz += t->ch[0]->siz + 1;
            t = t->ch[1];
        }
    }
    return less_siz;
}
// from zero
Tp rank(Tp t, int k) {
    k += 1;
    while (true) {
        if (t->ch[0]->siz >= k) {
            t = t->ch[0];
        } else if (t->ch[0]->siz + 1 < k) {
            k -= t->ch[0]->siz + 1;
            t = t->ch[1];
        } else
            break;
    }
    return t;
}
// from zero
Tp operator[] (int k) {
    return rank(root, k);
}
// by val
static constexpr int inf = std::numeric_limits<int>::max();
Info prev(Info v) {
    Tp t = root, p;
    while (t) {
        if (t->val < v) {
            p = t;
            t = t->ch[1];
        } else {
            t = t->ch[0];
        }
    }
    return p ? p->val : -inf;
}
// by val
Info next(Info v) {
    Tp t = root, p;
    while (t) {
        if (t->val <= v) {
            t = t->ch[1];
        } else {
            p = t;
            t = t->ch[0];
        }
    }
    return p ? p->val : inf;
}
void dfs(Tp t, int dep = 0) {
    if (!t) {
        return;
```

```
        }
        dfs(t->ch[0], dep + 1);
        for (int i = 0; i < dep; i += 1) cerr << '\t';
        cerr << t->val << ' ' << t->key << '\n';
        dfs(t->ch[1], dep + 1);
    }
    void dfs() {return dfs(root);}
};

template<typename Value>
struct SegTreap {
    int n;
    vector<Value> val;
    vector<FHQ_treap<Value>> info;
    SegTreap() : n(0) {}
    SegTreap(int n_, Value v_ = Value()) {
        init(n_, v_);
    }
    template<class T>
    SegTreap(vector<T> init_) {
        init(init_);
    }
    void init(int n_, Value v_ = Value()) {
        init(vector(n_, v_));
    }
    template<class T>
    void init(vector<T> init_) {
        n = init_.size();
        val = init_;
        info.assign(4 << __lg(n), {});
        function<void(int, int, int)>
        build = [&](int p, int l, int r) {
            for (int i = l; i < r; i += 1) {
                info[p].insert(val[i]);
            }
            if (r - l == 1) {
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
        };
        build(1, 0, n);
    }
    void modify(int p, int l, int r, int x, const Value &v) {
        info[p].erase(val[x]);
        info[p].insert(v);
        if (r - l == 1) return;
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
    }
    void modify(int p, const Value &v) {
```

```
        if(p >= n) return;
        modify(1, 0, n, p, v);
        val[p] = v;
    }
    int less(int p, int l, int r, int x, int y, const Value &v) {
        if (l >= x && r <= y) {
            return info[p].less(v);
        }
        int m = (l + r) / 2;
        if (m >= y) {
            return less(2 * p, l, m, x, y, v);
        } else if (m <= x) {
            return less(2 * p + 1, m, r, x, y, v);
        } else {
            return less(2 * p, l, m, x, y, v) + less(2 * p + 1, m, r, x, y, v);
        }
    }
    int less(int l, int r, const Value &v) {
        if (l >= r) return 0;
        return less(1, 0, n, l, r, v);
    }

    // from zero
    Value kth (int x, int y, int k) {
        int l = 0, r = 1e8 + 1;
        while (l + 1 != r) {
            int m = l + r >> 1;
            if (less(x, y, m) <= k) l = m;
            else r = m;
        }
        return l;
    }

    Value prev(int p, int l, int r, int x, int y, const Value &v) {
        if (l >= x && r <= y) {
            return info[p].prev(v);
        }
        int m = (l + r) / 2;
        if (m >= y) {
            return prev(2 * p, l, m, x, y, v);
        } else if (m <= x) {
            return prev(2 * p + 1, m, r, x, y, v);
        } else {
            return std::max(prev(2 * p, l, m, x, y, v), prev(2 * p + 1, m, r, x,
y, v));
        }
    }

    Value prev(int x, int y, const Value &v) {
        return prev(1, 0, n, x, y, v);
    }

    Value next(int p, int l, int r, int x, int y, const Value &v) {
        if (l >= x && r <= y) {
            return info[p].next(v);
        }
```

```
        int m = (l + r) / 2;
        if (m >= y) {
            return next(2 * p, l, m, x, y, v);
        } else if (m <= x) {
            return next(2 * p + 1, m, r, x, y, v);
        } else {
            return std::min(next(2 * p, l, m, x, y, v), next(2 * p + 1, m, r, x,
y, v));
        }
    }

    Value next(int x, int y, const Value &v) {
        return next(1, 0, n, x, y, v);
    }

    void show(int p, int l, int r, int x, int y, int dep = 0) {
        if (l >= y || r <= x) return;
        int m = (l + r) >> 1;
        if (r - l > 1)
        show(p * 2, l, m, x, y, dep + 1);
        for (int i = 0; i < dep; i += 1) {
            cerr << '\t';
        }
        cerr << l << ' ' << r << ' '; info[p].show();
        cerr << '\n';
        if (r - l > 1)
        show(p * 2 + 1, m, r, x, y, dep + 1);
    }
    void show(int l, int r) {
        show(1, 0, n, l, r);
    }
};

using Tree = SegTreap<int>;
```

树状数组

标准版

```
template<typename T>
struct Fenwick {
    int n;
    std::vector <T> a;

    Fenwick(int n_ = 0) {
        init(n_);
    }

    void init(int n_) {
        n = n_;
        a.assign(n, T{});
    }

    void add(int x, const T &v) {
```

```
        if (x < 0 || x >= n) return;
        for (int i = x + 1; i <= n; i += i & -i) {
            a[i - 1] = a[i - 1] + v;
        }
    }

    T Query(int x) {
        if (x <= 0) return T{};
        if (x > n) x = n;
        T ans{};
        for (int i = x; i != 0; i -= i & -i) {
            ans = ans + a[i - 1];
        }
        return ans;
    }

    T range_Query(int l, int r) {
        if (l >= r) return 0;
        return Query(r) - Query(l);
    }

    int kth(const T &k) {
        int x = 0;
        T cur{};
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && cur + a[x + i - 1] < k) {
                x += i;
                cur = cur + a[x - 1];
            }
        }
        return x;
    }
};
```

二维树状数组

```
template<typename T>
struct Two_dimensional_Fenwick {
    struct Base_Fenwick {
        int n, m;
        std::vector <std::vector<T>> s;

        Base_Fenwick(int _n = 0, int _m = 0) {
            init(_n, _m);
        }

        void init(int _n, int _m) {
            n = _n, m = _m;
            s.assign(n + 1, std::vector<T>(m + 1, T()));
        }

        void change(int x, int y, const T &v) {
            if (x <= 0 || y <= 0) return;
            if (x > n) x = n;
            if (y > m) y = m;
```

```
        for (int i = x; i <= n; i += i & (-i))
            for (int j = y; j <= m; j += j & (-j))
                s[i][j] += v;
    }

    T Query(int x, int y) {
        if (x <= 0 || y <= 0) return T();
        if (x > n) x = n;
        if (y > m) y = m;
        T ans = 0;
        for (int i = x; i != 0; i -= i & (-i))
            for (int j = y; j != 0; j -= j & (-j))
                ans += s[i][j];
        return ans;
    }
};

int n, m;
Base_Fenwick A, B, C, D;

Two_dimensional_Fenwick(int _n = 0, int _m = 0) {
    init(_n, _m);
}

void init(int _n, int _m) {
    n = _n, m = _m;
    A.init(n, m);
    B.init(n, m);
    C.init(n, m);
    D.init(n, m);
}

void Base_add(int x, int y, int v) {
    A.change(x, y, v);
    B.change(x, y, v * x);
    C.change(x, y, v * y);
    D.change(x, y, v * x * y);
}

T Base_Query(int x, int y) {
    return A.Query(x, y) * (x * y + x + y + 1)
           - B.Query(x, y) * (y + 1)
           - C.Query(x, y) * (x + 1)
           + D.Query(x, y);
}

void add(int x0, int y0, int x1, int y1, int v) {
    Base_add(x0, y0, v);
    Base_add(x0, y1 + 1, -v);
    Base_add(x1 + 1, y0, -v);
    Base_add(x1 + 1, y1 + 1, v);
}

T Query(int x0, int y0, int x1, int y1) {
    return Base_Query(x1, y1) - Base_Query(x0 - 1, y1)
           - Base_Query(x1, y0 - 1) + Base_Query(x0 - 1, y0 - 1);
```

```
    }
};
```

区间加树状数组

```
template<typename T>
struct Range_Fenwick {
    int n;
    Fenwick <T> a, b;

    Range_Fenwick (int _n = 0) {
        init (_n);
    }

    void init (int _n) {
        n = _n;
        a.init(n); b.init(n);
    }

    void range_Change (int l, int r, const T& k) {
        a.add(l, k); a.add(r + 1, -k);
        b.add(l, k * l); b.add(r + 1, -k * (r + 1)) ;
    }

    T range_Query (int l, int r) {
        return (r + 1) * a.Query(r) - l * a.Query(l - 1) - b.range_Query(l, r);
    }

    int kth(const T &k) {
        int x = 0;
        T cur0{}, cur1{};
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && (cur0 + a.a[x + i]) * (x + i + 1) - (cur1 + b.a[x +
i]) < k) {
                x += i;
                cur0 = cur0 + a.a[x];
                cur1 = cur1 + b.a[x];
            }
        }
        return x + 1;
    }
};
```

线段树

单点

```
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) {
```

```
        init(n_, v_);
    }
    template<class T>
    SegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p, l, m, r);
        };
        build(1, 0, n);
    }
    void pull(int p, int l, int m, int r) {
        info[p].update(info[2 * p], info[2 * p + 1], l, m, r);
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p].apply(v, l, r);
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p, l, m, r);
    }
    void modify(int p, const Info &v) {
        if(p >= n) return;
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        if (m >= y) {
            return rangeQuery(2 * p, l, m, x, y);
        } else if (m <= x) {
            return rangeQuery(2 * p + 1, m, r, x, y);
        } else {
```

```
            return Info::merge(rangeQuery(2 * p, l, m, x, y), rangeQuery(2 * p +
1, m, r, x, y), std::max(l, x), m, std::min(r, y));
        }
    }
    Info rangeQuery(int l, int r) {
        if (l >= r) return Info();
        return rangeQuery(1, 0, n, l, r);
    }

    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F pred) {
        return findLast(1, 0, n, l, r, pred);
    }
    void DFS(int p, int l, int r, int x, int y, int dep = 0) {
        if (l >= y || r <= x) return;
        int m = (l + r) >> 1;
        if (r - l > 1)
        DFS(p * 2, l, m, x, y, dep + 1);
        cerr << string(dep, '\t');
        cerr << l << ' ' << r << ' ' << info[p];
        cerr << '\n';
        if (r - l > 1)
```

```
            DFS(p * 2 + 1, m, r, x, y, dep + 1);
    }
    void dfs(int l, int r) {
        DFS(1, 0, n, l, r);
    }
};

struct Info {
    void apply(const Info &rhs, int l, int r) {}
    void update(const Info &lhs, const Info &rhs, int l, int m, int r) {}
    static Info merge(const Info &lhs, const Info &rhs, int l, int m, int r) {
        Info info = Info();
        info.update(lhs, rhs, l, m, r);
        return info;
    }
    friend ostream &operator<<(ostream &cout, Info t) {
        return cout << "Info" << "; ";
    }
};

using Tree = SegmentTree<Info>;
```

区间

```
template<class Info, class Tag>
struct LazySegmentTree {
    int n;
    std::vector<Info> info;
    std::vector<Tag> tag;
    LazySegmentTree() : n(0) {}
    LazySegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    LazySegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(n * 4, Info());
        tag.assign(n * 4, Tag());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p, l, m, r);
        };
```

```
        build(1, 0, n);
    }
    void pull(int p, int l, int m, int r) {
        info[p].update(info[2 * p], info[2 * p + 1], l, m, r);
    }
    void apply(int p, const Tag &v, int l, int r) {
        info[p].apply(v, l, r);
        tag[p].apply(v);
    }
    void push(int p, int l, int m, int r) {
        if (bool(tag[p])) {
            apply(2 * p, tag[p], l, m);
            apply(2 * p + 1, tag[p], m, r);
            tag[p] = Tag();
        }
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        push(p, l, m, r);
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p, l, m, r);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        push(p, l, m, r);
        if (m >= y) {
            return rangeQuery(2 * p, l, m, x, y);
        } else if (m <= x) {
            return rangeQuery(2 * p + 1, m, r, x, y);
        } else {
            return Info::merge(rangeQuery(2 * p, l, m, x, y), rangeQuery(2 * p +
1, m, r, x, y), l, m, r);
        }
    }
    Info rangeQuery(int l, int r) {
        if (l >= r) return Info();
        return rangeQuery(1, 0, n, l, r);
    }
    void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
        if (l >= y || r <= x) {
            return;
        }
```

```
        int m = (l + r) / 2;
        if (l >= x && r <= y) {
            apply(p, v, l, r);
            return;
        }
        push(p, l, m, r);
        rangeApply(2 * p, l, m, x, y, v);
        rangeApply(2 * p + 1, m, r, x, y, v);
        pull(p, l, m, r);
    }
    void rangeApply(int l, int r, const Tag &v) {
        return rangeApply(1, 0, n, l, r, v);
    }
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        push(p, l, m, r);
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        push(p, l, m, r);
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F pred) {
        return findLast(1, 0, n, l, r, pred);
    }
    void DFS(int p, int l, int r, int x, int y, int dep = 0) {
        if (l >= y || r <= x) return;
        int m = (l + r) >> 1;
```

```
        if (r - l > 1)
        DFS(p * 2, l, m, x, y, dep + 1);
        cerr << string(dep, '\t');
        cerr << l << ' ' << r << ' ' << info[p] << tag[p];
        cerr << '\n';
        if (r - l > 1)
            DFS(p * 2 + 1, m, r, x, y, dep + 1);
    }
    void dfs(int l, int r) {
        DFS(1, 0, n, l, r);
    }
};

struct Tag {
    void apply(Tag t) {}
    constexpr operator bool() {
        return true;
    }
    friend ostream &operator<<(ostream &cout, Tag t) {
        return cout << "tag" << ";";
    }
};

struct Info {
    void apply(const Tag &t, int l, int r) {}
    void update(const Info &lhs, const Info &rhs, int l, int m, int r) {}
    static Info merge(const Info &lhs, const Info &rhs, int l, int m, int r) {
        Info info = Info();
        info.update(lhs, rhs, l, m, r);
        return info;
    }
    friend ostream &operator<<(ostream &cout, Info t) {
        return cout << "Info" << "; ";
    }
};

using lazySegmentTree = LazySegmentTree<Info, Tag>;
```

tourist zkw 线段树（精简版）区间最大值

```
struct SegmTree {
  vector<int> T; int n;
  SegmTree(int n) : T(2 * n, (int)-2e9), n(n) {}

  void Update(int pos, int val) {
    for (T[pos += n] = val; pos > 1; pos /= 2)
      T[pos / 2] = max(T[pos], T[pos ^ 1]);
  }

  int Query(int b, int e) {
    int res = -2e9;
    for (b += n, e += n; b < e; b /= 2, e /= 2) {
      if (b % 2) res = max(res, T[b++]);
      if (e % 2) res = max(res, T[--e]);
    }
```

```
    return res;
  }
};
```

动态开点线段树

```
template<typename Info, typename Tag>
struct segment_tree {
    int n;
    struct node;
    using Tp = Base<node>;

    struct node {
        Info info;
        Tag tag;
        Tp ch[2];
    };
    Tp t{0};
    Tp news(i64 l, i64 r) {
        Tp t = Tp::news();
        return t;
    }
    void apply(Tp &t, const Tag &v, i64 l, i64 r) {
        if (!t) {
            t = news(l, r);
        }
        t->info.apply(v, l, r);
        t->tag.apply(v);
    }
    void push(Tp &t, i64 l, i64 m, i64 r) {
        if (!bool(t->tag))
            return;
        apply(t->ch[0], t->tag, l, m);
        apply(t->ch[1], t->tag, m, r);
        t->tag = Tag();
    }
    void pull(Tp &t, i64 l, i64 m, i64 r) {
        t->info.update(t->ch[0]->info, t->ch[1]->info, l, m, r);
    }
    i64 floor, ceil;
    segment_tree(i64 floor, i64 ceil) : floor(floor) , ceil(ceil) {}
    void modify(Tp &t, const Tag &v, i64 l, i64 r, i64 x) {
        if (!t)
            t = news(l, r);
        i64 m = (l + r) >> 1;
        if (r - l == 1) {
            t->info.apply(v, l, r);
            return;
        }
        push(t, l, m, r);
        if (m > x)
            modify(t->ch[0], v, l, m, x);
        else
            modify(t->ch[1], v, m, r, x);
        pull(t, l, m, r);
```

```
    }
    void modify(i64 x, const Tag &v) {
        modify(t, v, floor, ceil, x);
    }
    void rangeApply(Tp &t, const Tag &v, i64 l, i64 r, i64 x, i64 y) {
        if (!t)
            t = news(l, r);
        i64 m = (l + r) >> 1;
        if (x <= l && r <= y) {
            apply(t, v, l, r);
            return;
        }
        push(t, l, m, r);
        if (m > x)
            rangeApply(t->ch[0], v, l, m, x, y);
        if (m < y)
            rangeApply(t->ch[1], v, m, r, x, y);
        pull(t, l, m, r);
    }
    void rangeApply(i64 x, i64 y, const Tag &v) {
        if (x >= y) return;
        rangeApply(t, v, floor, ceil, x, y);
    }
    Info Query(Tp &t, i64 l, i64 r, i64 x) {
        if (!t)
            return Info::merge(l, r);
        i64 m = (l + r) >> 1;
        if (r - l == 1) {
            return t->info;
        }
        push(t, l, m, r);
        if (m > x)
            return Query(t->ch[0], l, m, x);
        else
            return Query(t->ch[1], m, r, x);
    }
    Info Query(i64 x) {
        return Query(t, floor, ceil, x);
    }
    Info rangeQuery(Tp &t, i64 l, i64 r, i64 x, i64 y) {
        if (!t)
            return Info::merge(l, r);
        i64 m = (l + r) >> 1;
        if (x <= l && r <= y) {
            return t->info;
        }
        push(t, l, m, r);
        if (m >= y) {
            return rangeQuery(t->ch[0], l, m, x, y);
        } else if (m <= x) {
            return rangeQuery(t->ch[1], m, r, x, y);
        } else {
            return Info::merge(rangeQuery(t->ch[0], l, m, x, y), rangeQuery(t-
>ch[1], m, r, x, y), l, m, r);
        }
    }
```

```
    Info rangeQuery(i64 x, i64 y) {
        return rangeQuery(t, floor, ceil, x, y);
    }
    void DFS(Tp &t, i64 l, i64 r, i64 x, i64 y, int dep = 0) {
        if (l >= y || r <= x || !t) return;
        i64 m = (l + r) >> 1;
        if (r - l > 1)
        DFS(t->ch[0], l, m, x, y, dep + 1);
        cerr << string(dep, '\t');
        cerr << l << ' ' << r << ' ' << t->info << t->tag;
        cerr << '\n';
        if (r - l > 1)
            DFS(t->ch[1], m, r, x, y, dep + 1);
    }
    void dfs(i64 x, i64 y) {
        DFS(t, floor, ceil, x, y);
    }
};

struct Tag {
    i64 x = 0;
    void apply(const Tag &rhs) {
        x += rhs.x;
    }
    operator bool() {
        return x != 0;
    }
    void clear() {
        x = 0;
    }
    friend ostream &operator<<(ostream &cout, Tag t) {
        return cout << "tag" << ";";
    }
};

struct Info {
    i64 x = 0;
    void apply(const Tag &rhs, i64 l, i64 r) {
        x += (r - l) * rhs.x;
    }
    void update(const Info &lhs, const Info &rhs, i64 l, i64 m, i64 r) {
        x = lhs.x + rhs.x;
    }
    static Info merge(const Info &lhs, const Info &rhs, i64 l, i64 m, i64 r) {
        Info info = Info();
        info.update(lhs, rhs, l, m, r);
        return info;
    }
    static Info merge(i64 l, i64 r) {
        return {0};
    }
    friend ostream &operator<<(ostream &cout, Info t) {
        return cout << "Info" << "; ";
    }
};
```

```
using SegmentTree = segment_tree<Info, Tag>;
```

可持久化线段树

```
template<typename Info>
struct segment_tree {
    int n;
    struct node;
    using Tp = Base<node>;
    struct node {
        Info info;
        Tp ch[2];
    };
    Tp news() {
        Tp t = Tp::news();
        return t;
    }
    segment_tree(): n(0) {}
    segment_tree(int n, Info v = Info()) {
        init(std::vector(n, v));
    }
    template<typename T>
    segment_tree(std::vector<T> _init) {
        init(_init);
    }
    void pull(Tp &t) {
        t->info.update(t->ch[0]->info, t->ch[1]->info);
    }
    template<typename T>
    void init(const std::vector<T> &_init) {
        n = _init.size();
        auto build = [&] (auto &&self, Tp &t, int l, int r) {
            t = news();
            if (r - l == 1) {
                t->info = _init[l];
                return;
            }
            int m = (l + r) / 2;
            self(self, t->ch[0], l, m), self(self, t->ch[1], m, r);
            pull(t);
        };
        build(t, 0, n);
    }
    Tp &modify(Tp &u, const Info &M, int l, int r, int x) {
        Tp v = news();
        if (r - l == 1) {
            v->info = u->info;
            v->info.apply(M);
            return v;
        }
        int m = (l + r) / 2;
        if (m > x) {
            v->ch[1] = u->ch[1];
```

```
            v->ch[0] = modify(u->ch[0], M, l, m, x);
        } else {
            v->ch[0] = u->ch[0];
            v->ch[1] = modify(u->ch[1], M, m, r, x);
        }
        pull(v);
    }
    Tp &modify(Tp &t, int x, const Info &M) {
        modify(t, M, 0, n, x);
    }
    Info range_query(Tp u, Tp v, int l, int r, int x, int y) {
        if (x <= l && r <= y) {
            return Info::del(v->info, u->info);
        }
        int m = (l + r) >> 1;
        if (m >= y) {
            return range_query(u->ch[0], v->ch[0], l, m, x, y);
        } else if (m <= x) {
            return range_query(u->ch[1], v->ch[1], m, r, x, y);
        } else {
            return Info::merge(range_query(u->ch[0], v->ch[0], l, m, x, y),
range_query(u->ch[1], v->ch[1], m, r, x, y));
        }
    }
    Info range_query(Tp u, Tp v, int x, int y) {
        return range_query(u, v, 0, n, x, y);
    }
    int kth(Tp u, Tp v, int l, int r, i64 k) {
        i64 x = Info::del(u->info, v->info);
        if (x < k) {
            k -= x;
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = kth(u->ch[0], v->ch[0], l, m, k);
        if (res == -1) {
            res = kth(u->ch[1], v->ch[1], m, r, k);
        }
        return res;
    }
    int kth(Tp u, Tp v, i64 k) {
        return kth(u, v, 0, n, k);
    }
    void DFS(Tp t, int l, int r, int dep = 0) {
        if (!t) {
            return;
        }
        int m = (l + r) / 2;
        DFS(t->ch[0], l, m, dep + 1);
        cerr << string(dep, '\t');
        cerr << t->info << endl;
        DFS(t->ch[1], m, r, dep + 1);
    }
```

```
    void dfs(Tp t) {
        DFS(t, 0, n);
    }
};

struct Info {
    i64 cnt = 0;
    void apply(const Info &v) {
        cnt += v.cnt;
    }
    void update(const Info &lhs, const Info &rhs) {
        cnt = lhs.cnt + rhs.cnt;
    }
    Info del(const Info &lhs, const Info &rhs) {
        return {lhs.cnt - rhs.cnt};
    }
    Info merge(const Info &lhs, const Info &rhs) {
        Info info = Info();
        info.update(lhs, rhs);
        return info;
    }
    friend ostream &operator<<(ostream &cout, Info t) {
        return cout << "Info" << "; ";
    }
};

using SegmentTree = segment_tree<Info>;
```

李超线段树

```
template<typename T, class Line, class Cmp>
struct Li_Chao_SegmentTree {
    int n;
    std::vector<int> id;
    std::vector<T> real;
    std::vector<Line> line;
    Cmp cmp;
    Li_Chao_SegmentTree() {}
    Li_Chao_SegmentTree(int _n) {
        init(_n);
    }
    Li_Chao_SegmentTree(const std::vector<T> &_init) {
        init(_init);
    }
    void init(int _n) {
        std::vector<int> _init(_n);
        iota(_init.begin(), _init.end(), 0);
        init(_init);
    }
    void init(const std::vector<T> &_init) {
        n = _init.size();
        id.assign(4 << std::__lg(n), 0);
        line.push_back(Line());
        real = _init;
        sort(real.begin(), real.end());
```

```
        real.erase(std::unique(real.begin(), real.end()), real.end());
        real.push_back(real.back() + 1);
    }
    void rangeChange (int x, int y, Line add) {
        int u = line.size();
        line.push_back(add);
        std::function<void(int, int, int, int)>
        range_Change = [&] (int l, int r, int p, int u) {
            int &v = id[p], m = (l + r) / 2;
            if (cmp(line, u, v, real[m])) {
                swap(u, v);
            }
            if (cmp(line, u, v, real[l])) {
                range_Change(l, m, p * 2, u);
            }
            if (cmp(line, u, v, real[r - 1])) {
                range_Change(m, r, p * 2 + 1, u);
            }
        };
        std::function<void(int, int, int)>
        range_find = [&] (int l, int r, int p) {
            if (real[l] >= y || real[r] <= x) {
                return;
            }
            if (x <= real[l] && real[r] <= y) {
                range_Change(l, r, p, u);
                return;
            }
            int m = (l + r) / 2;
            range_find(l, m, p * 2);
            range_find(m, r, p * 2 + 1);
        };
        range_find(0, n, 1);
    }
    void insert(Line add) {
        rangeChange(real[0], real.back(), add);
    }
    int Query(int x) {
        std::function<int(int, int, int)>
        Query = [&] (int l, int r, int p) {
            if (r - l == 1) {
                return id[p];
            }
            int m = (l + r) / 2;
            int u = id[p], v = -1;
            if (x < real[m]) {
                v = Query(l, m, p * 2);
            } else {
                v = Query(m, r, p * 2 + 1);
            }
            return cmp(line, u, v, x) ? u : v;
        };
        return Query(0, n, 1);
    }
    T slope_dp_Query(int x) {
        return line[Query(x)](x);
```

```
    }
};

template<typename T>
struct Line {
    T k, b;
    Line(T k = 0, T b = 0) : k(k), b(b){}
    T operator()(T x) {
        return __int128(k) * x + b;
    }
};
template<>
struct Line<double> {
    double k, b;
    Line(double k = 0, double b = 0) : k(k), b(b){}
    template<typename T>
    Line(T x0, T y0, T x1, T y1) {
        if (x0 == x1) {
            k = 0;
            b = std::max(y0, y1);
        } else {
            k = (y0 - y1) / (0. + x0 - x1);
            b = y0 - k * x0;
        }
    }
    double operator()(double x) {
        return k * x + b;
    }
};

template<typename T>
struct Cmp {
    bool operator() (vector<Line<T>> &line, int u, int v, T x) {
        return line[u](x) < line[v](x) || (line[u](x) == line[v](x) && u < v);
    }
};
template<>
struct Cmp<double> {
    bool operator() (vector<Line<double>> &line, int u, int v, double x) {
        constexpr double exp = 1e-9;
        return line[u](x) - line[v](x) > exp || (std::abs(line[u](x) - line[v]
(x)) <= exp && u < v);
    }
};

template<typename T, typename T1 = int>
using SegmentTree =
    Li_Chao_SegmentTree<T1, Line<T>, Cmp<T>>;
```

扫描线

```
struct SegmentTree {
    SegmentTree() {}
    struct line {
        int h, l, r, add;
```

```
    friend bool operator<(const line &u, const line &v) {
        return u.h < v.h;
    }
};
vector<line> a;
vector<int> pos, len, tag;

void reserve(int n) {
    a.reserve(2 * n), pos.reserve(2 * n);
}

// 左下和右上 在笛卡尔坐标系中
void addRectangle(int x, int l, int y, int r) {
    a.emplace_back(x, l, r, 1);
    a.emplace_back(y, l, r, -1);
    pos.push_back(l);
    pos.push_back(r);
}

void addRange(int x, int y, int l, int r) {
    addRectangle(x, l, y + 1, r + 1);
}

void pull(int p, int l, int r) {
    if (tag[p]) len[p] = pos[r + 1] - pos[l];
    else len[p] = len[p << 1] + len[p << 1 | 1];
}

void modify(int p, int l, int r, int x, int y, int v) {
    if (x <= pos[l] && pos[r + 1] <= y) {
        tag[p] += v;
        pull(p, l, r);
        return;
    }
    int m = l + r >> 1;
    if (x <= pos[m])
        modify(p << 1, l, m, x, y, v);
    if (pos[m + 1] < y)
        modify(p << 1 | 1, m + 1, r, x, y, v);
    pull(p, l, r);
}

i64 answer() {
    if (a.empty()) return 0LL;
    i64 ans = 0;
    int n = a.size();
    sort(a.begin(), a.end());
    sort(pos.begin(), pos.end());
    int m = unique(pos.begin(), pos.end()) - pos.begin();
    len.assign(8 * m, 0);
    tag.assign(8 * m, 0);
    for (int i = 0; i < n - 1; i += 1) {
        modify(1, 0, m - 2, a[i].l, a[i].r, a[i].add);
        ans += 1LL * len[1] * (a[i + 1].h - a[i].h);
    }
    return ans;
```

```
    }
};
```

区间容斥

```
struct intervalRepulsion {
    intervalRepulsion() {}
    struct line {
        int h, l, r, add;
        friend bool operator<(const line &u, const line &v) {
            return u.h < v.h;
        }
    };
    vector<line> a;
    vector<int> pos, len, tag;
    vector<array<int, 2>> del; // triangle

    void reserve(int n) {
        a.reserve(2 * n), pos.reserve(2 * n);
    }

    // 左下和右上 在笛卡尔坐标系中
    void addRectangle(int x, int l, int y, int r) {
        a.emplace_back(x, l, r, 1);
        a.emplace_back(y, l, r, -1);
        pos.push_back(l);
        pos.push_back(r);
    }

    void addRange(int u, int v, int x, int y) {
        if (!(u <= v && x <= y)) {
            return;
        }
        if (x < v) {
            int le = v - x + 1;
            int l = x, r = v;
            le = y - x;
            del.push_back({v - le, le});
        }
        addRectangle(u, x, v + 1, y + 1);
    }

    void pull(int p, int l, int r) {
        if (tag[p]) len[p] = pos[r + 1] - pos[l];
        else len[p] = len[p << 1] + len[p << 1 | 1];
    }

    void modify(int p, int l, int r, int x, int y, int v) {
        if (x <= pos[l] && pos[r + 1] <= y) {
            tag[p] += v;
            pull(p, l, r);
            return;
        }
        int m = l + r >> 1;
        if (x <= pos[m])
```

```
            modify(p << 1, l, m, x, y, v);
        if (pos[m + 1] < y)
            modify(p << 1 | 1, m + 1, r, x, y, v);
        pull(p, l, r);
    }

    i64 calc1() {
        if (a.empty()) return 0LL;
        i64 ans = 0;
        int n = a.size();
        sort(a.begin(), a.end());
        sort(pos.begin(), pos.end());
        int m = unique(pos.begin(), pos.end()) - pos.begin();
        len.assign(8 * m, 0);
        tag.assign(8 * m, 0);
        for (int i = 0; i < n - 1; i += 1) {
            modify(1, 0, m - 2, a[i].l, a[i].r, a[i].add);
            ans += 1LL * len[1] * (a[i + 1].h - a[i].h);
        }
        return ans;
    }

    i64 calc2(int L, int R) {
        sort(del.begin(), del.end());
        del.push_back({R + 1, 0});
        int pos = L - 1, siz = 0;
        auto calc = [&] (int x, int y, int h) {
            return 1LL * (x + y) * h / 2;
        };
        i64 ans = 0;
        for (auto [l, le] : del) {
            if (l + le <= pos + siz) {
                continue;
            } else {
                if (pos + siz - 1 < l) {
                    ans += calc(siz, 1, siz);
                } else {
                    int h = l - pos;
                    ans += calc(siz, siz - h + 1, h);
                }
                pos = l, siz = le;
            }
        }
        return ans;
    }

    // 左右区间范围
    i64 answer(int L, int R) {
        i64 le = R - L + 1;
        i64 ans = le * (le + 1) / 2;
        ans = ans - calc1() + calc2(L, R);
        return ans;
    }
};
```

2SAT

```
template <class E> struct csr {
    vector<int> r;
    vector<E> e;
    csr(int n, const vector<pair<int, E>>& edges)
        : r(n + 1), e(edges.size()) {
        for (auto e : edges) {
            r[e.first + 1]++;
        }
        for (int i = 1; i <= n; i++) {
            r[i] += r[i - 1];
        }
        auto c = r;
        for (auto e : edges) {
            e[c[e.first]++] = e.second;
        }
    }
};

struct scc_graph {
    int n;
    struct E {
        int to;
    };
    vector<pair<int, E>> edges;

    scc_graph(int n) : n(n) {}

    void add_edge(int u, int v) { edges.push_back({u, {v}}); }

    pair<int, vector<int>> work() {
        auto g = csr<E>(n, edges);
        int now = 0, siz = 0;
        vector<int> vis, low(n), ord(n, -1), ids(n);
        vis.reserve(n);
        auto dfs = [&](auto &&self, int v) -> void {
            low[v] = ord[v] = now++;
            vis.push_back(v);
            for (int i = g.r[v]; i < g.r[v + 1]; i++) {
                auto to = g.e[i].to;
                if (ord[to] == -1) {
                    self(self, to);
                    low[v] = min(low[v], low[to]);
                } else {
                    low[v] = min(low[v], ord[to]);
                }
            }
            if (low[v] == ord[v]) {
                while (true) {
                    int u = vis.back();
                    vis.pop_back();
                    ord[u] = n;
```

```
                    ids[u] = siz;
                    if (u == v) break;
                }
                siz++;
            }
        };
        for (int i = 0; i < n; i++) {
            if (ord[i] == -1) dfs(dfs, i);
        }
        return {siz, ids};
    }

    vector<vector<int>> scc() {
        auto ids = work();
        int siz = ids.first;
        vector<int> c(siz);
        for (auto x : ids.second) c[x]++;
        vector<vector<int>> g(ids.first);
        for (int i = 0; i < siz; i++) {
            g[i].reserve(c[i]);
        }
        for (int i = 0; i < n; i++) {
            g[ids.second[i]].push_back(i);
        }
        return g;
    }
};

struct two_sat {
    int n;
    vector<bool> ans;
    scc_graph scc;

    two_sat() : n(0), scc(0) {}
    two_sat(int n) : n(n), ans(n), scc(2 * n) {}

    void addClause(int i, bool f, int j, bool g) {
        scc.add_edge(2 * i + (f ? 0 : 1), 2 * j + (g ? 1 : 0));
        scc.add_edge(2 * j + (g ? 0 : 1), 2 * i + (f ? 1 : 0));
    }
    void notClause(int u, bool f, int v, bool g) {
        addClause(u, !f, v, !g) ;
    }
    bool satisfiable() {
        auto id = scc.work().second;
        for (int i = 0; i < n; i++) {
            if (id[2 * i] == id[2 * i + 1]) return false;
            ans[i] = id[2 * i] > id[2 * i + 1];
        }
        return true;
    }
};
```

一份无封装可持久化线段树参考

```
struct node;
using Tp = u32_p<node>;
Tp _new() {
    return Tp::__new();
}
struct node {
    Tp ch[2];
    int val;
    int add;
};

int cnt = 0;

Tp _new(Tp t) {
    cnt += 1;
    Tp u = _new();
    u->val = t->val;
    u->add = t->add;
    u->ch[0] = t->ch[0];
    u->ch[1] = t->ch[1];
    return u;
}

void apply(Tp &t, int val) {
    t = _new(t);
    t->val += val;
    t->add += val;
}

void push(Tp &t) {
    int val = t->add;
    if (val) {
        apply(t->ch[0], val);
        apply(t->ch[1], val);
        t->add = 0;
    }
}

void rangeAdd(Tp &t, int l, int r, int x, int y, int val) {
    t = _new(t);
    if (x <= l && r <= y) {
        t->add += val;
        t->val += val;
        return;
    }
    push(t);
    int m = l + r >> 1;
    if (m > x) {
        rangeAdd(t->ch[0], l, m, x, y, val);
    }
    if (m < y) {
        rangeAdd(t->ch[1], m, r, x, y, val);
    }
```

```
}

int query(Tp &t, int x, int l, int r) {
    t = _new(t);
    if (r - l == 1) {
        return t->val;
    }
    push(t);
    int m = l + r >> 1;
    return m > x ? query(t->ch[0], x, l, m) : query(t->ch[1], x, m, r);
}

void modify(Tp &t, int x, int l, int r) {
    t = _new(t);
    if (r - l == 1) {
        t->val = 0;
        return;
    }
    push(t);
    int m = l + r >> 1;
    m > x ? modify(t->ch[0], x, l, m) : modify(t->ch[1], x, m, r);
}

void merge(Tp &u, Tp &v, Tp &t, int c, int l, int r) {
    if (r <= c) {
        t = u;
        return;
    }
    if (l >= c) {
        t = v;
        return;
    }
    t = _new();
    int m = l + r >> 1;
    u = _new(u);
    push(u);
    v = _new(v);
    push(v);
    merge(u->ch[0], v->ch[0], t->ch[0], c, l, m);
    merge(u->ch[1], v->ch[1], t->ch[1], c, m, r);
}

void dfs(Tp t, int l, int r, int dep = 0) {
# ifdef LOCAL
    if (!t) {
        return;
    }
    int m = l + r >> 1;
    dfs(t->ch[0], l, m, dep + 1);
    cerr << string(dep, '\t');
    cerr << l << ' ' << r << ' ' << t->val << ' ' << t->add << '\n';
    dfs(t->ch[1], m, r, dep + 1);
# endif
}
```