

数据结构

linkCutTree

标准版

LazyLinkCutTree

维护子树信息

动态维护直径

RMQ

catTree

状压rmq

ST表

并查集

标准

可持久化

可撤回

平衡树

set

FHQtreap

替罪羊树

区间操作

FhqTreap

FhqTreap带懒标

splay

带父亲的FhqTreap

可持久化

参考旧版

FHQtreap

splay

treap

可持久化文艺平衡树

jls splay

线段树套平衡树

树状数组

标准版

二维树状数组

区间加树状数组

线段树

单点

区间

tourist zkw 线段树（精简版）区间最大值

动态开点线段树

可持久化线段树

李超线段树

扫描线

区间容斥

2SAT

一份无封装可持久化线段树参考

数学

取模类

多项式

标准

c++版本修复

FFT

更快的FFT

- 更快的NTT
- 多项式扩展包
- 矩阵
- 数学类
 - base
 - Exgcd
 - 中国剩余定理
 - 质因数分解，素数检验
 - 扩展欧拉定理
 - 扩展卢卡斯定理
 - 扩展大步小步算法
 - n次剩余
 - 原根
 - 原根2
 - 旧版参考
- 线性基
- 线性筛
- 组合数学
- 行列式
- 高斯消元

图论

- SCC
 - 一般
 - 割边
 - 边双
 - 割点
 - 点双
- 倍增lca
- 带权
- 虚树
- 重链剖分
 - 重链剖分套线段树
- 流
 - 网络流
 - 网络流前向星
 - 网络流未封装
 - 费用流
 - 费用流前向星
 - 费用流多类型EK
 - 费用流原始对偶
 - 费用流多类型原始对偶
 - 单纯形
 - 单纯形未封装
- 笛卡尔树
- 板题实现
 - 欧拉图
- 静态圆方树
- 存图
 - 稀疏行压缩

杂

- 初始
- 对拍
- 简易版取模类
- 取模类丐版
- debug.h

hash

$O2$ 优化

快读

u32 指针

字符串

*Ac*自动机

字符串哈希

后缀数组

KMP

Trie

Manacher

Z函数

PAM

SAM

子序列自动机

动态规划

dp优化

斜率优化

板子：x 单调, k 单调

板子：x 单调, k 不单调

板子：x 不单调, k 不单调

计算几何

二维计算几何基础

凸包

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) {
#ifdef ONLINE_JUDGE
    cerr << "\n!ct 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) {
#ifdef ONLINE_JUDGE
        cerr << "\n\trct 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) {
#ifdef ONLINE_JUDGE
    cerr << "\n!ct 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() {}
#ifdef 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(1 ^ (r - l));
        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 = 1; j < r; j++) {
                while (s && cmp(v[j], v[std::__lg(s) + 1])) {
                    s ^= 1ULL << std::__lg(s);
                }
            }
        }
    }
};
```

```

        s |= 1ULL << (j - 1);
        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)) + 1];
    }
}
};

```

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 = 1;
        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, siz_u, fav] = h.back();
            siz[u] = siz_u;
            fa[v] = fav;
            h.pop_back();
            lazy[v] += lazy[u];
        }
    }
};

```

平衡树

set

FHQtreap

```

/**
 * 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;
    }
}

```

```

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, l, 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
}

```

数学

取模类

```

using i64 = long long;
template<class T>
constexpr T power(T a, i64 b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}

constexpr i64 mul(i64 a, i64 b, i64 p) {
    i64 res = a * b - i64(1.L * a * b / p) * p;
    res %= p;
    if (res < 0) {
        res += p;
    }
    return res;
}

template<i64 P>
struct MLong {
    i64 x;
    constexpr MLong() : x{} {}
    constexpr MLong(i64 x) : x{norm(x % getMod())} {}

    static i64 Mod;
};

```

```

constexpr static i64 getMod() {
    if (P > 0) {
        return P;
    } else {
        return Mod;
    }
}

constexpr static void setMod(i64 Mod_) {
    Mod = Mod_;
}

constexpr i64 norm(i64 x) const {
    if (x < 0) {
        x += getMod();
    }
    if (x >= getMod()) {
        x -= getMod();
    }
    return x;
}

constexpr i64 val() const {
    return x;
}

explicit constexpr operator i64() const {
    return x;
}

constexpr MLong operator-() const {
    MLong res;
    res.x = norm(getMod() - x);
    return res;
}

constexpr MLong inv() const {
    assert(x != 0);
    return power(*this, getMod() - 2);
}

constexpr MLong &operator*=(MLong rhs) & {
    x = mul(x, rhs.x, getMod());
    return *this;
}

constexpr MLong &operator+=(MLong rhs) & {
    x = norm(x + rhs.x);
    return *this;
}

constexpr MLong &operator--(MLong rhs) & {
    x = norm(x - rhs.x);
    return *this;
}

constexpr MLong &operator/=(MLong rhs) & {
    return *this *= rhs.inv();
}

friend constexpr MLong operator*(MLong lhs, MLong rhs) {
    MLong res = lhs;
    res *= rhs;
    return res;
}

friend constexpr MLong operator+(MLong lhs, MLong rhs) {
    MLong res = lhs;

```

```

        res += rhs;
        return res;
    }
    friend constexpr MLong operator-(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res -= rhs;
        return res;
    }
    friend constexpr MLong operator/(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr std::istream &operator>>(std::istream &is, MLong &a) {
        i64 v;
        is >> v;
        a = MLong(v);
        return is;
    }
    friend constexpr std::ostream &operator<<(std::ostream &os, const MLong &a) {
        return os << a.val();
    }
    friend constexpr bool operator==(MLong lhs, MLong rhs) {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(MLong lhs, MLong rhs) {
        return lhs.val() != rhs.val();
    }
};

template<>
i64 MLong<OLL>::Mod = i64(1E18) + 9;

template<int P>
struct MInt {
    int x;
    constexpr MInt() : x{} {}
    constexpr MInt(i64 x) : x{norm(x % getMod())} {}

    static int Mod;
    constexpr static int getMod() {
        if (P > 0) {
            return P;
        } else {
            return Mod;
        }
    }
    constexpr static void setMod(int Mod_) {
        Mod = Mod_;
    }
    constexpr int norm(int x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x >= getMod()) {
            x -= getMod();
        }
    }
};

```

```

    }
    return x;
}
constexpr int val() const {
    return x;
}
explicit constexpr operator int() const {
    return x;
}
}
explicit constexpr operator i64() const {
    return x;
}
}
constexpr MInt operator-() const {
    MInt res;
    res.x = norm(getMod() - x);
    return res;
}
constexpr MInt inv() const {
    assert(x != 0);
    return power(*this, getMod() - 2);
}
constexpr MInt &operator*=(MInt rhs) & {
    x = 1LL * x * rhs.x % getMod();
    return *this;
}
constexpr MInt &operator+=(MInt rhs) & {
    x = norm(x + rhs.x);
    return *this;
}
constexpr MInt &operator-=(MInt rhs) & {
    x = norm(x - rhs.x);
    return *this;
}
constexpr MInt &operator/=(MInt rhs) & {
    return *this *= rhs.inv();
}
}
friend constexpr MInt operator*(MInt lhs, MInt rhs) {
    MInt res = lhs;
    res *= rhs;
    return res;
}
friend constexpr MInt operator+(MInt lhs, MInt rhs) {
    MInt res = lhs;
    res += rhs;
    return res;
}
friend constexpr MInt operator-(MInt lhs, MInt rhs) {
    MInt res = lhs;
    res -= rhs;
    return res;
}
friend constexpr MInt operator/(MInt lhs, MInt rhs) {
    MInt res = lhs;
    res /= rhs;
    return res;
}
}

```

```

    friend constexpr std::istream &operator>>(std::istream &is, MInt &a) {
        i64 v;
        is >> v;
        a = MInt(v);
        return is;
    }
    friend constexpr std::ostream &operator<<(std::ostream &os, const MInt &a) {
        return os << a.val();
    }
    friend constexpr bool operator==(MInt lhs, MInt rhs) {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(MInt lhs, MInt rhs) {
        return lhs.val() != rhs.val();
    }
};

template<>
int MInt<0>::Mod = 998244353;

template<int V, int P>
constexpr MInt<P> CInv = MInt<P>(V).inv();

constexpr int P = 998244353;
using Z = MInt<P>;

```

多项式

标准

```

std::vector<int> rev;
template<int P>
std::vector<MInt<P>> roots{0, 1};

template<int P>
constexpr MInt<P> findPrimitiveRoot() {
    MInt<P> i = 2;
    int k = __builtin_ctz(P - 1);
    while (true) {
        if (power(i, (P - 1) / 2) != 1) {
            break;
        }
        i += 1;
    }
    return power(i, (P - 1) >> k);
}

template<int P>
constexpr MInt<P> primitiveRoot = findPrimitiveRoot<P>();

template<>
constexpr MInt<998244353> primitiveRoot<998244353> {31};

template<int P>

```

```

constexpr void dft(std::vector<MInt<P>> &a) {
    int n = a.size();

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

    for (int i = 0; i < n; i++) {
        if (rev[i] < i) {
            std::swap(a[i], a[rev[i]]);
        }
    }

    if (roots<P>.size() < n) {
        int k = __builtin_ctz(roots<P>.size());
        roots<P>.resize(n);
        while ((1 << k) < n) {
            auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k - 1));

            for (int i = 1 << (k - 1); i < (1 << k); i++) {
                roots<P>[2 * i] = roots<P>[i];
                roots<P>[2 * i + 1] = roots<P>[i] * e;
            }
            k++;
        }
    }

    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                MInt<P> u = a[i + j];
                MInt<P> v = a[i + j + k] * roots<P>[k + j];
                a[i + j] = u + v;
                a[i + j + k] = u - v;
            }
        }
    }
}

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

template<int P = ::P>
struct Poly : public std::vector<MInt<P>> {
    using Value = MInt<P>;

```

```

Poly() : std::vector<Value>() {}
explicit constexpr Poly(int n) : std::vector<Value>(n) {}

explicit constexpr Poly(const std::vector<Value> &a) : std::vector<Value>(a)
{}

constexpr Poly(const std::initializer_list<Value> &a) : std::vector<Value>(a)
{}

template<class InputIt, class = std::_RequireInputIter<InputIt>>
explicit constexpr Poly(InputIt first, InputIt last) : std::vector<Value>
(first, last) {}

template<class F>
explicit constexpr Poly(int n, F f) : std::vector<Value>(n) {
    for (int i = 0; i < n; i++) {
        (*this)[i] = f(i);
    }
}

constexpr Poly shift(int k) const {
    if (k >= 0) {
        auto b = *this;
        b.insert(b.begin(), k, 0);
        return b;
    } else if (this->size() <= -k) {
        return Poly();
    } else {
        return Poly(this->begin() + (-k), this->end());
    }
}

constexpr Poly trunc(int k) const {
    Poly f = *this;
    f.resize(k);
    return f;
}

constexpr friend Poly operator+(const Poly &a, const Poly &b) {
    Poly res(std::max(a.size(), b.size()));
    for (int i = 0; i < a.size(); i++) {
        res[i] += a[i];
    }
    for (int i = 0; i < b.size(); i++) {
        res[i] += b[i];
    }
    return res;
}

constexpr friend Poly operator-(const Poly &a, const Poly &b) {
    Poly res(std::max(a.size(), b.size()));
    for (int i = 0; i < a.size(); i++) {
        res[i] += a[i];
    }
    for (int i = 0; i < b.size(); i++) {
        res[i] -= b[i];
    }
    return res;
}

constexpr friend Poly operator-(const Poly &a) {

```



```

std::vector<Value> res(a.size());
for (int i = 0; i < int(res.size()); i++) {
    res[i] = -a[i];
}
return Poly(res);
}

constexpr friend Poly operator*(Poly a, Poly b) {
    if (a.size() == 0 || b.size() == 0) {
        return Poly();
    }
    if (a.size() < b.size()) {
        std::swap(a, b);
    }
    int n = 1, tot = a.size() + b.size() - 1;
    while (n < tot) {
        n *= 2;
    }
    if (((P - 1) & (n - 1)) != 0 || b.size() < 128) {

        Poly c(a.size() + b.size() - 1);
        for (int i = 0; i < a.size(); i++) {
            for (int j = 0; j < b.size(); j++) {
                c[i + j] += a[i] * b[j];
            }
        }
        return c;
    }
    a.resize(n);
    b.resize(n);
    dft(a);
    dft(b);
    for (int i = 0; i < n; ++i) {
        a[i] *= b[i];
    }
    idft(a);
    a.resize(tot);
    return a;
}

constexpr friend Poly operator*(Value a, Poly b) {
    for (int i = 0; i < int(b.size()); i++) {
        b[i] *= a;
    }
    return b;
}

constexpr friend Poly operator*(Poly a, Value b) {
    for (int i = 0; i < int(a.size()); i++) {
        a[i] *= b;
    }
    return a;
}

constexpr friend Poly operator/(Poly a, Value b) {
    for (int i = 0; i < int(a.size()); i++) {
        a[i] /= b;
    }
    return a;
}

```

```

constexpr Poly &operator+=(Poly b) {
    return (*this) = (*this) + b;
}
constexpr Poly &operator-=(Poly b) {
    return (*this) = (*this) - b;
}
constexpr Poly &operator*=(Poly b) {
    return (*this) = (*this) * b;
}
constexpr Poly &operator*=(Value b) {
    return (*this) = (*this) * b;
}
constexpr Poly &operator/=(Value b) {
    return (*this) = (*this) / b;
}
template <class T>
constexpr Value operator() ( T x ) {
    Value ans = 0 ;
    Value cnt = 1 ;
    for ( int i = 0 ; i < this->size () ; ++ i ) {
        ans += (* this) [ i ] * cnt ;
        cnt *= x ;
    }
    return ans ;
}
constexpr Poly deriv() const {
    if (this->empty()) {
        return Poly();
    }
    assert (this->size() != 0) ;
    Poly res(this->size() - 1);
    for (int i = 0; i < this->size() - 1; ++i) {
        res[i] = (i + 1) * (*this)[i + 1];
    }
    return res;
}
constexpr Poly integr() const {
    Poly res(this->size() + 1);
    for (int i = 0; i < this->size(); ++i) {
        res[i + 1] = (*this)[i] / (i + 1);
    }
    return res;
}
constexpr Poly inv(int m) const {
    Poly x{(*this)[0].inv()};
    int k = 1;
    while (k < m) {
        k *= 2;
        x = (x * (Poly{2} - trunc(k) * x)).trunc(k);
    }
    return x.trunc(m);
}
constexpr Poly log(int m) const {
    return (deriv() * inv(m)).integr().trunc(m);
}
constexpr Poly exp(int m) const {

```

```

Poly x{1};
int k = 1;
while (k < m) {
    k *= 2;
    x = (x * (Poly{1} - x.log(k) + trunc(k))).trunc(k);
}
return x.trunc(m);
}

constexpr Poly pow(int k, int m) const {
    int i = 0;
    while (i < this->size() && (*this)[i] == 0) {
        i++;
    }
    if (i == this->size() || 1LL * i * k >= m) {
        return Poly(m);
    }
    value v = (*this)[i];
    auto f = shift(-i) * v.inv();
    return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v, k);
}

constexpr Poly pow(int k, int m, int k2) const {
    int i = 0;
    while (i < this->size() && (*this)[i] == 0) {
        i++;
    }
    if (i == this->size() || 1LL * i * k >= m) {
        return Poly(m);
    }
    value v = (*this)[i];
    auto f = shift(-i) * v.inv();
    return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v, k2);
}

constexpr Poly sqrt(int m) const {
    Poly x{1};
    int k = 1;
    while (k < m) {
        k *= 2;
        x = (x + (trunc(k) * x.inv(k)).trunc(k)) * CInv<2, P>;
    }
    return x.trunc(m);
}

constexpr Poly inv() const {
    return move (inv(this->size ())) ;
}

constexpr Poly log() const {
    return move(log(this->size ()));
}

constexpr Poly exp() const {
    return move(exp(this->size ()));
}

constexpr Poly pow(i64 b) const {
    Poly<> res (vector <Z> { 1 }) ;
    auto a = * this ;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;

```

```

    }
}
return res;
}
constexpr Poly sqrt() const {
    return move(sqrt(this->size()));
}
constexpr Poly mult(Poly b) const {
    if (b.size() == 0) {
        return Poly();
    }
    int n = b.size();
    std::reverse(b.begin(), b.end());
    return ((*this) * b).shift(-(n - 1));
}
constexpr std::vector<Value> eval(std::vector<Value> x) const {
    if (this->size() == 0) {
        return std::vector<Value>(x.size(), 0);
    }
    const int n = std::max(x.size(), this->size());
    std::vector<Poly> q(4 * n);
    std::vector<Value> ans(x.size());
    x.resize(n);
    std::function<void(int, int, int)> build = [&](int p, int l, int r) {
        if (r - l == 1) {
            q[p] = Poly{1, -x[l]};
        } else {
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            q[p] = q[2 * p] * q[2 * p + 1];
        }
    };
    build(1, 0, n);
    std::function<void(int, int, int, const Poly &)> work = [&](int p, int l,
int r, const Poly &num) {
        if (r - l == 1) {
            if (l < int(ans.size())) {
                ans[l] = num[0];
            }
        } else {
            int m = (l + r) / 2;
            auto need = move(num.mult(q[2 * p + 1]));
            need.resize ( m - 1 );
            work(2 * p, l, m, need);
            need = move(num.mult(q[2 * p]));
            need.resize ( r - m );
            work(2 * p + 1, m, r, need);
        }
    };
    work(1, 0, n, mult(q[1].inv(n)));
    return ans;
}
};
template<int P = ::P>

```

```

Poly<P> berlekampMassey(const Poly<P> &s) {
    Poly<P> c;
    Poly<P> oldC;
    int f = -1;
    for (int i = 0; i < s.size(); i++) {
        auto delta = s[i];
        for (int j = 1; j <= c.size(); j++) {
            delta -= c[j - 1] * s[i - j];
        }
        if (delta == 0) {
            continue;
        }
        if (f == -1) {
            c.resize(i + 1);
            f = i;
        } else {
            auto d = oldC;
            d *= -1;
            d.insert(d.begin(), 1);
            MInt<P> df1 = 0;
            for (int j = 1; j <= d.size(); j++) {
                df1 += d[j - 1] * s[f + 1 - j];
            }
            assert(df1 != 0);
            auto coef = delta / df1;
            d *= coef;
            Poly<P> zeros(i - f - 1);
            zeros.insert(zeros.end(), d.begin(), d.end());
            d = zeros;
            auto temp = c;
            c += d;
            if (i - temp.size() > f - oldC.size()) {
                oldC = temp;
                f = i;
            }
        }
    }
    c *= -1;
    c.insert(c.begin(), 1);
    return c;
}

```

```

template<int P = ::P>
MInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
    int m = q.size() - 1;
    while (n > 0) {
        auto newq = q;
        for (int i = 1; i <= m; i += 2) {
            newq[i] *= -1;
        }
        auto newp = p * newq;
        newq = q * newq;
        for (int i = 0; i < m; i++) {
            p[i] = newp[i * 2 + n % 2];
        }
    }
}

```

```

        for (int i = 0; i <= m; i++) {
            q[i] = newq[i * 2];
        }
        n /= 2;
    }
    return p[0] / q[0];
}

```

C++版本修复

```

#include <vector>
#include <tuple>    // for std::tie
#include <utility>  // for std::make_pair

using cp = complex<double>;
using _Tp = vector<cp>::iterator;

struct cxx20_It : public _Tp {
    using _Tp::_Tp;
    cxx20_It(const _Tp& it) : _Tp(it) {}
    cp &operator [](int x) {
        return *(*this + x);
    }
};
using It = cxx20_It;

```

FFT

```

const double PI = acos(-1.0);

struct Complex {
    double x, y;
    Complex(double _x = 0.0, double _y = 0.0) {
        x = _x;
        y = _y;
    }
    Complex operator-(const Complex &b) const {
        return Complex(x - b.x, y - b.y);
    }
    Complex operator+(const Complex &b) const {
        return Complex(x + b.x, y + b.y);
    }
    Complex operator*(const Complex &b) const {
        return Complex(x * b.x - y * b.y, x * b.y + y * b.x);
    }
    friend ostream &operator<<(ostream &cout, Complex u) {
        return cout << "(" << u.x << ", " << u.y << ")";
    }
};

void change(vector<Complex> &y) {
    int len = y.size();
    for (int i = 1, j = len / 2; i < len - 1; i++) {
        if (i < j) std::swap(y[i], y[j]);
    }
}

```

```

        int k = len / 2;
        while (j >= k) {
            j = j - k;
            k = k / 2;
        }
        if (j < k) j += k;
    }
}

void fft(vector<Complex> &y, int on) {
    int len = y.size();
    change(y);
    for (int h = 2; h <= len; h <= 1) {
        Complex wn(cos(2 * PI / h), sin(on * 2 * PI / h));
        for (int j = 0; j < len; j += h) {
            Complex w(1, 0);
            for (int k = j; k < j + h / 2; k++) {
                Complex u = y[k];
                Complex t = w * y[k + h / 2];
                y[k] = u + t;
                y[k + h / 2] = u - t;
                w = w * wn;
            }
        }
    }
    if (on == -1) {
        for (int i = 0; i < len; i++) {
            y[i].x /= len;
        }
    }
}

struct Poly : public vector<double> {
    using std::vector<double>::vector;
    friend Poly operator+(Poly x, Poly y) {
        int n = std::max(x.size(), y.size());
        Poly z(n);
        for (int i = 0; i < x.size(); i += 1) {
            z[i] += x[i];
        }
        for (int i = 0; i < y.size(); i += 1) {
            z[i] += y[i];
        }
        return z;
    }
    friend Poly operator*(Poly x, Poly y) {
        int len = x.size() + y.size();
        int n = 1;
        while (n < len) {
            n *= 2;
        }
        vector<Complex> _x(n), _y(n);
        for (int i = 0; i < x.size(); i += 1) {
            _x[i].x = x[i];
        }
        for (int i = 0; i < y.size(); i += 1) {

```

```

        _y[i].x = y[i];
    }
    fft(_x, 1), fft(_y, 1);
    for (int i = 0; i < n; i += 1) {
        _x[i] = _x[i] * _y[i];
    }
    fft(_x, -1);
    Poly ans(n);
    for (int i = 0; i < n; i += 1) {
        ans[i] = _x[i].x;
    }
    return ans;
}
Poly operator-(Poly a) {
    Poly a = *this;
    for (auto &i : a) {
        i = -i;
    }
    return a;
}
friend Poly operator-(Poly x, Poly y) {
    return x + -y;
}

friend Poly operator*(Poly x, int _mul) {
    for(int i = 0; i < x.size(); ++i) {
        x[i] *= _mul;
    }
    return x;
};
friend Poly operator*(int _mul, Poly x) {
    return x * _mul;
};

Poly &operator+=(Poly y) {
    return *this = *this + y;
}
Poly &operator-=(Poly y) {
    return *this = *this - y;
}
Poly &operator*=(Poly y) {
    return *this = *this * y;
}

Poly &operator*=(int y) {
    return *this = *this * y;
}

template<typename T>
operator vector<T>() {
    vector<T> a(size());
    for (int i = 0; i < size(); i += 1) {
        a[i] = T(this->operator[](i) + 0.5);
    }
    return a;
}
};

```


更快的FFT

```
constexpr double pi = 3.141592653589793115997963468544185161590576171875, pi2 = 2
* pi;

using cp = complex<double>;
using It = vector<cp>::iterator;
vector<cp> a;

template<int on, int n>
void fft(It a) {
    if (n == 1)
        return;
    if (n == 2) {
        tie(a[0], a[1]) = make_pair(a[0] + a[1], a[0] - a[1]);
        return;
    }
    constexpr int h = n >> 1, q = n >> 2;
    if (on == -1) {
        fft<on, h>(a);
        fft<on, q>(a + h);
        fft<on, q>(a + h + q);
    }
    cp w(1, 0), w3(1, 0);
    constexpr cp wn(cos(pi2 / n), on * sin(pi2 / n)),
        wn3(cos(pi2 * 3 / n), on * sin(pi2 * 3 / n));
    for (int i = 0; i < q; i++) {
        if (on == -1) {
            cp tmp1 = w * a[i + h], tmp2 = w3 * a[i + h + q],
                x = a[i], y = tmp1 + tmp2,
                x1 = a[i + q], y1 = tmp1 - tmp2;
            y1 = cp(y1.imag(), -y1.real());
            a[i] += y;
            a[i + q] += y1;
            a[i + h] = x - y;
            a[i + h + q] = x1 - y1;
        } else {
            cp x = a[i] - a[i + h], y = a[i + q] - a[i + h + q];
            y = cp(y.imag(), -y.real());
            a[i] += a[i + h];
            a[i + q] += a[i + h + q];
            a[i + h] = (x - y) * w;
            a[i + h + q] = (x + y) * w3;
        }
        w *= wn;
        w3 *= wn3;
    }
    if (on == 1) {
        fft<on, h>(a);
        fft<on, q>(a + h);
        fft<on, q>(a + h + q);
    }
}
```

```

    }
}

template<>
void fft<1, 0> (It a) {}
template<>
void fft<-1, 0> (It a) {}

template<int on>
void FFT(It a, int n) {
    # define C(x)\
        case 1 << x:\
            fft<on, 1 << x>(a);\
            break
    switch (n) {
        C(1);
        C(2);
        C(3);
        C(4);
        C(5);
        C(6);
        C(7);
        C(8);
        C(9);
        C(10);
        C(11);
        C(12);
        C(13);
        C(14);
        C(15);
        C(16);
        C(17);
        C(18);
        C(19);
        C(20);
        C(21);
    }
    # undef C
}

vector<cp> _x;
struct Poly : public vector<double> {
    using std::vector<double>::vector;
    friend Poly operator+(Poly x, Poly y) {
        int n = std::max(x.size(), y.size());
        Poly z(n);
        for (int i = 0; i < x.size(); i += 1) {
            z[i] += x[i];
        }
        for (int i = 0; i < y.size(); i += 1) {
            z[i] += y[i];
        }
        return z;
    }
    friend Poly operator*(Poly &x, Poly &y) {
        int len = x.size() + y.size() + 1;

```

```

    int n = 1;
    while (n < len) {
        n *= 2;
    }
    _x.assign(n, {});
    for (int i = 0; i < x.size(); i += 1) {
        _x[i].real(x[i]);
    }
    for (int i = 0; i < y.size(); i += 1) {
        _x[i].imag(y[i]);
    }
    FFT<1>(_x.begin(), n);
    for (int i = 0; i < n; i += 1) {
        _x[i] *= _x[i];
    }
    FFT<-1>(_x.begin(), n);
    Poly ans(n);
    const double inv = 0.5 / n;
    for (int i = 0; i < n; i += 1) {
        ans[i] = _x[i].imag() * inv;
    }
    return ans;
}

Poly operator-() {
    Poly a = *this;
    for (auto &i : a) {
        i = -i;
    }
    return a;
}

friend Poly operator-(Poly x, Poly y) {
    return x + -y;
}

Poly &operator+=(Poly y) {
    return *this = *this + y;
}

Poly &operator-=(Poly y) {
    return *this = *this - y;
}

Poly &operator*=(Poly y) {
    return *this = *this * y;
}

template<typename T>
operator vector<T>() {
    vector<T> a(size());
    for (int i = 0; i < size(); i += 1) {
        a[i] = T(this->operator[](i) + 0.5);
    }
    return a;
}

};

```

更快的NTT

```
#include <bits/stdc++.h>
#include <immintrin.h>
using namespace std;

using u32 = uint32_t;
using i64 = int64_t;
using u64 = uint64_t;
using ci = const int;

constexpr int insZ = 1 << 17, outSZ = 1 << 21;
char ibuf[insZ], *in1 = ibuf, *in2 = ibuf;
char obuf[outSZ], *out1 = obuf, *out2 = obuf + outSZ;

inline char gc() {
    if (__builtin_expect(in1 == in2 && (in2 = (in1 = ibuf) +
        fread(ibuf, 1, insZ, stdin), in1 == in2), 0)) return EOF;
    return *in1++;
}

inline void flush() {
    fwrite(obuf, 1, out1 - obuf, stdout);
    out1 = obuf;
}

#define pc(c) (*out1++ = c)

inline void read(int &x) {
    x = 0; static char c;
    while (!isdigit((c = gc())));
    while (x = 10 * x + (c ^ 48), isdigit(c = gc()));
}

inline void write(int x) {
    if (__builtin_expect(out1 + 20 > out2, 0)) flush();
    int tot = 0;
    do { pc(x % 10 + 48); } while (++tot, x /= 10);
    reverse(out1 - tot, out1);
}

constexpr int N = 1 << 21;
constexpr int mod = 998244353, g = 3;
inline int dil(int x) { return x >> 31 ? x + mod : x; }
inline int mu(int x, int y) { return u64(x) * y % mod; }

inline int qpow(int x, int y) {
    int z = 1;
    do { if (y & 1) z = mu(z, x); x = mu(x, x); } while (y >>= 1);
    return z;
}

inline int bceil(int x) { return 1 << __lg(x - 1) + 1; }
```

```

int w[N >> 1], iw[N >> 1];
void preNTT(int n) {
    int l = bceil(n) >> 1;
    w[0] = iw[0] = 1;
    for (int i = 1; i < l; i <= 1) {
        w[i] = qpow(g, (mod - 1 >> 2) / i);
        iw[i] = qpow(g, mod - 1 - (mod - 1 >> 2) / i);
    }
    for (int i = 1; i < l; ++i) {
        w[i] = mu(w[i & (i - 1)], w[i & -i]);
        iw[i] = mu(iw[i & (i - 1)], iw[i & -i]);
    }
}

struct poly : vector<int> {
    friend void dif(poly &f, int lim) {
        f.resize(lim);
        for (int l = lim >> 1, r = lim; l >>= 1, r >>= 1)
            for (int i = 0, *o = w; i != lim; i += r, ++o)
                for (int j = i, x, y; j != i + 1; ++j)
                    x = dil(f[j] - mod), y = mu(f[j + 1], *o), f[j] = x + y, f[j
+ 1] = x - y + mod;
        for (int i = 0; i < lim; ++i) f[i] = dil(f[i] - mod);
    }
    friend void dit(poly &f, int lim) {
        f.resize(lim);
        for (int l = 1, r = 2; l < lim; l <= 1, r <= 1)
            for (int i = 0, *o = iw; i != lim; i += r, ++o)
                for (int j = i, x, y; j != i + 1; ++j)
                    x = f[j], y = mod - f[j + 1], f[j] = dil(x - y), f[j + 1] =
mu(x + y, *o);
        ci iv = mod - (mod - 1) / lim;
        for (int i = 0; i < lim; ++i) f[i] = mu(f[i], iv);
    }
    friend poly operator*(poly f, poly g) {
        int len = f.size() + g.size() - 1;
        int lim = bceil(len);
        f.resize(lim), g.resize(lim);
        preNTT(lim); dif(f, lim); dif(g, lim);
        for (int i = 0; i < lim; ++i) f[i] = mu(f[i], g[i]);
        dit(f, lim); f.resize(len);
        return forward<poly>(f);
    }
};

poly F, G;

int main() {
    int n, m, lim;
    read(n), read(m);
    F.resize(n + 1), G.resize(m + 1);
    lim = bceil(n + m + 1);
    for (int i = 0; i <= n; ++i) read(F[i]);
    for (int i = 0; i <= m; ++i) read(G[i]);
    F = F * G;
    for (int i = 0; i <= n + m; ++i) write(F[i]), pc(' ');
}

```

```

    return flush(), 0;
}

```

多项式扩展包

```

/**
 * 多项式扩展包
 */
namespace ExPoly {
    template<int P = ::P, class T1, class T2>
    constexpr static Poly <P> Lagrange(T1 x, T2 y) {
        int n = x.size();
        vector <Poly<>> M(4 * n);
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                M[p] = Poly{(int) -x[l], 1};
            } else {
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m, r);
                M[p] = M[2 * p] * M[2 * p + 1];
            }
        };
        build(1, 0, n);
        auto M_ = M[1].deriv().eval(x);
        for (int i = 0; i < n; ++i) {
            M_[i] = y[i] * M_[i].inv();
        }
        vector <Poly<>> f(4 * n);
        std::function<void(int, int, int)> work = [&](int p, int l, int r) ->
void {
            if (r - l == 1) {
                if (l < n) {
                    f[p] = Poly{(int) M_[l]};
                }
            } else {
                int m = (l + r) / 2;
                work(2 * p, l, m);
                work(2 * p + 1, m, r);
                f[p] = f[2 * p] * M[2 * p + 1] + f[2 * p + 1] * M[2 * p];
            }
        };
        work(1, 0, n);
        return f[1];
    }

    /**
     * 作用：对多项式进行平移操作
     * 时间复杂度 $O(n\log(n))$ 
     */
    template<int P = ::P>
    constexpr static Poly <P> Polynomial_translation(Poly <P> f, int k) {
        i64 n = (i64) f.size() - 1;

```

```

    Poly <P> g(n + 1);
    Z res = 1;
    for (int i = 0; i <= n; ++i) {
        g[n - i] = res * comb.invfac(i);
        res *= k;
        f[i] *= comb.fac(i);
    }
    Poly <P> here = g * f;
    here = here.shift(-n);
    for (int i = 0; i <= n; ++i) {
        here[i] *= comb.invfac(i);
    }
    return here;
}

/**
 *作用：对相同的n对i \in ( 0 , n ) 求出将n个不同的元素划分为i个非空集的方案数
 *第二类Stirling数
 *时间复杂度O(nlog(n))
 */
template<int P = ::P>
constexpr static Poly <P> Second_Stirling_Same_N(int n) {
    Poly <P> f(n + 1), g(n + 1);
    for (int i = 0; i <= n; ++i) {
        g[i] = (i & 1 ? (Z) - 1 : Z(1)) * comb.invfac(i);
        f[i] = power((Z) i, n) * comb.invfac(i);
    }
    f *= g;
    f.resize(n + 1);
    return f;
}

/**
 *作用：对相同的k对不同n 求出将n个不同的元素划分为k个非空集的方案数
 *第二类Stirling数
 *时间复杂度O(nlog(n))
 */
template<int P = ::P>
constexpr static Poly <P> Second_Stirling_Same_K(int Max_n, int k) {
    comb.init(Max_n + 1);
    Poly <P> f(vector<Z>(comb._invfac.begin(), comb._invfac.begin() + Max_n +
1));
    f[0] = 0;
    f = f.pow(k, Max_n + 1);
    for (int i = 0; i <= Max_n; ++i) {
        f[i] = f[i] * comb.fac(i) * comb.invfac(k);
    }
    return f;
}

/**
 *作用：对相同的n对i \in ( 0 , n ) 求出将n个不同的元素划分为i个非空轮换的方案数
 *第一类Stirling数
 *时间复杂度O(nlog(n))
 */
template<int P = ::P>

```

```
constexpr static Poly <P> First_Stirling_Same_N(int n) {
    ll len = __lg(n);
    Poly <P> f = {1};
    ll cnt = 0;
    for (int i = len; i >= 0; --i) {
        f *= Polynomial_translation(f, cnt);
        cnt <= 1;
        if (n >> i & 1) f *= Poly{cnt, 1}, cnt += 1;
    }
    return f;
}

/**
 *作用：对相同的k对不同n 求出将n个不同的元素划分为k个非轮换的方案数
 *第一类Stirling数
 *时间复杂度O(nlog(n))
 */
template<int P = ::P>
constexpr static Poly <P> First_Stirling_Same_K(int Max_n, int k) {
    comb.init(Max_n + 1);
    Poly <P> f(comb._inv.begin(), comb._inv.begin() + Max_n + 1);
    f = f.pow(k, Max_n + 1);
    for (int i = 0; i <= Max_n; ++i) {
        f[i] *= comb.fac(i) * comb.invfac(k);
    }
    return f;
}
};
```

矩阵

```
namespace matrix {
    using i64 = long long;

    template<typename T>
    struct Matrix : public std::vector<std::vector<T>> {
        using std::vector<std::vector<T>>::vector;

        Matrix(int x) : std::vector<std::vector<T>>(x, std::vector<T>(x)) {};
        Matrix(int x, int y) : std::vector<std::vector<T>>(x, std::vector<T>(y))
    };

    Matrix(int x, int y, T c) : std::vector<std::vector<T>>(x, std::vector<T>
(y, c)) {};

    constexpr Matrix operator+(Matrix a);
    constexpr Matrix operator-(Matrix a);
    constexpr Matrix operator*(Matrix a);

    template <typename T1, typename T2>
    friend constexpr Matrix<T1> operator*(Matrix<T1> x, T2 a);

    constexpr Matrix& operator+=(Matrix a);
    constexpr Matrix& operator-=(Matrix a);
    constexpr Matrix& operator*=(Matrix a);
}
```



```

template <typename T1, typename T2>
friend constexpr Matrix<T1>& operator*=(Matrix<T1>& x, T2 a);

constexpr Matrix pow(i64 b);
constexpr Matrix Transpose();
constexpr Matrix inv();
};

template <typename T>
constexpr Matrix<T> Matrix<T>::operator+(Matrix<T> a) {
    auto it = *this;
    int n = (int)a.size();
    int m = (int)a.back().size();
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            it[i][j] += a[i][j];
    return it;
}

template <typename T>
constexpr Matrix<T> Matrix<T>::operator-(Matrix<T> a) {
    auto it = *this;
    int n = (int)a.size();
    int m = (int)a.back().size();
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            it[i][j] -= a[i][j];
    return it;
}

template <typename T>
constexpr Matrix<T> Matrix<T>::operator*(Matrix<T> a) {
    int n = (int)this->size();
    int mid = (int)a.size();
    int m = (int)a.back().size();
    Matrix<T> it(n, m);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            for (int k = 0; k < mid; ++k)
                it[i][j] += (*this)[i][k] * a[k][j];
    return it;
}

template <typename T1, typename T2>
constexpr Matrix<T1> operator*(Matrix<T1> x, T2 a) {
    int n = (int)x.size();
    int m = (int)x.back().size();
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            x[i][j] *= a;
    return x;
}

template <typename T>
constexpr Matrix<T>& Matrix<T>::operator+=(Matrix<T> a) {
    return *this = *this + a;
}

```

```

}

template <typename T>
constexpr Matrix<T>& Matrix<T>::operator--(Matrix<T> a) {
    return *this = *this - a;
}

template <typename T>
constexpr Matrix<T>& Matrix<T>::operator*=(Matrix<T> a) {
    return *this = *this * a;
}

template <typename T1, typename T2>
constexpr Matrix<T1>& operator*=(Matrix<T1>& x, T2 a) {
    return x = x * a;
}

template <typename T>
constexpr Matrix<T> Matrix<T>::pow(int64 b) {
    auto res = Matrix<T>(this->size(), this->size());
    for (int i = 0; i < (int)this->size(); ++i)
        res[i][i] = 1;
    auto a = *this;
    for (; b; b /= 2, a *= a)
        if (b % 2) res *= a;
    return res;
}

template <typename T>
constexpr Matrix<T> Matrix<T>::Transpose() {
    int n = this->back().size(), m = this->size();
    auto it = Matrix<T>(n, m);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            it[i][j] = (*this)[j][i];
    return it;
}

template <typename T>
constexpr Matrix<T> Matrix<T>::inv() {
    int n = this->size();
    Matrix<T> it(n, 2 * n);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            it[i][j] = (*this)[i][j];
    for (int i = 0; i < n; ++i)
        it[i][i + n] = 1;
    for (int i = 0; i < n; ++i) {
        int r = i;
        for (int k = i; k < n; ++k)
            if ((int64)it[k][i]) { r = k; break; }
        if (r != i)
            swap(it[r], it[i]);
        if (!(int64)it[i][i])
            return Matrix<T>();
    }
}

```

```

        T x = (T) 1 / it[i][i];
        for (int k = 0; k < n; ++k) {
            if (k == i)
                continue;
            T t = it[k][i] * x;
            for (int j = i; j < 2 * n; ++j)
                it[k][j] -= t * it[i][j];
        }
        for (int j = 0; j < 2 * n; ++j)
            it[i][j] *= x;
    }
    Matrix<T> ans(n, n);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            ans[i][j] = it[i][j + n];
    return ans;
}
// namespace Matrix
};

```

数学类

base

```

int mul(int a, int b, int P) {
    return 1ll * a * b % P;
}

template<typename T>
T power(T a, i64 b, i64 P) {
    T res = 1;
    for (; b; b >>= 1) {
        if (b & 1) {
            res = 1ll * res * a % P;
        }
        a = 1ll * a * a % P;
    }
    return res;
}

int sum2(int a) {
    return a * (a + 1) * (2 * a + 1) / 6;
}

```

Exgcd

```

/**
 * 算法：扩展欧几里得算法
 * 作用：求解  $ax + by = \gcd(a, b)$ 
 * 返回：gcd, x, y
 */
template<typename T = i64>

```

```

array<T, 3> _Exgcd(T a, T b) {
    T x1 = 1, x2 = 0, x3 = 0, x4 = 1;
    while (b != 0) {
        T c = a / b;
        std::tie(x1, x2, x3, x4, a, b)
            = std::make_tuple(x3, x4, x1 - x3 * c, x2 - x4 * c, b, a - b * c);
    }
    return {a, x1, x2}; //x = x1, y = x2;
}

/**
 * 算法: 扩展欧几里得算法
 * 作用: 求解  $ax + by = res$ 
 * 限制:  $\gcd(a, b) \mid res$ 
 */
template<typename T = i64>
array<T, 3> Exgcd(T a, T b, T res) {
    assert(res % __gcd(a, b) == 0);
    auto [gcd, x, y] = _Exgcd(a, b);
    return {gcd, res / gcd * x, res / gcd * y};
}

/**
 * 算法: 线性同余方程
 * 作用: 求解  $ax \equiv b \pmod{P}$ 
 *         的最小整数解
 * 要求:  $\gcd(a, P) \mid b$ 
 */
template<typename T>
T linearCongruenceEquation(i64 a, i64 b, i64 P) {
    auto [gcd, x, k] = Exgcd<T>(a, P, b);
    T t = P / gcd;
    return (x % t + t) % t;
}

/**
 * 算法: 扩展欧几里得算法求逆元
 * 作用: 求解  $ax \equiv 1 \pmod{n}$  的最小整数解
 * 要求:  $a$  与  $n$  互质
 */
template<typename T>
T inv(i64 a, i64 P) {
    auto [gcd, x, k] = _Exgcd(a, P);
    return (x % P + P) % P;
}

```

中国剩余定理

```

template<typename T = i64>
array<T, 3> _Exgcd(T a, T b) {
    T x1 = 1, x2 = 0, x3 = 0, x4 = 1;
    while (b != 0) {
        T c = a / b;
        std::tie(x1, x2, x3, x4, a, b)
            = std::make_tuple(x3, x4, x1 - x3 * c, x2 - x4 * c, b, a - b * c);
    }
    return {a, x1, x2};
}

```

```

    }
    return {a, x1, x2}; //x = x1, y = x2;
}

template<typename T>
T inv(i64 a, i64 P) {
    auto [gcd, x, k] = _Exgcd(a, P);
    return (x % P + P) % P;
}

/**
 * 算法: 中国剩余定理
 * 作用: 求解一元线性同余方程 (x == a (mod P)) 在模n (所有的模积) 的解
 * 限制: 所有模互质
 */

template<typename T>
T chineseRemainderTheorem(vector<i64> &a, vector<i64> &P) {
    T n = accumulate(P.begin(), P.end(), (T) 1, multiplies<T>()), ans = 0;

    for (int i = 0; i < (i64) a.size(); ++i) {
        T P1 = n / P[i], b;
        b = inv<T>(P1, P[i]);
        ans = (ans + a[i] * P1 * b % n) % n;
    }
    return (ans % n + n) % n;
}

template<typename T = i64>
array<T, 3> Exgcd(T a, T b, T res) {
    assert(res % __gcd(a, b) == 0);
    auto [gcd, x, y] = _Exgcd(a, b);
    return {gcd, res / gcd * x, res / gcd * y};
}

/**
 * 算法: 扩展中国剩余定理
 * 作用: 求解一元线性同余方程 ( x == a ( mod m ) ) 在模n (所有模的最小公倍数) 的解
 * 无限制: 所有模互质
 */

template<typename T>
T extendTheChineseRemainderTheorem(vector<i64> &a, vector<i64> &P) {
    T P1 = P[0], a1 = a[0];
    for (int i = 1; i < a.size(); ++i) {
        T P2 = P[i], a2 = a[i];
        auto [gcd, p, q] = Exgcd(P1, P2, a2 - a1);
        a1 = P1 * p + a1;
        P1 = P1 * P2 / gcd;
        a1 = (a1 % P1 + P1) % P1;
    }
    return a1;
}

```

质因数分解，素数检验

```
i64 mul(i64 a, i64 b, i64 m) {
    return static_cast<__int128>(a) * b % m;
}

i64 power(i64 a, i64 b, i64 m) {
    i64 res = 1 % m;
    for (; b >= 1, a = mul(a, a, m))
        if (b & 1)
            res = mul(res, a, m);
    return res;
}

/**
 * 算法: Miller_Rabin_Test
 * 作用: 在long long范围内快速判断质数
 * 时间复杂度:  $O(\log^3(n))$ 
 */
bool isprime(i64 n) {
    if (n < 2)
        return false;
    static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    int s = __builtin_ctzll(n - 1);
    i64 d = (n - 1) >> s;
    for (auto a : A) {
        if (a == n)
            return true;
        i64 x = power(a, d, n);
        if (x == 1 || x == n - 1)
            continue;
        bool ok = false;
        for (int i = 0; i < s - 1; ++i) {
            x = mul(x, x, n);
            if (x == n - 1) {
                ok = true;
                break;
            }
        }
        if (!ok)
            return false;
    }
    return true;
}

/**
 * 时间复杂度 :  $O(n \wedge (1 / 4))$ 
 */
std::vector<i64> factorize(i64 n) {
    std::vector<i64> p;
    std::function<void(i64)> f = [&](i64 n) {
        if (n <= 10000) {
            for (int i = 2; i * i <= n; ++i)
                for (; n % i == 0; n /= i)
                    p.push_back(i);
            if (n > 1)
                p.push_back(n);
            return;
        }
    };
    f(n);
    return p;
}
```

```

    }
    if (isprime(n)) {
        p.push_back(n);
        return;
    }
    auto g = [&](i64 x) {
        return (mul(x, x, n) + 1) % n;
    };
    i64 x0 = 2;
    while (true) {
        i64 x = x0;
        i64 y = x0;
        i64 d = 1;
        i64 power = 1, lam = 0;
        i64 v = 1;
        while (d == 1) {
            y = g(y);
            ++lam;
            v = mul(v, std::abs(x - y), n);
            if (lam % 127 == 0) {
                d = std::gcd(v, n);
                v = 1;
            }
            if (power == lam) {
                x = y;
                power *= 2;
                lam = 0;
                d = std::gcd(v, n);
                v = 1;
            }
        }
        if (d != n) {
            f(d);
            f(n / d);
            return;
        }
        ++x0;
    }
};
f(n);
std::sort(p.begin(), p.end());
return p;
}

void dfs(int i, auto &b, vector<i64>& ans, i64 c) {
    if (i == b.size()) {
        ans.push_back(c);
        return;
    }
    auto [u, siz] = b[i];
    i64 t = 1;
    for (int j = 0; j <= siz; j += 1) {
        dfs(i + 1, b, ans, c * t);
        t *= u;
    }
}

```

```

/**
 * 时间复杂度 :  $O(\text{siz} * P.\text{size}())$ 
 */
vector<i64> factorize2(i64 n) {
    auto p = factorize(n);
    std::map<i64, int> map;
    for (auto u : p) {
        map[u] ++;
    }
    auto v = vector(map.begin(), map.end());
    vector<i64> ans;
    dfs(0, v, ans, 1);
    sort(ans.begin(), ans.end());
    return ans;
}

```

扩展欧拉定理

```

template<typename T>
T power(T a, i64 b, i64 P) {
    T res = 1;
    for (; b; b >>= 1) {
        if (b & 1) {
            res = 1ll * res * a % P;
        }
        a = 1ll * a * a % P;
    }
    return res;
}

/**
 * 算法: 欧拉函数
 * 作用: 求欧拉函数
 * 时间复杂度:  $O(\sqrt{n})$ 
 */
template<typename T = i64>
T Phi(T n) {
    T ans = n;
    for (i64 i = 2; i * i <= n; i++)
        if (n % i == 0) {
            ans = ans / i * (i - 1);
            while (n % i == 0) n /= i;
        }
    if (n > 1) ans = ans / n * (n - 1);
    return ans;
}

/**
 * 算法: 扩展欧拉定理 (欧拉降幂)
 * 作用: 大指数快速幂
 * 时间复杂度:  $O(\sqrt{m})$ 
 */

i64 exEulertheorem(i64 a, string b, i64 P) {

```



```

i64 gcd = __gcd(a, P);
i64 phi = Phi(P);
i64 res = 0;
bool ok = 0;
for (auto u: b) {
    res = res * 10 + u - '0';
    while (res >= phi) {
        res -= phi;
        if (!ok) ok = 1;
    }
}
if (gcd != 1 && ok) res += phi;
return power(a, res, P);
}

```

扩展卢卡斯定理

```

/**
 * 算法: 扩展lucas
 * 作用: 在p为非质数情况下, 大数组合数C(n,m)
 * 必要情况下, 预处理降低复杂度, 复杂度O(p logp)
 */
using i64 = long long;
i64 mul(i64 a, i64 b, i64 P) {
    return static_cast<__int128>(a) * b % P;
}
i64 power(i64 a, i64 b, i64 P) {
    i64 res = 1 % P;
    for (; b >= 1, a = mul(a, a, P))
        if (b & 1)
            res = mul(res, a, P);
    return res;
}
template<typename T = i64>
constexpr array<T, 3> Exgcd(T a, T b) {
    T x1 = 1, x2 = 0, x3 = 0, x4 = 1;
    while (b != 0) {
        T c = a / b;
        std::tie(x1, x2, x3, x4, a, b) =
            std::make_tuple(x3, x4, x1 - x3 * c, x2 - x4 * c, b, a - b * c);
    }
    return {a, x1, x2}; //x = x1, y = x2;
}
template<typename T = i64>
constexpr array<T, 3> __Exgcd(T a, T b, T res) {
    assert(res % __gcd(a, b) == 0);
    auto [gcd, x, y] = Exgcd(a, b);
    return {gcd, res / gcd * x, res / gcd * y};
}
template<typename T = i64>
constexpr T inv(i64 a, i64 mod) {
    auto [gcd, x, k] = Exgcd<T>((T) a, (T) mod);
    return (x % mod + mod) % mod;
}
template<typename T = i64>

```

```
constexpr T Extend_the_Chinese_remainder_theorem
(vector<i64> &a, vector<i64> &m) {
    T m1 = m[0], a1 = a[0];
    for (int i = 1; i < (i64) a.size(); ++i) {
        T m2 = m[i], a2 = a[i];
        auto [gcd, p, q] = __Exgcd(m1, m2, a2 - a1);
        a1 = m1 * p + a1;
        m1 = m1 * m2 / gcd;
        a1 = (a1 % m1 + m1) % m1;
    }
    return a1;
}

i64 Exlucas(i64 n, i64 m, i64 P) {
    std::vector<i64> p, a;
    function<i64(i64, i64, i64)> calc = [&](i64 n, i64 x, i64 P) mutable -> i64
    {
        if (!n) return 1;
        i64 s = 1;
        for (i64 i = 1; i <= P; ++i) //求阶乘, 可预处理降低复杂度
            if (i % x != 0) s = mul(s, i, P);
        s = power(s, n / P, P);
        for (i64 i = n / P * P + 1; i <= n; ++i)
            if (i % x != 0) s = mul(i, s, P);
        return mul(s, calc(n / x, x, P), P);
    };
    function<i64(i64, i64, i64, i64)> multilucas = [&](i64 n, i64 m, i64 x, i64
P) -> i64 {
        i64 cnt = 0;
        for (i64 i = n; i != 0; i /= x) cnt += i / x;
        for (i64 i = m; i != 0; i /= x) cnt -= i / x;
        for (i64 i = n - m; i != 0; i /= x) cnt -= i / x;
        return static_cast<__int128>(1) * power(x, cnt, P) % P * calc(n, x, P) %
P
            * inv(calc(m, x, P), P) % P * inv(calc(n - m, x, P), P) % P;
    };
    for (i64 i = 2; i * i <= P; ++i) {
        if (P % i == 0) {
            p.emplace_back(1);
            while (P % i == 0) p.back() *= i, P /= i;
            a.emplace_back(multilucas(n, m, i, p.back()));
        }
    }
    if (P > 1) p.emplace_back(P), a.emplace_back(multilucas(n, m, P, P));
    return Extend_the_Chinese_remainder_theorem(a, p);
}
```

扩展大步小步算法

```
/**
 * 算法: 扩展BSGS
 * 作用: 求解  $a^x = b \pmod m$ 
 * 无要求:  $a$ 与 $m$ 互质
 * 返回: 问题的最小非负 $x$ , 无解返回-1
 * 建议使用自定义Hash
```

```

*/
using i64 = long long;
using ui64 = unsigned long long;
constexpr i64 exBSGS(i64 a, i64 b, i64 m, i64 k = 1) {
    constexpr i64 inf = 1e15;
    auto BSGS = [&](i64 a, i64 b, i64 m, i64 k = 1) {
# ifdef _Hash
        unordered_map <ui64, ui64, Hash> map;
# else
        std::map <ui64, ui64> map;
# endif
        i64 cur = 1, t = sqrt(m) + 1;
        for (i64 B = 1; B <= t; ++B) {
            (cur *= a) %= m;
            map[b * cur % m] = B;
        }
        ll now = cur * k % m;
        for (i64 A = 1; A <= t; ++A) {
            auto it = map.find(now);
            if (it != map.end())
                return A * t - (i64) it->second;
            (now *= cur) %= m;
        }
        return -inf; // 无解
    };
    i64 A = a %= m, B = b %= m, M = m;
    if (b == 1) return 0;
    i64 cur = 1 % m;
    for (int i = 0;; i++) {
        if (cur == B) return i;
        cur = cur * A % M;
        i64 d = __gcd(a, m);
        // if (b % d) return -inf;
        if (b % d) return -1;
        if (d == 1) {
            auto ans = BSGS(a, b, m, k * a % m);
            if (ans == -inf) return -1;
            else return ans + i + 1;
        }
        k = k * a / d % m, b /= d, m /= d;
    }
}
}

```

n次剩余

```

/**
 * 算法: n次剩余
 * 作用: 求解  $x^a = b \pmod m$ 
 * 要求: m是质数
 * 返回: x, 无解返回-1e15
 * 建议使用自定义Hash
*/
using i64 = long long;
i64 mul(i64 a, i64 b, i64 m) {

```

```

        return static_cast<__int128>(a) * b % m;
    }
    template<class T = i64>
    constexpr T power(T a, i64 b) {
        T res = 1;
        for (; b; b /= 2, a *= a)
            if (b % 2) res *= a;
        return res;
    }

    i64 power(i64 a, i64 b, i64 m) {
        i64 res = 1 % m;
        for (; b; b >>= 1, a = mul(a, a, m))
            if (b & 1)
                res = mul(res, a, m);
        return res;
    }

    std::vector<i64> n_times_remaining(i64 a, i64 b, i64 m) {
        b %= m;
        vector<array<i64, 3>> fs;
        [&] (i64 m) {
            for (i64 i = 2; i * i <= m; i += 1) {
                if (m % i == 0) {
                    array<i64, 3> f{i, 1, 0};
                    while(m % i == 0) m /= i, f[1] *= i, f[2] += 1;
                    fs.push_back(f);
                }
            }
            if (m > 1) fs.push_back({m, m, 1});
        }(m);
        auto get_Step = [&] (i64 a, i64 n, i64 mod) { //求阶
            i64 ans = n;
            for (i64 i = 2; i * i <= n; i++)
                if (n % i == 0) {
                    while (ans % i == 0 && power(a, ans / i, mod) == 1) ans /= i;
                    for (; n % i == 0; n /= i);
                }
            if (power(a, ans / n, mod) == 1) ans /= n;
            return ans;
        };

        i64 ans = 1;
        auto cntor = [&] (i64 A, i64 B, i64 m, i64 phi) {
            i64 c = get_Step(B, phi, m), y = phi / c, G = __gcd(A, phi);
            if (y % G) ans = 0; ans *= G;
        };
        for (auto [p, pt, t] : fs) {
            if (!ans) break;
            if (b % pt == 0) ans *= power(p, t - (t + a - 1) / a, 1e9);
            else {
                i64 z = 0, b0 = b;
                for (; b0 % p == 0; z++, pt /= p, t--, b0 /= p);
                if (z % a) ans = 0;
                else {
                    cntor(a, b0, pt, pt - pt / p);
                    ans *= power(p, z - z / a, 1e9);
                }
            }
        }
    }

```

```

    }
}
}
return std::vector<i64>{ans};
}

```

原根

```

template<typename T = i64>
T Phi(T n) {
    T ans = n;
    for (i64 i = 2; i * i <= n; i++)
        if (n % i == 0) {
            ans = ans / i * (i - 1);
            while (n % i == 0) n /= i;
        }
    if (n > 1) ans = ans / n * (n - 1);
    return ans;
}

template<typename T>
T power(T a, i64 b, i64 P) {
    T res = 1;
    for (; b >= 1) {
        if (b & 1) {
            res = 1ll * res * a % P;
        }
        a = 1ll * a * a % P;
    }
    return res;
}

i64 min_primitive_root(i64 m) {
    i64 phi = Phi(m);
    auto div = [&](i64 x) {
        vector<i64> f;
        for (i64 i = 2; i * i <= x; ++i) {
            if (x % i != 0) continue;
            f.push_back(i);
            while (x % i == 0) x /= i;
        }
        if (x != 1 && x != phi) f.push_back(x);
        return f;
    };
    auto d = div(phi);
    i64 root = -1;
    auto check = [&](i64 x) {
        for (auto u: d)
            if (power(x, u, m) == 1)
                return false;
        root = x;
        return true;
    };
    for (i64 i = 1;; ++i) {

```

```

        if (__gcd(i, m) != 1)
            continue;
        if (check(i)) break;
    }
    return root;
}

```

原根2

```

struct Sieves {
    int n;
    vector<int> Prime, Euler, Morbius, Approximate, Approximate_cnt;
    vector<bool> notprime;
    vector<array<i64, 2>> div;

    Sieves() {};

    Sieves(int _n) { init(_n); };

    void init(int _n) {
        n = _n;
        Prime_work();
    }

    void Prime_work() {
        notprime.assign(n + 1, 0);
        notprime[0] = 1;
        notprime[1] = 1;
        for (i64 i = 2; i <= n; ++i) {
            if (notprime[i] == 0) {
                Prime.push_back(i);
            }
            for (i64 j = 0; i * Prime[j] <= n; ++j) {
                notprime[i * Prime[j]] = 1;

                if (i % Prime[j] == 0) break;
            }
        }
    }

    void Euler_work() {
        Euler.assign(n + 1, 0);
        Euler[1] = 1;
        for (i64 i = 2; i <= n; ++i) {
            if (notprime[i] == 0) Euler[i] = i - 1;
            for (i64 j = 0; i * Prime[j] <= n; ++j) {
                i64 now = i * Prime[j];
                if (i % Prime[j] != 0) {
                    Euler[now] = (Prime[j] - 1) * Euler[i];
                } else {
                    Euler[now] = Prime[j] * Euler[i];
                    break;
                }
            }
        }
    }
}

```

```

}

void Morbius_work() {
    Morbius.assign(n + 1, 0);
    Morbius[1] = 1;
    for (i64 i = 2; i <= n; ++i) {
        if (notprime[i] == 0) Morbius[i] = -1;
        for (i64 j = 0; i * Prime[j] <= n; ++j) {
            i64 now = i * Prime[j];
            if (i % Prime[j] != 0) {
                Morbius[now] = -Morbius[i];
            } else break;
        }
    }
}

void Div_work() {
    div.resize(n + 1);
    div[0] = {1, 1};
    div[1] = {1, 1};
    for (i64 i = 2; i <= n; ++i) {
        if (notprime[i] == 0) {
            div[i] = {1, i};
        }
        for (i64 j = 0; i * Prime[j] <= n; ++j) {
            div[i * Prime[j]] = {i, Prime[j]};
            if (i % Prime[j] == 0) break;
        }
    }
}

/**
 * 求约数个数
 */
void Approximate_work() {
    Approximate.assign(n + 1, 0);
    Approximate_cnt.assign(n + 1, 0);
    Approximate[1] = 1;
    Approximate_cnt[1] = 0;
    for (i64 i = 2; i <= n; ++i) {
        if (notprime[i] == 0) {
            Approximate[i] = 2;
            Approximate_cnt[i] = 1;
        }
        for (i64 j = 0; i * Prime[j] <= n; ++j) {
            i64 now = i * Prime[j];
            if (i % Prime[j] != 0) {
                Approximate_cnt[now] = 1;
                Approximate[now] = Approximate[i] * 2;
            } else {
                Approximate_cnt[now] = Approximate_cnt[i] + 1;
                Approximate[now] = Approximate[i] / Approximate_cnt[now] *
(Approximate_cnt[now] + 1);
                break;
            }
        }
    }
}

```

```

    }
}

std::vector<i64> get_frac(i64 x) {
    vector<i64> f;
    for (; x > 1; f.push_back(div[x][0]), x = div[x][1]);
    return f;
}

i64 size() { return (i64) Prime.size(); }

bool isprime(int n) { return !notprime[n]; }

i64 eu(int n) { return Euler[n]; }

i64 mo(int n) { return Morbius[n]; }
};

template<typename T>
T power(T a, i64 b, i64 P) {
    T res = 1;
    for (; b; b >>= 1) {
        if (b & 1) {
            res = 1ll * res * a % P;
        }
        a = 1ll * a * a % P;
    }
    return res;
}

/**
 * 求一个数的所有原根
 * 时间复杂度:  $O(\sqrt{m})$ 
 */
vector<i64> primitive_root(i64 n) {
    Sieves s(n);
    s.Euler_work();
    vector<bool> exist(n + 1);
    exist[2] = 1;
    exist[4] = 1;
    for (i64 p : s.Prime) {
        if ((p & 1) == 0) continue;
        for (i64 now = p; now < exist.size(); now *= p) {
            exist[now] = 1;
            if (now * 2 < exist.size())
                exist[now * 2] = 1;
        }
    }
    if (!exist[n]) return vector<i64>();
    vector<i64> f;
    i64 phi = s.eu(n);
    i64 pphi = s.eu(phi);
    i64 m = phi;
    for (int i = 2; i * i <= m; ++i) {
        if (m % i == 0) {
            f.push_back(i);

```



```

        while (m % i)
            m /= i;
    }
}
if (m != 1) f.push_back(m);
i64 root = -1;
auto check = [&](i64 x) {
    for (auto u: f)
        if (power(x, phi / u, n) == 1)
            return false;
    root = x;
    return true;
};
for (i64 i = 1;; ++i) {
    if (__gcd(i, n) != 1) continue;
    if (check(i)) break;
}
vector<i64> ans;
for (i64 now = root, i = 1; i <= phi; ++i) {
    if (__gcd(phi, i) == 1)
        ans.push_back(now);
    now = (now * root) % n;
}
sort(ans.begin(), ans.end());
return ans;
}

```

旧版参考

```

/**
 * 数学工具箱
 */

namespace Math {
    using i64 = long long;
    using Int = __int128;
    using ui64 = unsigned long long;
    std::mt19937
    rng(std::chrono::system_clock::now().time_since_epoch().count());

    struct math {

/**
 * @brief 带模乘
 * @return (a ^ b)% m
 */
        i64 static mul(i64 a, i64 b, i64 m);

/**
 * @brief 快速幂
 */
        template<class T>

```

```

constexpr static T power(T a, i64 b);

i64 static power(i64 a, i64 b, i64 m);

/**
 * @brief 求和
 */
template<typename T>
constexpr static T __sum1(T it);

template<typename T>
constexpr static T __sum2(T it);

/**
 * 欧几里得算法相关
 */

/**
 * 算法: 扩展欧几里得算法
 * 作用: 求解  $ax + by = \gcd(a, b)$ 
 * 返回:  $\gcd, x, y$ 
 */
template<typename T = i64>
constexpr array<T, 3> static Exgcd(T a, T b);

/**
 * 算法: 扩展欧几里得算法
 * 作用: 求解  $ax + by = \text{res}$ 
 * 限制:  $\gcd(a, b) \mid \text{res}$ 
 */
template<typename T = i64>
constexpr array<T, 3> static __Exgcd(T a, T b, T res);

/**
 * 算法: 线性同余方程
 * 作用: 求解  $ax \equiv b \pmod{n}$ 
 * 的最小整数解
 * 要求:  $\gcd(a, n) \mid b$ 
 */
template<typename T = i64>
constexpr T static Linear_congruence_equation(i64 a, i64 b, i64 mod);

/**
 * 算法: 扩展欧几里得算法求逆元
 * 作用: 求解  $ax \equiv 1 \pmod{n}$  的最小整数解
 * 要求:  $a$  与  $n$  互质
 */
template<typename T = i64>
constexpr T static inv(i64 a, i64 mod);

/**
 * 扩展欧几里得结束
 */

/**
 * 算法: Miller_Rabin_Test

```

```

* 作用：在long long范围内快速判断质数
* 时间复杂度：O(log^3(n))
*/

constexpr static bool Miller_Rabin_Test(i64 n);

/**
* 算法：Pollard_Rho
* 作用：能快速找到大整数的一个非1、非自身的因子的算法
* 时间复杂度：O(n^{1/4}log(n))
*/

static i64 Pollard_Rho(i64 N);

/**
* 算法：使用Pollard_Rho进行质因数分解
* 返回：顺序所有质因子(重复)
*/

std::vector<i64> static factorize(i64 n);

/**
* 算法：中国剩余定理
* 作用：求解一元线性同余方程 ( x == a ( mod m ) ) 在模n(所有的模积)的解
* 限制：所有模互质
*/

template<typename T = i64>
constexpr static T Chinese_remainder_theorem
(vector<i64> &a, vector<i64> &m);

/**
* 算法：扩展中国剩余定理
* 作用：求解一元线性同余方程 ( x == a ( mod m ) ) 在模n(所有模的最小公倍数)的解
* 无限制：所有模互质
*/

template<typename T = i64>
constexpr static T Extend_the_Chinese_remainder_theorem
(vector<i64> &a, vector<i64> &m);

/**
* 算法：欧拉函数
* 作用：求欧拉函数
* 时间复杂度：O(sqrt ( n ))
*/

template<typename T = i64>
constexpr static T Euler_phi(T n);

/**
* 算法：扩展欧拉定理(欧拉降幂)
* 作用：大指数快速幂
* 时间复杂度：O(sqrt ( m ))
*/

static i64 Extending_Euler_theorem(i64 a, string b, i64 m);

/**
* 算法：求最小原根
* 要求：请自行保证这个数有原根(2,4,p^q,2*p^q)
* 时间复杂度：O(sqrt(n))

```

```

*/
    static i64 min_primitive_root(i64 m);

/**
 * 求一个数的所有原根
 * 注意提前使用质数筛，名称为s，开到n，并筛出欧拉函数
 * 需要Linear_sieves_max、s
 * 时间复杂度:  $O(\sqrt{m})$ 
 */
#ifdef _Linear_sieves
    std::vector<i64> static primitive_root(i64 n);
#endif

/**
 * 算法: 扩展BSGS
 * 作用: 求解  $a^x = b \pmod{m}$ 
 * 无要求: a与m互质
 * 返回: 问题的最小非负x, 无解返回-1
 * 建议使用自定义Hash
 */
constexpr i64 static exBSGS(i64 a, i64 b, i64 m, i64 k = 1);

/**
 * 算法: n次剩余
 * 作用: 求解  $x^a = b \pmod{m}$ 
 * 要求: m是质数
 * 返回: x, 无解返回-1e15
 * 建议使用自定义Hash
 */
static std::vector<i64> n_times_remaining(i64 a, i64 b, i64 m);

/**
 * 算法: 扩展lucas
 * 作用: 在p为非质数情况下, 大数组合数C(n,m)
 * 必要情况下, 预处理降低复杂度
 */

static i64 Exlucas(i64 n, i64 m, i64 P);

//struct math
};

i64 math::mul(i64 a, i64 b, i64 m) {
    return static_cast<__int128>(a) * b % m;
}

template<class T>
constexpr T math::power(T a, i64 b) {
    T res = 1;
    for (; b; b /= 2, a *= a)
        if (b % 2) res *= a;
    return res;
}

```

```

}

i64 math::power(i64 a, i64 b, i64 m) {
    i64 res = 1 % m;
    for (; b; b >>= 1, a = mul(a, a, m))
        if (b & 1)
            res = mul(res, a, m);
    return res;
}

template<typename T>
constexpr T math::__sum1(T it) { return (it * (it + 1)) / ((T) 2); }

template<typename T>
constexpr T math::__sum2(T it) { return it * (it + 1) * (2 * it + 1) / ((T)
6); }

template<typename T>
constexpr array<T, 3> math::Exgcd(T a, T b) {
    T x1 = 1, x2 = 0, x3 = 0, x4 = 1;
    while (b != 0) {
        T c = a / b;
        std::tie(x1, x2, x3, x4, a, b) =
            std::make_tuple(x3, x4, x1 - x3 * c, x2 - x4 * c, b, a - b *
c);
    }
    return {a, x1, x2}; //x = x1, y = x2;
}

template<typename T>
constexpr array<T, 3> math::__Exgcd(T a, T b, T res) {
    assert(res % __gcd(a, b) == 0);
    auto [gcd, x, y] = Exgcd(a, b);
    return {gcd, res / gcd * x, res / gcd * y};
}

template<typename T>
constexpr T math::Linear_congruence_equation(i64 a, i64 b, i64 mod) {
    auto [gcd, x, k] = __Exgcd<T>((T) a, (T) mod, (T) b);
    T t = mod / gcd;
    return (x % t + t) % t;
}

template<typename T>
constexpr T math::inv(i64 a, i64 mod) {
    auto [gcd, x, k] = Exgcd<T>((T) a, (T) mod);
    return (x % mod + mod) % mod;
}

constexpr bool math::Miller_Rabin_Test(i64 n) {
    if (n < 3 || n % 2 == 0) return n == 2; //特判
    i64 u = n - 1, t = 0;
    while (u % 2 == 0) u /= 2, ++t;

```

```

constexpr std::array<i64, 7> ud = {2, 325, 9375, 28178, 450775, 9780504,
1795265022};
for (i64 a: ud) {
    i64 v = power(a, u, n);
    if (v == 1 || v == n - 1 || v == 0) continue;
    for (int j = 1; j <= t; j++) {
        v = mul(v, v, n);
        if (v == n - 1 && j != t) {
            v = 1;
            break;
        } // 出现一个n-1, 后面都是1, 直接跳出
        if (v == 1) return 0; // 这里代表前面没有出现n-1这个解, 二次检验失败
    }
    if (v != 1) return 0; // Fermat检验
}
return 1;
}

i64 math::Pollard_Rho(i64 N) {
    if (N == 4) // 特判4
        return 2;
    if (Miller_Rabin_Test(N)) // 特判质数
        return N;
    auto randint = [&](i64 l, i64 r) {
        return l + rng() % (r - l + 1);
    };
    while (true) {
        i64 c = randint(1, N - 1); // 生成随机的c
        auto f = [=](i64 x) { return ((Int) x * x + c) % N; }; // Int表示
__int128, 防溢出
        i64 t = f(0), r = f(f(0));
        while (t != r) {
            i64 d = gcd(abs(t - r), N);
            if (d > 1)
                return d;
            t = f(t), r = f(f(r));
        }
    }
}

std::vector<i64> math::factorize(i64 n) {
    std::vector<i64> p;
    std::function<void(i64)> f = [&](i64 n) {
        if (n <= 10000) {
            for (int i = 2; i * i <= n; ++i)
                for (; n % i == 0; n /= i)
                    p.push_back(i);
            if (n > 1)
                p.push_back(n);
            return;
        }
        if (Miller_Rabin_Test(n)) {
            p.push_back(n);
            return;
        }
        auto g = [&](i64 x) {

```

```

        return (mul(x, x, n) + 1) % n;
    };
    i64 x0 = 2;
    while (true) {
        i64 x = x0;
        i64 y = x0;
        i64 d = 1;
        i64 power = 1, lam = 0;
        i64 v = 1;
        while (d == 1) {
            y = g(y);
            ++lam;
            v = mul(v, std::abs(x - y), n);
            if (lam % 127 == 0) {
                d = std::gcd(v, n);
                v = 1;
            }
            if (power == lam) {
                x = y;
                power *= 2;
                lam = 0;
                d = std::gcd(v, n);
                v = 1;
            }
        }
        if (d != n) {
            f(d);
            f(n / d);
            return;
        }
        ++x0;
    }
};
f(n);
std::sort(p.begin(), p.end());
return p;
}

template<typename T>
constexpr T math::Chinese_remainder_theorem
    (vector<i64> &a, vector<i64> &m) {
    T n = accumulate(m.begin(), m.end(), (T) 1, multiplies<T>()), ans = 0;

    for (int i = 0; i < (i64) a.size(); ++i) {
        T m1 = n / m[i], b;
        b = inv(m1, m[i]);
        ans = (ans + a[i] * m1 * b % n) % n;
    }
    return (ans % n + n) % n;
}

template<typename T>
constexpr T math::Extend_the_Chinese_remainder_theorem
    (vector<i64> &a, vector<i64> &m) {
    T m1 = m[0], a1 = a[0];
    for (int i = 1; i < (i64) a.size(); ++i) {

```

```

        T m2 = m[i], a2 = a[i];
        auto [gcd, p, q] = __Exgcd(m1, m2, a2 - a1);
        a1 = m1 * p + a1;
        m1 = m1 * m2 / gcd;
        a1 = (a1 % m1 + m1) % m1;
    }
    return a1;
}

template<typename T>
constexpr T math::Euler_phi(T n) {
    T ans = n;
    for (i64 i = 2; i * i <= n; i++)
        if (n % i == 0) {
            ans = ans / i * (i - 1);
            while (n % i == 0) n /= i;
        }
    if (n > 1) ans = ans / n * (n - 1);
    return ans;
}

i64 math::Extending_Euler_theorem(i64 a, string b, i64 m) {
    i64 gcd = __gcd(a, m);
    i64 phi = Euler_phi(m);
    i64 res = 0;
    bool flag = 0;
    for (auto u: b) {
        res = res * 10 + u - '0';
        while (res >= phi) {
            res -= phi;
            if (!flag) flag = 1;
        }
    }
    if (gcd != 1 && flag) res += phi;
    return power(a, res, m);
}

i64 math::min_primitive_root(i64 m) {
    i64 phi = math::Euler_phi(m);
    auto div = [&](i64 x) {
        vector<i64> f;
        for (i64 i = 2; i * i <= x; ++i) {
            if (x % i != 0) continue;
            f.push_back(i);
            while (x % i == 0) x /= i;
        }
        if (x != 1 && x != phi) f.push_back(x);
        return f;
    };
    auto d = div(phi);
    i64 root = -1;
    auto check = [&](i64 x) {
        for (auto u: d)
            if (math::power(x, u, m) == 1)
                return false;
        return true;
    };
    for (i64 x = 2; x < m; ++x)
        if (check(x))
            return x;
    return -1;
}

```



```

        return true;
    };
    for (i64 i = 1;; ++i) {
        if (__gcd(i, m) != 1)
            continue;
        if (check(i)) break;
    }
    return root;
}

#ifdef _Linear_sieves
std::vector<i64> math::primitive_root(i64 n) {
    static vector<bool> exist(Linear_sieves_max + 1);
    auto __exist = [&]() {
        static bool __existed = 0;
        if (__existed) return;
        __existed = 1;
        exist[2] = 1;
        exist[4] = 1;
        for (ll p: s.Prime) {
            if ((p & 1) == 0) continue;
            for (ll now = p; now <= (ll) exist.size() - 1; now *= p) {
                exist[now] = 1;
                if (now * 2 <= (ll) exist.size() - 1)
                    exist[now * 2] = 1;
            }
        }
    };
    __exist();
    if (!exist[n]) return vector<i64>();
    vector<ll> f;
    ll phi = s.eu(n);
    ll pphi = s.eu(phi);
    ll m = phi;
    for (int i = 2; i * i <= m; ++i) {
        if (m % i == 0) {
            f.push_back(i);
            while (m % i)
                m /= i;
        }
    }
    if (m != 1) f.push_back(m);
    // Debug ( f ) ;
    ll root = -1;
    auto check = [&](ll x) {
        for (auto u: f)
            if (power(x, phi / u, n) == 1)
                return false;
        root = x;
        return true;
    };
    for (i64 i = 1;; ++i) {
        if (__gcd(i, n) != 1) continue;
        if (check(i)) break;
    }
    vector<ll> ans;

```

```

        for (i64 now = root, i = 1; i <= phi; ++i) {
            if (__gcd(phi, i) == 1)
                ans.push_back(now);
            now = (now * root) % n;
        }
        sort(ans.begin(), ans.end());
        return ans;
    }

# endif

constexpr i64 math::exBSGS(i64 a, i64 b, i64 m, i64 k) {
    constexpr i64 inf = 1e15;
    auto BSGS = [&](i64 a, i64 b, i64 m, i64 k = 1) {
# ifdef _Hash
        unordered_map <ui64, ui64, Hash> map;
# else
        std::map <ui64, ui64> map;
# endif

        i64 cur = 1, t = sqrt(m) + 1;
        for (i64 B = 1; B <= t; ++B) {
            (cur *= a) %= m;
            map[b * cur % m] = B;
        }
        ll now = cur * k % m;
        for (i64 A = 1; A <= t; ++A) {
            auto it = map.find(now);
            if (it != map.end())
                return A * t - (i64) it->second;
            (now *= cur) %= m;
        }
        return -inf; // 无解
    };

    i64 A = a %= m, B = b %= m, M = m;
    if (b == 1) return 0;
    i64 cur = 1 % m;
    for (int i = 0;; i++) {
        if (cur == B) return i;
        cur = cur * A % M;
        i64 d = __gcd(a, m);
        if (b % d) return -inf;
        if (d == 1) {
            auto ans = BSGS(a, b, m, k * a % m);
            if (ans == -inf) return -1;
            else return ans + i + 1;
        }
        k = k * a / d % m, b /= d, m /= d;
    }
}

std::vector <i64> math::n_times_remaining(i64 a, i64 b, i64 m) {
    auto root = min_primitive_root(m);
    i64 now = math::power(root, a, m);
    i64 c = math::exBSGS(now, b, m);
    if (c == -1) return vector<i64>();
    i64 x0 = math::power(root, c, m);

```

```

i64 phi = math::Euler_phi(m);
i64 gcd = __gcd(a, phi);
vector<i64> ans;
i64 cnt = math::power(root, phi / gcd, m);
for (int i = 0; i < gcd; ++i) {
    ans.push_back(x0);
    x0 = math::mul(x0, cnt, m);
}
return ans;
}

i64 math::Exlucas(i64 n, i64 m, i64 P) {
    std::vector<i64> p, a;
    function<i64(i64, i64, i64)> calc = [&](i64 n, i64 x, i64 P) mutable ->
i64 {
        if (!n) return 1;
        i64 s = 1;
        for (i64 i = 1; i <= P; ++i) //求阶乘, 可预处理降低复杂度
            if (i % x != 0) s = math::mul(s, i, P);
        s = math::power(s, n / P, P);
        for (i64 i = n / P * P + 1; i <= n; ++i)
            if (i % x != 0) s = math::mul(i, s, P);
        return math::mul(s, calc(n / x, x, P), P);
    };
    function<i64(i64, i64, i64, i64)> multilucas = [&](i64 n, i64 m, i64 x,
i64 P) -> i64 {
        i64 cnt = 0;
        for (i64 i = n; i != 0; i /= x) cnt += i / x;
        for (i64 i = m; i != 0; i /= x) cnt -= i / x;
        for (i64 i = n - m; i != 0; i /= x) cnt -= i / x;
        return static_cast<__int128>(1) * math::power(x, cnt, P) % P *
calc(n, x, P) % P
            * math::inv(calc(m, x, P), P) % P * math::inv(calc(n - m, x,
P), P) % P;
    };
    for (i64 i = 2; i * i <= P; ++i) {
        if (P % i == 0) {
            p.emplace_back(1);
            while (P % i == 0) p.back() *= i, P /= i;
            a.emplace_back(multilucas(n, m, i, p.back()));
        }
    }
    if (P > 1) p.emplace_back(P), a.emplace_back(multilucas(n, m, P, P));
    return math::Extend_the_Chinese_remainder_theorem(a, p);
}
// namespace Math
}

using namespace Math;

```

线性基

```
struct Linear_Base {
    int siz;
    vector<int> a;
    Linear_Base(int _siz = 61) {
        siz = _siz;
        a.resize(siz + 1);
    }
    void insert(int x) { //插入
        for (int i = siz; i >= 0; i--) if (x & (1ll << i)) {
            if (!a[i]) { a[i] = x; return; }
            else x ^= a[i];
        }
    }
    bool check(int x) { //查询x是否能被异或出来
        for (int i = siz; i >= 0; i--) if (x & (1ll << i)) {
            if (!a[i]) break;
            x ^= a[i];
        }
        return x == 0;
    }
    int querymax(int res) { //查询最大异或和
        for (int i = siz; i >= 0; i--) if ((res ^ a[i]) > res) res ^= a[i];
        return res;
    }
    int querymin(int res) { //查询最小
        for (int i = siz; i >= 0; i--) if (res & (1ll << i)) res ^= a[i];
        return res;
    }
    int querykth(int k) { //查询第k大的异或和
        vector<int> tmp(siz + 10);
        int res = 0, cnt = 0;
        for (int i = 0; i <= siz; i++) {
            for (int j = i - 1; j >= 0; j--) if (a[i] & (1ll << j)) a[i] ^= a[j];
            if(a[i]) tmp[cnt++] = a[i];
        }
        for (int i = 0; i < cnt; i++) if (k & (1ll << i)) res ^= tmp[i];
        return res;
    }
    void merge(const Linear_Base& other) //合并
    {
        for (int i = 0; i <= siz; i++) insert(other.a[i]);
    }
};
```

线性筛

```
struct Linear_sieves {
    # define _Linear_sieves
    int n;
    vector<int> Prime, Euler, Morbius, Approximate, Approximate_cnt;
    vector<bool> notprime;
```

```

vector<array<i64, 2>> div;

Linear_sieves() {};

Linear_sieves(int _n) { init(_n); };

void init(int _n) {
    n = _n;
    Prime_work();
}

void Prime_work() {
    notprime.assign(n + 1, 0);
    notprime[0] = 1;
    notprime[1] = 1;
    for (i64 i = 2; i <= n; ++i) {
        if (notprime[i] == 0) {
            Prime.push_back(i);
        }
        for (i64 j = 0; i * Prime[j] <= n; ++j) {
            notprime[i * Prime[j]] = 1;

            if (i % Prime[j] == 0) break;
        }
    }
}

void Euler_work() {
    Euler.assign(n + 1, 0);
    Euler[1] = 1;
    for (i64 i = 2; i <= n; ++i) {
        if (notprime[i] == 0) Euler[i] = i - 1;
        for (i64 j = 0; i * Prime[j] <= n; ++j) {
            i64 now = i * Prime[j];
            if (i % Prime[j] != 0) {
                Euler[now] = (Prime[j] - 1) * Euler[i];
            } else {
                Euler[now] = Prime[j] * Euler[i];
                break;
            }
        }
    }
}

void Morbius_work() {
    Morbius.assign(n + 1, 0);
    Morbius[1] = 1;
    for (i64 i = 2; i <= n; ++i) {
        if (notprime[i] == 0) Morbius[i] = -1;
        for (i64 j = 0; i * Prime[j] <= n; ++j) {
            i64 now = i * Prime[j];
            if (i % Prime[j] != 0) {
                Morbius[now] = -Morbius[i];
            } else break;
        }
    }
}

```

```

}

void Div_work() {
    div.resize(n + 1);
    div[0] = {1, 1};
    div[1] = {1, 1};
    for (i64 i = 2; i <= n; ++i) {
        if (notprime[i] == 0) {
            div[i] = {1, i};
        }
        for (i64 j = 0; i * Prime[j] <= n; ++j) {
            div[i * Prime[j]] = {Prime[j], i};
            if (i % Prime[j] == 0) break;
        }
    }
}

/**
 * 求约数个数
 */
void Approximate_work() {
    Approximate.assign(n + 1, 0);
    Approximate_cnt.assign(n + 1, 0);
    Approximate[1] = 1;
    Approximate_cnt[1] = 0;
    for (i64 i = 2; i <= n; ++i) {
        if (notprime[i] == 0) {
            Approximate[i] = 2;
            Approximate_cnt[i] = 1;
        }
        for (i64 j = 0; i * Prime[j] <= n; ++j) {
            i64 now = i * Prime[j];
            if (i % Prime[j] != 0) {
                Approximate_cnt[now] = 1;
                Approximate[now] = Approximate[i] * 2;
            } else {
                Approximate_cnt[now] = Approximate_cnt[i] + 1;
                Approximate[now] = Approximate[i] / Approximate_cnt[now] *
(Approximate_cnt[now] + 1);
                break;
            }
        }
    }
}

std::vector<i64> get_frac(i64 x) {
    vector<i64> f;
    for (; x > 1; f.push_back(div[x][0]), x = div[x][1]);
    return f;
}

i64 size() { return (i64) Prime.size(); }

bool isprime(int n) { return !notprime[n]; }

i64 eu(int n) { return Euler[n]; }

```

```
i64 mo(int n) { return Morbius[n]; }  
};
```

组合数学

```
template<class T>  
struct Comb {  
    int n;  
    std::vector<T> _fac;  
    std::vector<T> _invfac;  
    std::vector<T> _inv;  
  
    Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}  
  
    Comb(int n) : Comb() {  
        init(n);  
    }  
  
    void init(int m) {  
        m = std::min(m, T::getMod() - 1);  
        if (m <= n) return;  
        _fac.resize(m + 1);  
        _invfac.resize(m + 1);  
        _inv.resize(m + 1);  
  
        for (int i = n + 1; i <= m; i++) {  
            _fac[i] = _fac[i - 1] * i;  
        }  
        _invfac[m] = _fac[m].inv();  
        for (int i = m; i > n; i--) {  
            _invfac[i - 1] = _invfac[i] * i;  
            _inv[i] = _invfac[i] * _fac[i - 1];  
        }  
        n = m;  
    }  
  
    T fac(int m) {  
        if (m > n) init(2 * m);  
        return _fac[m];  
    }  
  
    T invfac(int m) {  
        if (m > n) init(2 * m);  
        return _invfac[m];  
    }  
  
    T inv(int m) {  
        if (m > n) init(2 * m);  
        return _inv[m];  
    }  
  
    T binom(int n, int m) {  
        if (n < m || m < 0) return 0;  
        return fac(n) * invfac(m) * invfac(n - m);  
    }  
};
```

```

    }

/**
 * 第二类斯特林数
 * 时间复杂度 :  $O(m * \log(m))$ 
 */
T Stirling(int n, int m) {
    T ans = 0;
    for (int i = 0; i <= m; ++i) {
        ans += (((m - i) & 1) == 1 ? -1 : 1) * power((T) i, n) * invfac(i) *
invfac(m - i);
    }
    return ans;
}

T Catalan(int n) {
    return binom(2 * n, n) * inv(n + 1);
}

/**
 * 算法: 卢卡斯定理
 * 作用: 大数组合数
 * 注意在p较小时使用p
 * p为Z的质数
 * 时间复杂度为 $O(\log p)$ 
 */
T lucas(i64 n, i64 m) {
    if (m == 0) return T(1);
    return binom(n % T::getMod(), m % T::getMod()) * lucas(n / T::getMod(), m
/ T::getMod());
}
};

Comb<Z> comb;

```

行列式

```

using i64 = long long;
// 时间复杂度  $O(n^3 + n^2 * \log p)$ 
// 行列式 mod p, a[1, n][1, n]
constexpr int calcDet(vector<vector<int>> &a, int n, const int p) {
    i64 zf = 1, ans = 1, tmp = 0;

    for(int i = 1; i <= n; ++i)
        for(int j = 1; j <= n; ++j)
            a[i][j] %= p;

    for (int i = 1; i <= n; i++) {
        int k = i;
        for (int j = i + 1; j <= n; j++)
            if (a[j][i] > a[k][i]) {
                k = j;
            }
        if (!a[k][i]) return 0;
    }
}

```



```

        if (k != i) swap(a[i], a[k]), zf = -zf;

        for (int j = i + 1; j <= n; j++) {
            if (a[j][i] > a[i][i]) swap(a[i], a[j]), zf = -zf;
            while (a[j][i]) {
                tmp = a[i][i] / a[j][i];
                for (int k = i; k <= n; k++)
                    a[i][k] = (a[i][k] + a[j][k] * (p - tmp) % p) % p;
                swap(a[i], a[j]), zf = -zf;
            }
        }
        ans = ans * a[i][i] % p;
    }

    if (zf == -1) ans = (-ans + p) % p;
    return ans;
}

// 时间复杂度 O(n^3)
// 行列式 a[0, n)[0, n)
constexpr double calcDet(vector<vector<double>> &a, int n, const double eps = 1e-9) {
    double det = 1;
    for (int i = 0; i < n; ++i) {
        int k = i;
        for (int j = i + 1; j < n; ++j)
            if (abs(a[j][i]) > abs(a[k][i])) k = j;
        if (abs(a[k][i]) < eps) {
            det = 0;
            break;
        }
        swap(a[i], a[k]);
        if (i != k) det = -det;
        det *= a[i][i];
        for (int j = i + 1; j < n; ++j) a[i][j] /= a[i][i];
        for (int j = 0; j < n; ++j)
            if (j != i && abs(a[j][i]) > eps)
                for (int k = i + 1; k < n; ++k) a[j][k] -= a[i][k] * a[j][i];
    }
    return det;
}

```

高斯消元

```

// 时间复杂度: O(m * n^2), -1无解, 0唯一解, 否则无穷解
// a为增广矩阵 行r:[0, n) 列c:[0, m], a[i][m]为b[0, n), 求解答案为 x[0, m)
int Gauss(vector<vector<double>> &a, vector<double> &x, int n, int m, double eps = 1e-7){
    int r = 0, c = 0;
    for(r = 0; r < n && c < m; r++, c++) {
        int maxr = r;
        for(int i = r + 1; i < n; i++) {
            if(abs(a[i][c]) > abs(a[maxr][c]))
                maxr = i;
        }
    }
}

```

```

    }
    if(maxr != r) std::swap(a[r], a[maxr]);
    if(fabs(a[r][c]) < eps) {
        r--;
        continue;
    }
    for(int i = r + 1; i < n; i++) {
        if(fabs(a[i][c]) > eps){
            double k = a[i][c] / a[r][c];
            for(int j = c; j < m + 1; j++) a[i][j] -= a[r][j] * k;
            a[i][c] = 0;
        }
    }
}
for(int i = r; i < m; i++) {
    if(fabs(a[i][c]) > eps) return -1;//无解
}
if(r < m) return m - r;//返回自由元个数
for(int i = m-1; i >= 0; i--) {
    for(int j = i + 1; j < m; j++) a[i][m] -= a[i][j] * x[j];
    x[i] = a[i][m] / a[i][i];
}
return 0;//有唯一解
}

```

图论

SCC

一般

```

struct SCC {
    int n, cnt = 0, cur = 0;
    vector<int> h, t, to;
    vector<int> dfn, low, stk, f;
    vector<bool> in;

    SCC(int n): n(n), h(n, -1), dfn(n, -1), low(n), f(n, -1), in(n, 0) {}

    void tarjan(int u) {
        dfn[u] = low[u] = cur++;
        stk.push_back(u);
        in[u] = 1;
        for (int i = h[u]; i != -1; i = t[i]) {
            int v = to[i];
            if (dfn[v] == -1) {
                tarjan(v);
                low[u] = min(low[u], low[v]);
            } else if (in[v]) {
                low[u] = min(low[u], dfn[v]);
            }
        }
    }
}

```

```

    }
    if (dfn[u] == low[u]) {
        int t = cnt ++;
        do {
            in[stk.back()] = 0;
            f[stk.back()] = t;
            stk.pop_back();
        } while (in[u]);
    }
}

void addEdge(int u, int v) {
    t.push_back(h[u]), h[u] = to.size(), to.push_back(v);
}

vector<vector<int>> Graph() {
    vector<vector<int>> g(cnt);
    for (int u = 0; u < n; u += 1) {
        for (int i = h[u]; i != -1; i = t[i]) {
            int v = to[i];
            if (f[u] != f[v]) {
                g[f[u]].push_back(f[v]);
            }
        }
    }
    return g;
}

vector<array<int, 2>> Edges() {
    vector<array<int, 2>> e;
    for (int u = 0; u < n; u += 1) {
        for (int i = h[u]; i != -1; i = t[i]) {
            int v = to[i];
            if (f[u] != f[v]) {
                e.push_back({f[u], f[v]});
            }
        }
    }
    sort(e.begin(), e.end());
    e.erase(unique(e.begin(), e.end()), e.end());
    return e;
}
};

```

割边

```

struct CutEdge {
    int n, tot = -1;
    vector<array<int, 2>> e;
    vector<int> h, t;
    vector<int> dfn, low, ans;

    CutEdge(int n) :n(n), dfn(n, -1), low(n, -1), h(n, -1) {};

    // in 表示从哪条边下来, 如果为根, by为一个极大值

```

```

void tarjan(int u, int in = 1E9) {
    dfn[u] = low[u] = ++tot;
    for (int i = h[u]; i != -1; i = t[i]) {
        int v = e[i][1];
        if (i == (in ^ 1)) {
            continue;
        }
        if (dfn[v] == -1) {
            tarjan(v, i);
            low[u] = min(low[u], low[v]);
            if (low[v] > dfn[u]) {
                ans.push_back(i);
            }
        } else {
            low[u] = min(low[u], dfn[v]);
        }
    }
}

void addEdge(int u, int v) {
    t.push_back(h[u]), h[u] = e.size(); e.push_back({u, v});
    t.push_back(h[v]), h[v] = e.size(); e.push_back({v, u});
}

vector<int> &answer() {
    for (int i = 0; i < n; i += 1) {
        if (dfn[i] == -1) {
            tarjan(i);
        }
    }
    return ans;
}
};

```

边双

```

struct EDCC {
    int n, cur = -1, cnt = 0;
    vector<array<int, 2>> e;
    vector<int> h, t;
    vector<int> dfn, low, ans, stk, f;

    EDCC(int n) : n(n), dfn(n, -1), low(n, -1), h(n, -1), f(n, -1) {};

    // in 表示从哪条边下来, 如果为根, by为一个极大值
    void tarjan(int u, int in = 1E9) {
        dfn[u] = low[u] = ++cur;
        stk.push_back(u);
        for (int i = h[u]; i != -1; i = t[i]) {
            int v = e[i][1];
            if (i == (in ^ 1)) {
                continue;
            }
            if (dfn[v] == -1) {

```

```

        tarjan(v, i);
        low[u] = min(low[u], low[v]);
        if (low[v] > dfn[u]) {
            ans.push_back(i);
        }
    } else {
        low[u] = min(low[u], dfn[v]);
    }
}
if (dfn[u] == low[u]) {
    int t = cnt ++;
    int v;
    do {
        v = stk.back();
        stk.pop_back();
        f[v] = t;
    } while (v != u);
}
}

void addEdge(int u, int v) {
    t.push_back(h[u]), h[u] = e.size(); e.push_back({u, v});
    t.push_back(h[v]), h[v] = e.size(); e.push_back({v, u});
}

vector<int> &answer() {
    for (int i = 0; i < n; i += 1) {
        if (dfn[i] == -1) {
            tarjan(i);
        }
    }
    return ans;
}
};

```

割点

```

struct CutPoint {
    int n, cur = 0;
    vector<int> h, t, to, dfn, low;
    vector<bool> cut;

    CutPoint(int n): n(n), h(n, -1), dfn(n, -1), low(n, -1), cut(n) {};

    // 传参时 tarjan(u, u)
    void tarjan(int u, int r) {
        dfn[u] = low[u] = cur ++;
        int child = 0;
        for (int i = h[u]; i != -1; i = t[i]) {
            int v = to[i];
            if (dfn[v] == -1) {
                tarjan(v, r);
                low[u] = min(low[u], low[v]);
            }
        }
    }
};

```

```

        if (low[v] >= dfn[u]) {
            ++child;
            if (u != r || child > 1) {
                cut[u] = true;
            }
        }
    } else {
        low[u] = min(low[u], dfn[v]);
    }
}

void addEdge(int u, int v) {
    t.push_back(h[u]), h[u] = to.size(); to.push_back(v);
    t.push_back(h[v]), h[v] = to.size(); to.push_back(u);
}

};

```

点双

```

struct VDCC {
    int n, cur = 0, cnt = 0;
    vector<int> h, t, to, dfn, low, stk;
    vector<bool> cut;
    vector<vector<int>> sorted;

    VDCC(int n): n(n), h(n, -1), dfn(n, -1), low(n, -1), cut(n) {};

    // 传参时 tarjan(u, u)
    void tarjan(int u, int r) {
        dfn[u] = low[u] = cur++;
        if (h[u] == -1) {
            debug(u);
            int t = cnt++;
            sorted.push_back({u});
            return;
        }
        stk.push_back(u);
        int child = 0;
        for (int i = h[u]; i != -1; i = t[i]) {
            int v = to[i];
            if (dfn[v] == -1) {
                tarjan(v, r);
                low[u] = min(low[u], low[v]);
                if (low[v] >= dfn[u]) {
                    ++child;
                    if (u != r || child > 1) {
                        cut[u] = true;
                    }
                }
                int t = cnt++;
                sorted.emplace_back();
                int x;
                do {
                    int x = stk.back();
                    stk.pop_back();

```

```

        sorted[t].push_back(x);
    } while (x != v);
    sorted[t].push_back(u);
}
} else {
    low[u] = min(low[u], dfn[v]);
}
}
}

void addEdge(int u, int v) {
    t.push_back(h[u]), h[u] = to.size(); to.push_back(v);
    t.push_back(h[v]), h[v] = to.size(); to.push_back(u);
}
};

```

倍增lca

```

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

// static unweighted tree
template<int N>
struct multiTree {
    int n;
    int K;
    vector<array<int, __lg(N) + 1>> p;
    struct Edge {
        int v; i64 w;
    };
};
vector<pair<int, int>> e;
csr<int> g;
vector<int> dep;
multiTree(int n): n(n), dep(n), p(n), K(__lg(n) + 1) {}
void addEdge(int u, int v) {
    e.push_back({u, v});
    e.push_back({v, u});
}

```

```

void work(int r = 0) {
    g = csr<int>(n, e);
    dfs(r, r);
}

void dfs(int u, int f) {
    p[u][0] = f;
    for (int i = 1; i < K; i += 1) {
        p[u][i] = p[p[u][i - 1]][i - 1];
    }
    for (int i = g.s[u]; i < g.s[u + 1]; i += 1) {
        auto v = g.adj[i];
        if (v == f) {
            continue;
        }
        dep[v] = dep[u] + 1;
        dfs(v, u);
    }
}

int jump(int u, int d) {
    for (int i = dep[u] - d; i > 0; i -= 1 << __lg(i)) {
        u = p[u][__lg(i)];
    }
    return u;
}

int lca(int u, int v) {
    if (dep[u] < dep[v]) {
        swap(u, v);
    }
    u = jump(u, dep[v]);
    if (u == v) {
        return u;
    }
    for (int i = __lg(dep[u]); i >= 0; i -= 1) {
        if (p[u][i] != p[v][i]) {
            u = p[u][i], v = p[v][i];
        }
    }
    return p[u][0];
}

int dis1(int u, int v) {
    int m = lca(u, v);
    return dep[u] + dep[v] - 2 * dep[m];
}
};

```

带权

```

template <class E>
struct csr {
    vector<int> s;
    vector<E> adj;
    csr() {}
    csr(int n, const vector<pair<int, E>>& g)
        : s(n + 1), adj(g.size()) {
        for (auto e : g) {

```



```

        s[e.first + 1]++;
    }
    for (int i = 1; i <= n; i++) {
        s[i] += s[i - 1];
    }
    auto c = s;
    for (auto e : g) {
        adj[c[e.first]++] = e.second;
    }
}

};

// static weighted tree
template<int N = int(1E6)>
struct multiTree {
    int n;
    int K;
    vector<array<int, __lg(N) + 1>> p;
    struct Edge {
        int v; i64 w;
        friend ostream &operator<<(ostream &cout, Edge u) {
            return cout << "(" << u.v << ", " << u.w << ")";
        }
    };
};

vector<pair<int, Edge>> e;
csr<Edge> g;
vector<int> dep;
vector<i64> dep2;
multiTree() {}
multiTree(int n) {
    init(n);
}

void init(int n) {
    this->n = n;
    K = __lg(n) + 1;
    p.assign(n, {});
    dep.assign(n, 0);
    dep2.assign(n, 0);
}

void addEdge(int u, int v, auto w) {
    e.push_back({u, {v, w}});
    e.push_back({v, {u, w}});
}

void work(int r = 0) {
    g = csr<Edge>(n, e);
    dfs(r, r);
}

void dfs(int u, int f) {
    p[u][0] = f;
    for (int i = 1; i < K; i += 1) {
        p[u][i] = p[p[u][i - 1]][i - 1];
    }
    for (int i = g.s[u]; i < g.s[u + 1]; i += 1) {
        auto [v, w] = g.adj[i];
        if (v == f) {
            continue;
        }
    }
}

```

```

    }
    dep[v] = dep[u] + 1;
    dep2[v] = dep2[u] + w;
    dfs(v, u);
}
}
int jump(int u, int d) {
    for (int i = dep[u] - d; i > 0; i -= 1 << __lg(i)) {
        u = p[u][__lg(i)];
    }
    return u;
}
int lca(int u, int v) {
    if (dep[u] < dep[v]) {
        swap(u, v);
    }
    u = jump(u, dep[v]);
    if (u == v) {
        return u;
    }
    for (int i = __lg(dep[u]); i >= 0; i -= 1) {
        if (p[u][i] != p[v][i]) {
            u = p[u][i], v = p[v][i];
        }
    }
    return p[u][0];
}
int dist(int u, int v) {
    int m = lca(u, v);
    return dep[u] + dep[v] - 2 * dep[m];
}
int dis2(int u, int v) {
    int m = lca(u, v);
    return dep2[u] + dep2[v] - 2 * dep2[m];
}
};

```

虚树

```

auto merge = [&] (int x, int y) {
    return std::max(x, y);
};
struct Tree {
    int n, cur;
    vector<vector<int>> adj;
    vector<int> dfn, dep, siz, fa, seq, dnp;
    SparseTable<int, decltype(merge)> rmq;
    Tree() = default;
    Tree(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
    }
};

```

```

        adj.assign(n, {});
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void work(int root = 0) {
        dfn.assign(n, 0);
        dep.assign(n, 0);
        siz.assign(n, 1);
        fa.assign(n, 0);
        seq.assign(n, 0);
        dnp.assign(n, 0);
        cur = 0;
        dfs(root, root);
        rmq.init(dnp, ::merge);
    }
    void dfs(int now, int f) {
        dnp[cur] = dfn[f];
        seq[dfn[now] = cur++] = now;
        dep[now] = dep[f] + 1;
        fa[now] = f;
        for (auto here : adj[now]) {
            if (here == f) continue;
            dfs(here, now);
            siz[now] += siz[here];
        }
    }
    int lca (int lhs, int rhs) {
        if (lhs == rhs) return lhs;
        if ((lhs = dfn[lhs]) > (rhs = dfn[rhs])) swap(lhs, rhs);
        return seq[rmq(lhs + 1, rhs + 1)];
    }
};

constexpr int maxTop = 3e5;
int stk[maxTop];
struct virTree : public Tree {
    virTree(int n) : vTree(n) {}
    vector<vector<int>> vTree;
    vector<vector<int>> &build_virtual_tree(vector<int> &key) {
        sort(key.begin(), key.end(), [&] (int x, int y) {return dfn[x] <
dfn[y];});
        int Top = 0;
        vTree[0].clear();
        for (int i = key[0] == 0; i < key.size(); i += 1) {
            int Lca = lca(key[i], stk[Top]);
            if (Lca != stk[Top]) {
                while(dfn[Lca] < dfn[stk[Top - 1]]) {
                    vTree[stk[Top - 1]].push_back(stk[Top --]);
                }
                if (Lca != stk[Top - 1]) {
                    vTree[Lca].clear();
                    vTree[Lca].push_back(stk[Top]); stk[Top] = Lca;
                } else {
                    vTree[Lca].push_back(stk[Top --]);
                }
            }
        }
    }
};

```

```

        }
    }
    vTree[key[i]].clear();
    stk[++ Top] = key[i];
}
for (int i = 0; i < Top; i += 1) {
    vTree[stk[i]].push_back(stk[i + 1]);
}
return vTree;
}
};

```

重链剖分

```

struct HLD {
    int n;
    std::vector<int> siz, top, dep, parent, in, out, seq;
    std::vector<std::vector<int>> adj;
    int cur;

    HLD() {}
    HLD(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        siz.resize(n);
        top.resize(n);
        dep.resize(n);
        parent.resize(n);
        in.resize(n);
        out.resize(n);
        seq.resize(n);
        cur = 0;
        adj.assign(n, {});
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void work(int root = 0) {
        top[root] = root;
        dep[root] = 0;
        parent[root] = -1;
        dfs1(root);
        dfs2(root);
    }
    void dfs1(int u) {
        if (parent[u] != -1) {
            adj[u].erase(std::find(adj[u].begin(), adj[u].end(), parent[u]));
        }

        siz[u] = 1;
        for (auto &v : adj[u]) {
            parent[v] = u;

```

```

        dep[v] = dep[u] + 1;
        dfs1(v);
        siz[u] += siz[v];
        if (siz[v] > siz[adj[u][0]]) {
            std::swap(v, adj[u][0]);
        }
    }
}

void dfs2(int u) {
    in[u] = cur++;
    seq[in[u]] = u;
    for (auto v : adj[u]) {
        top[v] = v == adj[u][0] ? top[u] : v;
        dfs2(v);
    }
    out[u] = cur;
}

int lca(int u, int v) {
    while (top[u] != top[v]) {
        if (dep[top[u]] > dep[top[v]]) {
            u = parent[top[u]];
        } else {
            v = parent[top[v]];
        }
    }
    return dep[u] < dep[v] ? u : v;
}

int dist(int u, int v) {
    return dep[u] + dep[v] - 2 * dep[lca(u, v)];
}

int jump(int u, int k) {
    if (dep[u] < k) {
        return -1;
    }

    int d = dep[u] - k;

    while (dep[top[u]] > d) {
        u = parent[top[u]];
    }

    return seq[in[u] - dep[u] + d];
}

bool isAncestor(int u, int v) {
    return in[u] <= in[v] && in[v] < out[u];
}

int rootedParent(int u, int v) {
    std::swap(u, v);
    if (u == v) {
        return u;
    }
    if (!isAncestor(u, v)) {

```

```

        return parent[u];
    }
    auto it = std::upper_bound(adj[u].begin(), adj[u].end(), v, [&](int x,
int y) {
        return in[x] < in[y];
    }) - 1;
    return *it;
}

int rootedSize(int u, int v) {
    if (u == v) {
        return n;
    }
    if (!isAncestor(v, u)) {
        return siz[v];
    }
    return n - siz[rootedParent(u, v)];
}

int rootedLca(int a, int b, int c) {
    return lca(a, b) ^ lca(b, c) ^ lca(c, a);
}
};

```

重链剖分套线段树

```

template<typename Info, typename Tag>
struct HLD_Seg : public HLD, LazySegmentTree<Info, Tag> {
    using LazySegmentTree<Info, Tag>::rangeApply, LazySegmentTree<Info,
Tag>::rangeQuery;
    HLD_Seg(int n) {
        init(n);
    }
    void init(int n) {
        HLD::init(n);
    }
    void work(int root, const vector<Info> &a) {
        HLD::work(root);
        vector<Info> b(n);
        for (int i = 0; i < n; i += 1) {
            b[in[i]] = a[i];
        }
        LazySegmentTree<Info, Tag>::init(b);
    }
    void LineApply(int u, int v, Tag t) {
        while (top[u] != top[v]) {
            if (dep[top[u]] < dep[top[v]]) {
                swap(u, v);
            }
            rangeApply(in[top[u]], in[u] + 1, t);
            u = parent[top[u]];
        }
        if (in[u] > in[v]) {
            swap(u, v);
        }
    }
};

```

```

    }
    rangeApply(in[u], in[v] + 1, t);
}
Info LineQuery(int u, int v) {
    Info ans = Info();
    while (top[u] != top[v]) {
        if (dep[top[u]] < dep[top[v]]) {
            swap(u, v);
        }
        ans = Info::merge(ans, rangeQuery(in[top[u]], in[u] + 1));
        u = parent[top[u]];
    }
    if (in[u] > in[v]) {
        swap(u, v);
    }
    ans = Info::merge(ans, rangeQuery(in[u], in[v] + 1));
    return ans;
}
void SubApply(int u, Tag t, int r = 0) {
    if (u == r) {
        rangeApply(0, n, t);
    } else if (isAncestor(u, r)) {
        if (top[u] == top[r]) {
            r = seq[in[u] + 1];
        } else {
            while (top[parent[top[r]]] != top[u]) {
                r = parent[top[r]];
            }
            r = top[r];
            if (parent[r] != u) {
                r = seq[in[u] + 1];
            }
        }
        rangeApply(0, in[r], t);
        rangeApply(out[r], n, t);
    } else {
        rangeApply(in[u], out[u], t);
    }
}
Info SubQuery(int u, int r = 0) {
    Info ans = Info();
    if (u == r) {
        return ans = rangeQuery(0, n);
    } else if (isAncestor(u, r)) {
        if (top[u] == top[r]) {
            r = seq[in[u] + 1];
        } else {
            while (top[parent[top[r]]] != top[u]) {
                r = parent[top[r]];
            }
            r = top[r];
            if (parent[r] != u) {
                r = seq[in[u] + 1];
            }
        }
    }
    ans = Info::merge(rangeQuery(0, in[r]), rangeQuery(out[r], n));
}

```

```

    } else {
        return ans = rangeQuery(in[u], out[u]);
    }
    return ans;
}
};

```

流

网络流

```

constexpr int inf = 1E9;
template<class T>
struct MaxFlow {
    struct _Edge {
        int to;
        T cap;
        _Edge(int to, T cap) : to(to), cap(cap) {}
    };

    int n;
    std::vector<_Edge> e;
    std::vector<std::vector<int>>> g;
    std::vector<int> cur, h;

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

    void init(int n) {
        this->n = n;
        e.clear();
        g.assign(n, {});
        cur.resize(n);
        h.resize(n);
    }

    bool bfs(int s, int t) {
        h.assign(n, -1);
        std::queue<int> que;
        h[s] = 0;
        que.push(s);
        while (!que.empty()) {
            const int u = que.front();
            que.pop();
            for (int i : g[u]) {
                auto [v, c] = e[i];
                if (c > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    if (v == t) {
                        return true;
                    }
                    que.push(v);
                }
            }
        }
    }
};

```



```

    }
    }
    }
    return false;
}

T dfs(int u, int t, T f) {
    if (u == t) {
        return f;
    }
    auto r = f;
    for (int &i = cur[u]; i < int(g[u].size()); ++i) {
        const int j = g[u][i];
        auto [v, c] = e[j];
        if (c > 0 && h[v] == h[u] + 1) {
            auto a = dfs(v, t, std::min(r, c));
            e[j].cap -= a;
            e[j ^ 1].cap += a;
            r -= a;
            if (r == 0) {
                return f;
            }
        }
    }
    return f - r;
}

void addEdge(int u, int v, T c) {
    g[u].push_back(e.size());
    e.emplace_back(v, c);
    g[v].push_back(e.size());
    e.emplace_back(u, 0);
}

T flow(int s, int t) {
    T ans = 0;
    while (bfs(s, t)) {
        cur.assign(n, 0);
        ans += dfs(s, t, std::numeric_limits<T>::max());
    }
    return ans;
}

std::vector<bool> minCut() {
    std::vector<bool> c(n);
    for (int i = 0; i < n; i++) {
        c[i] = (h[i] != -1);
    }
    return c;
}

struct Edge {
    int from;
    int to;
    T cap;
    T flow;
};

std::vector<Edge> edges() {

```

```

std::vector<Edge> a;
for (int i = 0; i < e.size(); i += 2) {
    Edge x;
    x.from = e[i + 1].to;
    x.to = e[i].to;
    x.cap = e[i].cap + e[i + 1].cap;
    x.flow = e[i + 1].cap;
    a.push_back(x);
}
return a;
}
};

```

网络流前向星

```

template<class T>
struct MaxFlow {
    int n;
    vector<int> r, t, to, h, cur;
    vector<T> c;
    MaxFlow(int n, int m = 0) {
        init(n, m);
    }
    void init(int n, int m = 0) {
        this->n = n;
        r.assign(n, -1);
        h.assign(n, -1);
        cur.assign(n, 0);
        t.reserve(2 * m);
        to.reserve(2 * m);
        c.reserve(2 * m);
    }
    void addEdge(int u, int v, T cap) {
        t.push_back(r[u]), r[u] = to.size(), to.push_back(v), c.push_back(cap);
        t.push_back(r[v]), r[v] = to.size(), to.push_back(u), c.push_back(0);
    }
    bool bfs(int s, int e) {
        fill(h.begin(), h.end(), -1);
        queue<int> q;
        h[s] = 0;
        cur[s] = r[s];
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int i = r[u]; ~i; i = t[i]) {
                int v = to[i];
                T cap = c[i];
                if (cap > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    cur[v] = r[v];
                    if (v == e) {
                        return true;
                    }
                }
            }
        }
        return false;
    }
};

```

```

        }
        q.push(v);
    }
}

return false;
}

T dfs(int u, int e, T f) {
    if (u == e) {
        return f;
    }
    T r = f;
    for (int &i = cur[u]; ~i; i = t[i]) {
        int v = to[i];
        T cap = c[i];
        if (cap > 0 && h[v] == h[u] + 1) {
            T k = dfs(v, e, min(cap, r));
            if (k == 0) {
                h[v] = -1;
            }
            c[i] -= k;
            c[i ^ 1] += k;
            r -= k;
            if (r == 0) {
                return f;
            }
        }
    }
    return f - r;
}

T flow(int s, int e) {
    T ans = 0;
    while (bfs(s, e)) {
        ans += dfs(s, e, std::numeric_limits<T>::max());
    }
    return ans;
}

std::vector<bool> minCut() {
    std::vector<bool> c(n);
    for (int i = 0; i < n; i++) {
        c[i] = (h[i] != -1);
    }
    return c;
}

struct Edge {
    int from;
    int to;
    T cap;
    T flow;
    friend ostream &operator<<(ostream &cout, Edge u) {
        return cout << '{' << u.from << ", " << u.to << ", " << u.cap << ", "
<< u.flow << "}";
    }
};

```

```

vector<Edge> edges() {
    vector<Edge> a;
    for (int i = 0; i < t.size(); i += 2) {
        Edge x;
        x.from = to[i + 1];
        x.to = to[i];
        x.cap = c[i] + c[i + 1];
        x.flow = c[i + 1];
        a.push_back(x);
    }
    return a;
}
};

```

网络流未封装

```

using T = int;

constexpr int N = 2e5 + 2, M = 2 * N + 1e5;
int head[N], nxt[2 * M], to[2 * M];
T cap[2 * M];
int cur = 0;
int _n = 0;
int h[N], now[N];

void init(int n) {
    fill(head, head + n, -1);
    _n = n;
    cur = -1;
}

void addEdge(int u, int v, T c) {
    nxt[++cur] = head[u], to[cur] = v, cap[cur] = c, head[u] = cur;
    nxt[++cur] = head[v], to[cur] = u, cap[cur] = 0, head[v] = cur;
}

bool bfs(int s, int t) {
    fill(h, h + _n, -1);
    queue<int> q;
    h[s] = 0;
    now[s] = head[s];
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int i = head[u]; ~i; i = nxt[i]) {
            int v = to[i];
            T c = cap[i];
            if (c > 0 && h[v] == -1) {
                h[v] = h[u] + 1;
                now[v] = head[v];
                if (v == t) {
                    return true;
                }
            }
        }
    }
}

```

```

        q.push(v);
    }
}
return false;
}

T dfs(int u, int t, T f) {
    if (u == t) {
        return f;
    }
    T r = f;
    for (int &i = now[u]; ~i; i = nxt[i]) {
        int v = to[i];
        T c = cap[i];
        if (c > 0 && h[v] == h[u] + 1) {
            T k = dfs(v, t, min(c, r));
            if (k == 0) {
                h[v] = -1;
            }
            cap[i] -= k;
            cap[i ^ 1] += k;
            r -= k;
            if (r == 0) {
                return f;
            }
        }
    }
    return f - r;
}

T flow(int s, int t) {
    T ans = 0;
    while (bfs(s, t)) {
        ans += dfs(s, t, std::numeric_limits<T>::max());
    }
    return ans;
}

struct Edge {
    int from;
    int to;
    T cap;
    T flow;
};

vector<bool> minCut() {
    vector<bool> c(_n);
    for (int i = 0; i < _n; i++) {
        c[i] = (h[i] != -1);
    }
    return c;
}

vector<Edge> Edges() {
    vector<Edge> a;

```

```

    for (int i = 0; i < cur; i += 2) {
        Edge x;
        x.from = to[i + 1];
        x.to = to[i];
        x.cap = cap[i] + cap[i + 1];
        x.flow = cap[i + 1];
        a.push_back(x);
    }
    return a;
}

```

费用流

```

struct MCFGraph {
    struct Edge {
        int v, c, f;
        Edge(int v, int c, int f) : v(v), c(c), f(f) {}
    };
    const int n;
    std::vector<Edge> e;
    std::vector<std::vector<int>>> g;
    std::vector<i64> h, dis;
    std::vector<int> pre;
    bool dijkstra(int s, int t) {
        dis.assign(n, std::numeric_limits<i64>::max());
        pre.assign(n, -1);
        std::priority_queue<std::pair<i64, int>, std::vector<std::pair<i64,
int>>, std::greater<std::pair<i64, int>>>> que;
        dis[s] = 0;
        que.emplace(0, s);
        while (!que.empty()) {
            i64 d = que.top().first;
            int u = que.top().second;
            que.pop();
            if (dis[u] < d) continue;
            for (int i : g[u]) {
                int v = e[i].v;
                int c = e[i].c;
                int f = e[i].f;
                if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
                    dis[v] = d + h[u] - h[v] + f;
                    pre[v] = i;
                    que.emplace(dis[v], v);
                }
            }
        }
        return dis[t] != std::numeric_limits<i64>::max();
    }
    MCFGraph(int n) : n(n), g(n) {}
    void addEdge(int u, int v, int c, int f) {
        g[u].push_back(e.size());
        e.emplace_back(v, c, f);
        g[v].push_back(e.size());
    }
}

```

```

        e.emplace_back(u, 0, -f);
    }
    std::pair<int, i64> flow(int s, int t) {
        int flow = 0;
        i64 cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t)) {
            for (int i = 0; i < n; ++i) h[i] += dis[i];
            int aug = std::numeric_limits<int>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug = std::min(aug,
e[pre[i]].c);
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
                e[pre[i]].c -= aug;
                e[pre[i] ^ 1].c += aug;
            }
            flow += aug;
            cost += i64(aug) * h[t];
        }
        return std::make_pair(flow, cost);
    }
};

```

费用流前向星

```

template<typename T>
using min_heap = priority_queue<T, vector<T>, greater<T>>;

struct MCFGraph {
    struct Edge {
        int v, c, f;
        Edge(int v, int c, int f) : v(v), c(c), f(f) {}
        template<class ostream>
        friend ostream& operator<<(ostream& cout, Edge e) {
            return cout << "{" << e.v << ", " << e.c << ", " << e.f << "}";
        }
    };
    int n;
    vector<Edge> e;
    vector<int> r, t, pre;
    vector<i64> h, dis;
    bool dijkstra(int S, int T) {
        dis.assign(n, numeric_limits<i64>::max());
        pre.assign(n, -1);
        min_heap<pair<i64, int>> q;
        dis[S] = 0;
        q.emplace(0, S);
        while (!q.empty()) {
            i64 d = q.top().first;
            int u = q.top().second;
            q.pop();
            if (dis[u] < d) continue;
            for (int i = r[u]; ~i; i = t[i]) {
                int v = e[i].v;
                int c = e[i].c;

```

```

        int f = e[i].f;
        if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
            dis[v] = d + h[u] - h[v] + f;
            pre[v] = i;
            q.emplace(dis[v], v);
        }
    }
}

return dis[T] != numeric_limits<i64>::max();
}

MCFGGraph(int n, int m = 0) : n(n), r(n, -1) {
    t.reserve(2 * m), e.reserve(2 * m);
}

void addEdge(int u, int v, int c, int f) {
    // cerr << u << ' ' << v << ' ' << c << '-' << f << '\n';
    t.push_back(r[u]), r[u] = e.size(), e.emplace_back(v, c, f);
    t.push_back(r[v]), r[v] = e.size(), e.emplace_back(u, 0, -f);
}

pair<int, i64> flow(int s, int t) {
    int flow = 0;
    i64 cost = 0;
    h.assign(n, 0);
    while (dijkstra(s, t)) {
        for (int i = 0; i < n; ++i) h[i] += dis[i];
        int aug = numeric_limits<int>::max();
        for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug = min(aug,
e[pre[i]].c);
        for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
            e[pre[i]].c -= aug;
            e[pre[i] ^ 1].c += aug;
        }
        flow += aug;
        cost += i64(aug) * h[t];
    }
    return make_pair(flow, cost);
}

struct _Edge {
    int from;
    int to;
    int cap;
    int flow;
    int cost;
};

std::vector<_Edge> edges() {
    std::vector<_Edge> a;
    for (int i = 0; i < e.size(); i += 2) {
        _Edge x;
        x.from = e[i + 1].v;
        x.to = e[i].v;
        x.cap = e[i].c + e[i + 1].c;
        x.flow = e[i + 1].c;
        x.cost = e[i].f;
        a.push_back(x);
    }
}

```



```

        return a;
    }
};

pair<bool, i64> MCFF(vector<array<int, 5>> &e, int n) {
    int N = n + 2;
    int s = N - 2, t = s + 1;
    vector<int> d(n);
    MCFGGraph g(N);
    for (auto [u, v, L, U, c] : e) {
        g.addEdge(u, v, U - L, c);
        d[u] -= L;
        d[v] += L;
    }
    for (int i = 0; i < n; i += 1) {
        if (d[i] > 0) {
            g.addEdge(s, i, d[i], 0);
        } else {
            g.addEdge(i, t, -d[i], 0);
        }
    }
    auto [flow, cost] = g.flow(s, t);
    bool ok = 1;
    for (int i = g.r[s]; ~i; i = g.t[i]) {
        ok &= g.e[i].c == 0;
    }
    for (int i = g.r[t]; ~i; i = g.t[i]) {
        ok &= g.e[i ^ 1].c == 0;
    }
    return {ok, cost};
}

```

费用流多类型EK

```

template<typename T>
using min_heap = priority_queue<T, vector<T>, greater<T>>;

template<typename T>
struct Ceil {
    constexpr static T max() {
        return numeric_limits<T>::max();
    }
};

using f64 = double;
template<>
struct Ceil<f64> {
    constexpr static f64 max() {
        return 1e9;
    }
};

template<typename Cap, typename Cost>

```

```

struct MCFGraph {
    struct Edge {
        int v; Cap c; Cost f;
        Edge(int v, Cap c, Cost f) : v(v), c(c), f(f) {}
        template<class ostream>
        friend ostream& operator<<(ostream& cout, Edge e) {
            return cout << "{" << e.v << ", " << e.c << ", " << e.f << "}";
        }
    };
    int n;
    vector<Edge> e;
    vector<int> r, t, pre;
    vector<Cost> h, dis;
    bool dijkstra(int S, int T) {
        dis.assign(n, Ceil<Cost>::max());
        pre.assign(n, -1);
        min_heap<pair<Cost, Cap>> q;
        dis[S] = 0;
        q.emplace(0, S);
        while (!q.empty()) {
            Cost d = q.top().first;
            Cap u = q.top().second;
            q.pop();
            if (dis[u] < d) continue;
            for (int i = r[u]; ~i; i = t[i]) {
                int v = e[i].v;
                Cap c = e[i].c;
                Cost f = e[i].f;
                if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
                    dis[v] = d + h[u] - h[v] + f;
                    pre[v] = i;
                    q.emplace(dis[v], v);
                }
            }
        }
        return dis[T] != Ceil<Cost>::max();
    }
    MCFGraph(int n, int m = 0) : n(n), r(n, -1) {
        t.reserve(2 * m), e.reserve(2 * m);
    }
    void addEdge(int u, int v, Cap c, Cost f) {
        // cerr << u << ' ' << v << ' ' << c << '-' << f << '\n';
        t.push_back(r[u]), r[u] = e.size(), e.emplace_back(v, c, f);
        t.push_back(r[v]), r[v] = e.size(), e.emplace_back(u, 0, -f);
    }
    pair<Cap, Cost> flow(int s, int t) {
        Cap flow = 0;
        Cost cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t)) {
            for (int i = 0; i < n; ++i) h[i] += dis[i];
            Cap aug = Ceil<Cap>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug = min(aug,
e[pre[i]].c);
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
                e[pre[i]].c -= aug;
            }
        }
    }
};

```

```

        e[pre[i] ^ 1].c += aug;
    }
    flow += aug;
    cost += aug * h[t];
}
return make_pair(flow, cost);
}

struct _Edge {
    int from;
    int to;
    Cap cap;
    Cap flow;
    Cost cost;
};

std::vector<_Edge> edges() {
    std::vector<_Edge> a;
    for (int i = 0; i < e.size(); i += 2) {
        _Edge x;
        x.from = e[i + 1].v;
        x.to = e[i].v;
        x.cap = e[i].c + e[i + 1].c;
        x.flow = e[i + 1].c;
        x.cost = e[i].f;
        a.push_back(x);
    }
    return a;
}
};

```

费用流原始对偶

```

template<class E>
struct csr {
    vector<int> h;
    vector<E> e;

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

struct MCFGraph {
    MCFGraph() {}

    MCFGraph(int n) : n(n) {}

```

```

void addEdge(int u, int to, int c, i64 p) {
    E.push_back({u, to, c, 0, p});
}

struct Edge {
    int u, v;
    int c, f;
    i64 p;
};

vector<Edge> Edges() { return E; }
static constexpr int inf = numeric_limits<int>::max();
using Ans = pair<int, i64>;

Ans flow(int s, int t, int f = inf) {
    return slope(s, t, f).back();
}

vector<Ans> slope(int s, int t, int f = inf) {
    int m = E.size();
    vector<int> id(m);

    auto g = [&]() {
        vector<int> d(n), rid(m);
        vector<pair<int, _Edge>> elist;
        elist.reserve(2 * m);
        for (int i = 0; i < m; i++) {
            auto e = E[i];
            id[i] = d[e.u]++;
            rid[i] = d[e.v]++;
            elist.push_back({e.u, {e.v, -1, e.c - e.f, e.p}});
            elist.push_back({e.v, {e.u, -1, e.f, -e.p}});
        }
        auto g = csr<_Edge>(n, elist);
        for (int i = 0; i < m; i++) {
            auto e = E[i];
            id[i] += g.h[e.u];
            rid[i] += g.h[e.v];
            g.e[id[i]].rev = rid[i];
            g.e[rid[i]].rev = id[i];
        }
        return g;
    }();

    auto ans = slope(g, s, t, f);

    for (int i = 0; i < m; i++) {
        E[i].f = E[i].c - g.e[id[i]].c;
    }

    return ans;
}

int n;
vector<Edge> E;

```

```

struct _Edge {
    int v, rev;
    int c;
    i64 p;
};

vector<Ans> slope(csr<_Edge> &g, int s, int t, int f) {
    vector<array<i64, 2>> d(n);
    vector<int> pre(n), qm;
    vector<bool> vis(n);
    vector<pair<i64, int>> q;
    auto cmp = greater<pair<i64, int>>();
    auto ref = [&]() {
        for (int i = 0; i < n; i++) {
            d[i][1] = numeric_limits<i64>::max();
        }
        fill(vis.begin(), vis.end(), false);
        qm.clear();
        q.clear();

        int r = 0;

        d[s][1] = 0;
        qm.push_back(s);
        while (!qm.empty() || !q.empty()) {
            int v;
            if (!qm.empty()) {
                v = qm.back();
                qm.pop_back();
            } else {
                while (r < q.size()) {
                    r++;
                    push_heap(q.begin(), q.begin() + r, cmp);
                }
                v = q.front().second;
                pop_heap(q.begin(), q.end(), cmp);
                q.pop_back();
                r--;
            }
            if (vis[v]) continue;
            vis[v] = true;
            if (v == t) break;
            i64 u = d[v][0], dis = d[v][1];
            for (int i = g.h[v]; i < g.h[v + 1]; i++) {
                auto e = g.e[i];
                if (!e.c) continue;
                i64 p = e.p - d[e.v][0] + u;
                if (d[e.v][1] - dis > p) {
                    i64 to = dis + p;
                    d[e.v][1] = to;
                    pre[e.v] = e.rev;
                    if (to == dis) {
                        qm.push_back(e.v);
                    } else {
                        q.push_back({to, e.v});
                    }
                }
            }
        }
    };
    ref();
    return d[t];
}

```

```

        }
    }
}
if (!vis[t]) {
    return false;
}

for (int v = 0; v < n; v++) {
    if (!vis[v]) continue;
    d[v][0] -= d[t][1] - d[v][1];
}
return true;
};

int r = 0;
i64 p = 0, cf = -1;
vector<Ans> ans(1);
while (r < f) {
    if (!ref()) break;
    int c = f - r;
    for (int v = t; v != s; v = g.e[pre[v]].v) {
        c = min(c, g.e[g.e[pre[v]].rev].c);
    }
    for (int v = t; v != s; v = g.e[pre[v]].v) {
        auto &e = g.e[pre[v]];
        e.c += c;
        g.e[e.rev].c -= c;
    }
    i64 D = -d[s][0];
    r += c;
    p += c * D;
    if (cf == D) {
        ans.pop_back();
    }
    ans.push_back({r, p});
    cf = D;
}
return ans;
}
};

```

费用流多类型原始对偶

```

template<class E>
struct csr {
    vector<int> h;
    vector<E> e;

    csr(int n, const vector<pair<int, E>> &edges)
        : h(n + 1), e(edges.size()) {
        for (auto u : edges) {
            h[u.first + 1]++;
        }
        for (int i = 1; i <= n; i++) {
            h[i] += h[i - 1];
        }
    }
};

```

```

    }
    auto c = h;
    for (auto u : edges) {
        e[c[u.first]++] = u.second;
    }
}
};

template<typename Cap, typename Cost>
struct MCFGGraph {
    MCFGGraph() {}

    MCFGGraph(int n) : n(n) {}

    void addEdge(int u, int to, Cap c, Cost p) {
        E.push_back({u, to, c, 0, p});
    }

    struct Edge {
        int u, v;
        Cap c, f;
        Cost p;
    };

    vector<Edge> Edges() { return E; }
    static constexpr Cap inf = numeric_limits<Cap>::max();
    using Ans = pair<Cap, Cost>;

    Ans flow(int s, int t, Cap f = inf) {
        return slope(s, t, f).back();
    }

    vector<Ans> slope(int s, int t, Cap f = inf) {
        int m = E.size();
        vector<int> id(m);

        auto g = [&]() {
            vector<int> d(n), rid(m);
            vector<pair<int, _Edge>> elist;
            elist.reserve(2 * m);
            for (int i = 0; i < m; i++) {
                auto e = E[i];
                id[i] = d[e.u]++;
                rid[i] = d[e.v]++;
                elist.push_back({e.u, {e.v, -1, e.c - e.f, e.p}});
                elist.push_back({e.v, {e.u, -1, e.f, -e.p}});
            }
            auto g = csr<_Edge>(n, elist);
            for (int i = 0; i < m; i++) {
                auto e = E[i];
                id[i] += g.h[e.u];
                rid[i] += g.h[e.v];
                g.e[id[i]].rev = rid[i];
                g.e[rid[i]].rev = id[i];
            }
        };
    }
};

```

```

        return g;
    }();

    auto ans = slope(g, s, t, f);

    for (int i = 0; i < m; i++) {
        E[i].f = E[i].c - g.e[id[i]].c;
    }

    return ans;
}

int n;
vector<Edge> E;

struct _Edge {
    int v, rev;
    Cap c;
    Cost p;
};

vector<Ans> slope(csr<_Edge> &g, int s, int t, Cap f) {
    vector<array<Cost, 2>> d(n);
    vector<int> pre(n), qm;
    vector<bool> vis(n);
    vector<pair<Cost, int>> q;
    auto cmp = greater<pair<Cost, int>>();
    auto ref = [&]() {
        for (int i = 0; i < n; i++) {
            d[i][1] = numeric_limits<Cost>::max();
        }
        fill(vis.begin(), vis.end(), false);
        qm.clear();
        q.clear();

        size_t r = 0;

        d[s][1] = 0;
        qm.push_back(s);
        while (!qm.empty() || !q.empty()) {
            int v;
            if (!qm.empty()) {
                v = qm.back();
                qm.pop_back();
            } else {
                while (r < q.size()) {
                    r++;
                    push_heap(q.begin(), q.begin() + r, cmp);
                }
                v = q.front().second;
                pop_heap(q.begin(), q.end(), cmp);
                q.pop_back();
                r--;
            }
            if (vis[v]) continue;
            vis[v] = true;

```



```

        if (v == t) break;
        Cost u = d[v][0], dis = d[v][1];
        for (int i = g.h[v]; i < g.h[v + 1]; i++) {
            auto e = g.e[i];
            if (!e.c) continue;
            Cost p = e.p - d[e.v][0] + u;
            if (d[e.v][1] - dis > p) {
                Cost to = dis + p;
                d[e.v][1] = to;
                pre[e.v] = e.rev;
                if (to == dis) {
                    qm.push_back(e.v);
                } else {
                    q.push_back({to, e.v});
                }
            }
        }
    }
    if (!vis[t]) {
        return false;
    }

    for (int v = 0; v < n; v++) {
        if (!vis[v]) continue;
        d[v][0] -= d[t][1] - d[v][1];
    }
    return true;
};

Cap r = 0;
Cost p = 0, cf = -1;
vector<Ans> ans(1);
while (r < f) {
    if (!ref()) break;
    Cap c = f - r;
    for (int v = t; v != s; v = g.e[pre[v]].v) {
        c = min(c, g.e[g.e[pre[v]].rev].c);
    }
    for (int v = t; v != s; v = g.e[pre[v]].v) {
        auto &e = g.e[pre[v]];
        e.c += c;
        g.e[e.rev].c -= c;
    }
    Cost D = -d[s][0];
    r += c;
    p += c * D;
    if (cf == D) {
        ans.pop_back();
    }
    ans.push_back({r, p});
    cf = D;
}
return ans;
}
};

```

单纯形

```
struct MCFGraph {
    struct Edge {
        int u, v, nxt;
        i64 f, w;
    };
    static constexpr int inf = numeric_limits<int>::max();
    vector<Edge> E;
    vector<int> fa, fe, cir, tag, H;
    vector<i64> pre;

    MCFGraph(int n, int m = 0): fa(n), fe(n), cir(n), tag(n), H(n), pre(n), E(2)
    {
        E.reserve(2 * m + 4);
    }

    int tot = 1;

    void addEdge(int u, int v, i64 f, i64 w) {
        E.push_back({u, v, H[u], f, + w}), H[u] = ++ tot;
        E.push_back({v, u, H[v], 0, - w}), H[v] = ++ tot;
    }

    int now = 0;

    void InitZCT(int x, int e, int nod = 1) {
        fa[x] = E[fe[x] = e].u, tag[x] = nod;
        for (int i = H[x]; i; i = E[i].nxt)
            if(tag[E[i].v] != nod && E[i].f)
                InitZCT(E[i].v, i, nod);
    }

    i64 sum(int x) {
        if(tag[x] == now) return pre[x];
        return tag[x] = now, pre[x] = sum(fa[x]) + E[fe[x]].w;
    }

    i64 PushFlow(int x) {
        int rt = E[x].u, lca = E[x].v, p = 2, del = 0, cnt = 0;
        ++ now;
        while(rt) tag[rt] = now, rt = fa[rt];
        while(tag[lca] != now) tag[lca] = now, lca = fa[lca];

        i64 f = E[x].f, cost = 0;

        for (int u = E[x].u; u != lca; u = fa[u]) {
            cir[++ cnt] = fe[u];
            if(E[fe[u]].f < f) del = u, p = 0, f = E[fe[u]].f;
        }

        for (int u = E[x].v; u != lca; u = fa[u]) {
            cir[++ cnt] = fe[u] ^ 1;
```

```

        if(E[fe[u] ^ 1].f < f) del = u, p = 1, f = E[fe[u] ^ 1].f;
    }

    cir[++ cnt] = x;

    for (int i = 1; i <= cnt; ++ i)
        cost += E[cir[i]].w * f, E[cir[i]].f -= f, E[cir[i] ^ 1].f += f;

    if(p == 2) return cost;
    int u = E[x].u, v = E[x].v;
    if(p == 1) std::swap(u, v);
    int le = x ^ p, lu = v, tmp;

    while(lu != del) {
        le ^= 1, -- tag[u], std::swap(fe[u], le);
        tmp = fa[u], fa[u] = lu, lu = u, u = tmp;
    }

    return cost;
}

pair<i64, i64> flow(int S, int T) {
    addEdge(T, S, inf, - inf);
    InitZCT(T, 0, ++ now);

    tag[T] = ++ now, fa[T] = 0;

    bool Run = 1;
    i64 MinC = 0;

    while(Run) {
        Run = 0;
        for (int i = 2; i <= tot; ++ i)
            if(E[i].f && E[i].w + sum(E[i].u) - sum(E[i].v) < 0)
                MinC += PushFlow(i), Run = 1;
    }

    MinC += E[tot].f * inf;

    return {E[tot].f, MinC};
}
};

```

单纯形未封装

```

using T = int;

constexpr int N = 2e5 + 2, M = 2 * N + 1e5;
int head[N], nxt[2 * M], to[2 * M];
T cap[2 * M];
int cur = 0;
int _n = 0;
int h[N], now[N];

```

```

void init(int n) {
    fill(head, head + n, -1);
    _n = n;
    cur = -1;
}

void addEdge(int u, int v, T c) {
    nxt[++cur] = head[u], to[cur] = v, cap[cur] = c, head[u] = cur;
    nxt[++cur] = head[v], to[cur] = u, cap[cur] = 0, head[v] = cur;
}

bool bfs(int s, int t) {
    fill(h, h + _n, -1);
    queue<int> q;
    h[s] = 0;
    now[s] = head[s];
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int i = head[u]; ~i; i = nxt[i]) {
            int v = to[i];
            T c = cap[i];
            if (c > 0 && h[v] == -1) {
                h[v] = h[u] + 1;
                now[v] = head[v];
                if (v == t) {
                    return true;
                }
                q.push(v);
            }
        }
    }
    return false;
}

T dfs(int u, int t, T f) {
    if (u == t) {
        return f;
    }
    T r = f;
    for (int &i = now[u]; ~i; i = nxt[i]) {
        int v = to[i];
        T c = cap[i];
        if (c > 0 && h[v] == h[u] + 1) {
            T k = dfs(v, t, min(c, r));
            if (k == 0) {
                h[v] = -1;
            }
            cap[i] -= k;
            cap[i ^ 1] += k;
            r -= k;
            if (r == 0) {
                return f;
            }
        }
    }
}

```

```

    }
}
return f - r;
}

T flow(int s, int t) {
    T ans = 0;
    while (bfs(s, t)) {
        ans += dfs(s, t, std::numeric_limits<T>::max());
    }
    return ans;
}

struct Edge {
    int from;
    int to;
    T cap;
    T flow;
};

vector<bool> minCut() {
    vector<bool> c(_n);
    for (int i = 0; i < _n; i++) {
        c[i] = (h[i] != -1);
    }
    return c;
}

vector<Edge> Edges() {
    vector<Edge> a;
    for (int i = 0; i < cur; i += 2) {
        Edge x;
        x.from = to[i + 1];
        x.to = to[i];
        x.cap = cap[i] + cap[i + 1];
        x.flow = cap[i + 1];
        a.push_back(x);
    }
    return a;
}

```

笛卡尔树

```

template<class T>
struct Descartes {
    int n;
    vector<T> v;
    vector<int> ls, rs;

    Descartes(int n) : ls(n, -1), rs(n, -1), v(n) {}

    Descartes(vector<T> &v) : n((ll) v.size()), ls(n, -1), rs(n, -1), v(v) {}

```

```

int build() /* return root */ {
    vector<int> s(n);
    int top = 0;
    int root = -1;
    for (int i = 0; i < n; ++i) {
        int realtop = top;
        while (top != 0 && v[s[top]] > v[i]) { --top; }
        if (top < realtop) ls[i] = s[top + 1];
        if (top != 0) rs[s[top]] = i;
        s[++top] = i;
    }
    root = s[1];
    assert(!s.empty());
    return root;
}
};

```

板题实现

欧拉图

```

#include <bits/stdc++.h>
using namespace std;

#ifdef LOCAL
    #include "C:\Users\Kevin\Desktop\demo\save\debug.h"
#else
    #define debug(...) 114514
    #define ps 114514
#endif

using ll = long long;
using i64 = long long;

void solve() {
    int n, m; cin >> n >> m;
    vector<vector<int>> a(n);
    vector<int> in(n);
    for (int i = 0; i < m; i += 1) {
        int u, v; cin >> u >> v; -- u, -- v;
        a[u].push_back(v);
        in[v] ++;
    }
    array<int, 2> cnt{}; int s = 0;
    for (int i = 0; i < n; i += 1) {
        if (a[i].size() != in[i]) {
            if (int(a[i].size()) - in[i] == -1) {
                cnt[0] += 1;
            } else if (int(a[i].size()) - in[i] == 1) {
                cnt[1] += 1; s = i;
            } else {
                cout << "No" << endl;
                return;
            }
        }
    }
}

```

```

    }
}
for (auto i : {0, 1}) {
    if (cnt[i] > 1) {
        cout << "No" << endl;
        return;
    }
}
vector<int> cur(n);
vector<int> seq;
auto dfs = [&] (auto&&dfs, int now) -> void {
    // ps;
    if (cur[now] == 0) sort(a[now].begin(), a[now].end());
    for (int &i = cur[now]; i < a[now].size(); i++) {
        dfs(dfs, a[now][i]);
    }
    seq.push_back(now);
};
dfs(dfs, s);
reverse(seq.begin(), seq.end());
for (auto u : seq) {
    cout << u + 1 << ' ';
}
cout << endl;
}

signed main () {
#ifdef cin
    ios::sync_with_stdio (false);
    cin.tie (nullptr) ;
#endif
    // __fin("C:\\Users\\Kevin\\Desktop\\cpp\\in.in") ;

    i64 _ = 1 ;
    // cin >> _ ;
    while (_ --) {
        // debug(_);
        solve ();
    }
    return 0 ;
}

```

静态圆方树

```

template <class E>
struct csr {
    vector<int> s;
    vector<E> adj;
    csr() {}
    csr(int n, const vector<pair<int, E>>& g)
        : s(n + 1), adj(g.size()) {
        for (auto e : g) {
            s[e.first + 1]++;
        }
        for (int i = 1; i <= n; i++) {

```

```

        s[i] += s[i - 1];
    }
    auto c = s;
    for (auto e : g) {
        adj[c[e.first]++] = e.second;
    }
}

};

// static weighted tree
template<int N = int(1E6)>
struct multiTree {
    int n;
    int K;
    vector<array<int, __lg(N) + 1>> p;
    struct Edge {
        int v; i64 w;
        friend ostream &operator<<(ostream &cout, Edge u) {
            return cout << "(" << u.v << ", " << u.w << ")";
        }
    };
    vector<pair<int, Edge>> e;
    csr<Edge> g;
    vector<int> dep;
    vector<i64> dep2;
    multiTree() {}
    multiTree(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        K = __lg(n) + 1;
        p.assign(n, {});
        dep.assign(n, 0);
        dep2.assign(n, 0);
    }
    void addEdge(int u, int v, auto w) {
        e.push_back({u, {v, w}});
        e.push_back({v, {u, w}});
    }
    void work(int r = 0) {
        g = csr<Edge>(n, e);
        dfs(r, r);
    }
    void dfs(int u, int f) {
        p[u][0] = f;
        for (int i = 1; i < K; i += 1) {
            p[u][i] = p[p[u][i - 1]][i - 1];
        }
        for (int i = g.s[u]; i < g.s[u + 1]; i += 1) {
            auto [v, w] = g.adj[i];
            if (v == f) {
                continue;
            }
            dep[v] = dep[u] + 1;
            dep2[v] = dep2[u] + w;
        }
    }
};

```



```

        dfs(v, u);
    }
}

int jump(int u, int d) {
    for (int i = dep[u] - d; i > 0; i -= 1 << __lg(i)) {
        u = p[u][__lg(i)];
    }
    return u;
}

int lca(int u, int v) {
    if (dep[u] < dep[v]) {
        swap(u, v);
    }
    u = jump(u, dep[v]);
    if (u == v) {
        return u;
    }
    for (int i = __lg(dep[u]); i >= 0; i -= 1) {
        if (p[u][i] != p[v][i]) {
            u = p[u][i], v = p[v][i];
        }
    }
    return p[u][0];
}

int dist(int u, int v) {
    int m = lca(u, v);
    return dep[u] + dep[v] - 2 * dep[m];
}

int dis2(int u, int v) {
    int m = lca(u, v);
    return dep2[u] + dep2[v] - 2 * dep[m];
}
};

template<class E>
struct star {
    vector<int> s, t;
    star() {}
    star(int n, const vector<pair<int, E>> &e): s(n, -1) {
        // debug(e);
        t.reserve(e.size());
        for (int i = 0; i < e.size(); i += 1) {
            int u = e[i].first;
            t.push_back(s[u]), s[u] = i;
        }
    }
};

struct RST : public multiTree<> {
    using multiTree<>::operator=; // 导入基类的所有成员
    int n, N;
    struct Edge {
        int v; i64 w;
        friend ostream &operator<<(ostream &cout, Edge u) {
            return cout << "(" << u.v << ", " << u.w << ")";
        }
    }
};

```

```

};
vector<pair<int, Edge>> e;

vector<i64> s, cir;
RST(int n): n(n), N(n), s(n), cir(n) {}
void tarjan() {
    star<Edge> g(n, e);
    int cur = 0;
    vector<int> dfn(n, -1), low(n, -1), fe(n, -1);
    auto tarjan = [&] (auto &&tarjan, int u, int E) -> void {
        dfn[u] = low[u] = cur ++;
        for (int i = g.s[u]; i != -1; i = g.t[i]) {
            auto [v, w] = e[i].second;
            if (dfn[v] == -1) {
                fe[v] = i;
                tarjan(tarjan, v, i);
                low[u] = min(low[u], low[v]);
                if (dfn[u] < low[v]) {
                    addEdge2(u, v, w);
                }
            } else if (i != (E ^ 1)) {
                low[u] = min(low[u], low[v]);
            }
        }
        for (int i = g.s[u]; i != -1; i = g.t[i]) {
            auto [v, w] = e[i].second;
            if (dfn[u] < dfn[v] && fe[v] != i) {
                int t = N ++;
                i64 sum = w;
                for (int j = v; j != u; j = e[fe[j]].first) {
                    s[j] = sum;
                    sum += e[fe[j]].second.w;
                }
                for (int j = v; j != u; j = e[fe[j]].first) {
                    addEdge2(t, j, std::min(s[j], sum - s[j]));
                    cir[j] = sum;
                }
                addEdge2(u, t, 0);
            }
        }
    };
    tarjan(tarjan, 0, 1E9);
    init(N);
    work();
}

void addEdge(int u, int v, i64 w) {
    e.push_back({u, {v, w}});
    e.push_back({v, {u, w}});
}

void addEdge2(int u, int v, i64 w) {
    multiTree::addEdge(u, v, w);
}

i64 dis(int u, int v) {
    int m = lca(u, v);
    if (m < n) {
        return dep2[u] + dep2[v] - 2 * dep2[m];
    }
}

```

```

    }
    int d = dep[m];
    int x = jump(u, d + 1), y = jump(v, d + 1);
    i64 delta = std::abs(s[x] - s[y]);
    return dep2[u] + dep2[v] - dep2[x] - dep2[y] + std::min(delta, cir[x] -
delta);
    }
};

```

存图

稀疏行压缩

```

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

```

杂

初始

```

# include <bits/stdc++.h>
using namespace std;

using i64 = long long;

void solve () {
}

// 修一下爆没爆int
// 多测

signed main () {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int t = 1;
    cin >> t;
}

```

```

while (t --) {
    solve ();
}
return 0;
}

```

对拍

- 一共 4 个文件:
 - baoli.cpp
 - std.cpp
 - data.cpp
 - 关键

```

std::mt19937
rng(std::chrono::steady_clock::now().time_since_epoch().count());

int gen(int min, int max) {
    std::uniform_int_distribution<long long> dis(min, max);
    return dis(rng);
}

shuffle(v.begin(), v.end(), rng);

```

```

# include <bits/stdc++.h>
using namespace std;

using i64 = long long;

std::mt19937
rng(std::chrono::steady_clock::now().time_since_epoch().count());

int gen(int min, int max) {
    std::uniform_int_distribution<long long> dis(min, max);
    return dis(rng);
}

int main () {
    std::ios::sync_with_stdio (false);
    std::cin.tie (nullptr) ;
    vector<int> a;
    shuffle(a.begin(), a.end(), rng);

    return 0 ;
}

```

- 对拍.cpp

- ```

include<bits/stdc++.h>
using namespace std;

void solve() {

```

```

 for (int i = 1; ; i += 1) {
 system("data.exe > in.txt");
 system("std.exe < in.txt > std.txt");
 system("baoli.exe < in.txt > baoli.txt");
 if (system("fc std.txt baoli.txt")) {
 cout << "case " << i << " Wrong Answer" << endl;
 system("pause");
 } else {
 cout << "case " << i << " Accepted Answer" << endl;
 }
 }
}

signed main() {
 ios::sync_with_stdio(0);
 cin.tie(0);
 signed t = 1;
 // cin >> t;
 while (t --) {
 solve();
 }
 return 0;
}

```

## 简易版取模类

```

template<typename T>
T power(T x, long long b) {
 T res = 1;
 while (b) {
 if (b & 1) res *= x;
 x *= x;
 b >>= 1;
 }
 return res;
}

template<int P>
struct mod_int {
 int x;
 static int mod;
 mod_int() : x{} {}
 mod_int(long long x) : x(norm(x % getMod())) {}

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

 static void setMod(int x) {
 mod = x;
 }

 static int getMod() {

```

```

 return (P > 0 ? P : mod);
 }
 mod_int operator-() {
 return -x;
 }

 mod_int &operator+=(mod_int rhs) {
 x = norm(x + rhs.x);
 return *this;
 }
 mod_int &operator-=(mod_int rhs) {
 x = norm(x - rhs.x);
 return *this;
 }
 mod_int &operator*=(mod_int rhs) {
 x = 111 * x * rhs.x % getMod();
 return *this;
 }

 mod_int inv() {
 return power(*this, P - 2);
 }
 mod_int &operator/=(mod_int rhs) {
 x = 111 * x * rhs.inv().x % getMod();
 return *this;
 }

 friend mod_int operator+(mod_int lhs, mod_int rhs) {
 return lhs += rhs;
 }
 friend mod_int operator-(mod_int lhs, mod_int rhs) {
 return lhs -= rhs;
 }
 friend mod_int operator*(mod_int lhs, mod_int rhs) {
 return lhs *= rhs;
 }
 friend mod_int operator/(mod_int lhs, mod_int rhs) {
 return lhs /= rhs;
 }
 friend bool operator==(mod_int lhs, mod_int rhs) {
 return lhs.x == rhs.x;
 }
 friend bool operator!=(mod_int lhs, mod_int rhs) {
 return lhs.x != rhs.x;
 }

 template<class istream>
 friend istream &operator>>(istream &input, mod_int &rhs) {
 long long x;
 input >> x;
 rhs = x;
 return input;
 }
 template<class ostream>
 friend ostream &operator<<(ostream &output, mod_int rhs) {
 return output << rhs.x;
 }

```

```

 }
};

template<>
int mod_int<0>::mod = 998244353;

constexpr int P = 1e9 + 7;
using Z = mod_int<P>;

```

## 取模类丐版

```

struct Z {
 static constexpr int P = 998244353;
 int x = 0;
 Z() {}
 Z(i64 x) : x(norm(x % P)) {}
 int norm(int x) {
 if (x >= P) {
 x -= P;
 }
 if (x < 0) {
 x += P;
 }
 return x;
 }
 Z operator-() {
 return -x;
 }
 Z &operator+=(Z rhs) {
 x = norm(x + rhs.x);
 return *this;
 }
 Z &operator-=(Z rhs) {
 x = norm(x - rhs.x);
 return *this;
 }
 Z &operator*=(Z rhs) {
 x = 1ll * x * rhs.x % P;
 return *this;
 }
 friend Z operator+(Z lhs, Z rhs) {
 return lhs += rhs;
 }
 friend Z operator-(Z lhs, Z rhs) {
 return lhs -= rhs;
 }
 friend Z operator*(Z lhs, Z rhs) {
 return lhs *= rhs;
 }
 friend istream &operator>>(istream &cin, Z &rhs) {
 i64 x;
 cin >> x;
 rhs = x;
 return cin;
 }
}

```

```

 friend ostream &operator<<(ostream &cout, Z rhs) {
 return cout << rhs.x;
 }
};

```

## debug.h

```

template<typename A, typename B>
ostream &operator<<(ostream &cout, const pair<A, B> &p) {
 return cout << '(' << p.first << ", " << p.second << ')';
}

template<typename Tp, typename T = typename
 enable_if<!is_same<Tp, string>::value, typename Tp::value_type>::type>
ostream &operator<<(ostream &cout, const Tp &v) {
 cout << '{';
 string sep;
 for (const T &x : v)
 cout << sep << x, sep = ", ";
 return cout << '}';
}

void Output() { cerr << endl; }
template<typename Head, typename... Tail>
void Output(Head H, Tail... T) {
 cerr << ' ' << H; Output(T...);
}

define ps cerr << "YES" << endl
define debug(...) \
 cerr << "(" << #__VA_ARGS__ << "):" << endl, \
 Output(__VA_ARGS__)

```

## hash

```

struct Hash {
 static uint64_t splitmix64(uint64_t x) {
 x += 0x9e3779b97f4a7c15;
 x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
 x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
 return x ^ (x >> 31);
 }

 size_t operator()(uint64_t x) const {
 static const uint64_t FIXED_RANDOM =
 chrono::steady_clock::now().time_since_epoch().count();
 return splitmix64(x + FIXED_RANDOM);
 }
}

// 针对 std::pair<int, int> 作为主键类型的哈希函数
size_t operator()(pair<uint64_t, uint64_t> x) const {
 static const uint64_t FIXED_RANDOM =
 chrono::steady_clock::now().time_since_epoch().count();

```



```

 return splitmix64(x.first + FIXED_RANDOM) ^
 (splitmix64(x.second + FIXED_RANDOM) >> 1);
 }
};

```

## O2优化

```

#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,avx2,fma")
#pragma GCC optimize("unroll-loops")

```

## 快读

```

struct Input {
 using i64 = long long;
 Input() {}
 static constexpr int MAXSIZE = 1 << 20;

 char buf[MAXSIZE], *p1 = buf, *p2 = buf;
 # define isdigit(x) ('0' <= x && x <= '9')

 #define gc()
 \
 (p1 == p2 &&(p2 =(p1 = buf) + fread(buf, 1, MAXSIZE, stdin), p1 == p2) \
 ? EOF
 \
 : *p1++)

 bool blank(char ch) {
 return ch == ' ' || ch == '\n' || ch == '\r' || ch == '\t' || ch == EOF;
 }
 void tie(int x) {}
 template <typename T>
 Input &operator>>(T &x) {
 x = 0;
 bool sign = 0;
 char ch = gc();
 for (; !isdigit(ch); ch = gc())
 if(ch == '-') sign = 1;
 for (; isdigit(ch); ch = gc())
 x = (x << 3) + (x << 1) + ch - '0';
 if(sign) x = -x;
 return *this;
 }
 Input &operator>>(char &x) {
 x = ' ';
 for (; blank(x); x = gc());
 return *this;
 }
 Input &operator>>(double &x) {
 x = 0;
 double tmp = 1;
 bool sign = 0;
 char ch = gc();

```

```

 for (; !isdigit(ch); ch = gc())
 if(ch == '-') sign = 1;
 for (; isdigit(ch); ch = gc())
 x = x * 10 + ch - '0';
 if(ch == '.')
 for (ch = gc(); isdigit(ch); ch = gc())
 tmp /= 10.0, x += tmp *(ch - '0');
 if(sign) x = -x;
 return *this;
 }

 Input &operator>>(string &s) {
 s.clear();
 char ch = gc();
 for (; blank(ch); ch = gc());
 for (; !blank(ch); ch = gc()) {
 s += ch;
 }
 return *this;
 }
 # undef isdigit
 # undef gc
}input;
define cin input
struct Output {
 struct setprecision {
 int precision;
 };
 static constexpr int MAXSIZE = 1 << 20;
 char pbuf[MAXSIZE], *pp = pbuf;
 void push(const char &c) {
 if(pp - pbuf == MAXSIZE)
 fwrite(pbuf, 1, MAXSIZE, stdout), pp = pbuf;
 *pp++ = c;
 }
 int precision;
 Output() { precision = 6;}
 ~Output() { fwrite(pbuf, 1, pp - pbuf, stdout);}
 char stack[40];
 int top = 0;
 template<class T>
 Output &operator<<(const T &x) {
 T tmp = x;
 bool _ = tmp < 0;
 if(_) tmp *= -1;
 while(tmp) stack[++ top] = '0' + tmp % 10, tmp /= 10;
 if(_) stack[++ top] = '-';
 while(top) push(stack [top]), -- top;
 if(x == 0)push('0');
 return *this;
 }
 Output &operator<<(const string &x) {
 for (auto &u : x) push(u);
 return *this;
 }
 template<size_t N>
 Output &operator<<(const char(&x)[N]) {

```

```

 *this << string(x);
 return *this;
 }
 Output &operator<<(const char* const &x) {
 for (const char* ptr = x; *ptr != '\0'; ++ptr)
 push(*ptr);
 return *this;
 }
 Output &operator<<(const char &x) {
 push(x);
 return *this;
 }
 Output &operator<<(const bool &x) {
 push(x ? '1' : '0');
 return *this;
 }
 Output &operator<<(const double &x) {
 int intPart = static_cast<int>(x);
 *this << intPart;

 push('.');

 double decimalPart = x - intPart;
 for (int i = 0; i < precision; ++i) {
 decimalPart *= 10;
 int digit = static_cast<int>(decimalPart);
 *this << char('0' + digit);
 decimalPart -= digit;
 }
 return *this;
 }
 Output &operator<<(setprecision x) {
 precision = x.precision;
 return *this;
 }
 # undef push
}output;
define cout output

```

## u32 指针

```

/**
 * 1 MB = 1024 KB
 * 1 KB = 1024 B
 * 134210000 128
 * 262144000 256
 * 520000000 524
 * 1070000000 1024
 * 注意事项：记得内存别开小了或者别爆了
 */

constexpr int max_size = 520000000;
uint8_t buf[max_size];
uint8_t *head = buf;

```

```

using u32 = uint32_t;

template <class T>
struct Base {
 u32 x;
 Base(u32 x = 0) : x(x) {}
 T *operator->() {
 return (T *) (buf + x);
 }
 T &operator*() {
 return *((T *) (buf + x));
 }
 operator bool() {
 return x;
 }
 operator u32() {
 return x;
 }
 bool operator==(Base rhs) const {
 return x == rhs.x;
 }
 static Base news() {
 return (head += sizeof(T)) - buf;
 }
};

```

# 字符串

## Ac自动机

```

struct AhoCorasick {
 static constexpr int ALPHABET = 26;
 struct Node {
 int len;
 int link;
 int top;
 int val;
 int d;
 std::array<int, ALPHABET> next;
 Node() : len{}, link{}, next{}, top{}, val {-1}, d{} {}
 };

 std::vector<Node> t;

 AhoCorasick() {
 init();
 }

 void init() {
 t.assign(2, Node());
 }
};

```

```

 t[0].next.fill(1);
 t[0].len = -1;
}

int newNode() {
 t.emplace_back();
 return t.size() - 1;
}

int add(const std::vector<int> &a) {
 int p = 1;
 for (auto x : a) {
 if (t[p].next[x] == 0) {
 t[p].next[x] = newNode();
 t[t[p].next[x]].len = t[p].len + 1;
 }
 p = t[p].next[x];
 }
 apply (t[p].val);
 return p;
}

int add(const std::string &a, char offset = 'a') {
 std::vector<int> b(a.size());
 for (int i = 0; i < a.size(); i++) {
 b[i] = a[i] - offset;
 }
 return add(b);
}

void work() {
 std::queue<int> q;
 q.push(1);

 while (!q.empty()) {
 int x = q.front();
 q.pop();

 t[x].top = t[link(x)].val >= 0 ? link(x) : top(link(x));

 for (int i = 0; i < ALPHABET; i++) {
 if (t[x].next[i] == 0) {
 t[x].next[i] = t[t[x].link].next[i];
 } else {
 t[t[x].next[i]].link = t[t[x].link].next[i];
 t[t[t[x].link].next[i]].d += 1;
 q.push(t[x].next[i]);
 }
 }
 }
}

int next(int p, int x) {
 return t[p].next[x];
}

```

```

int next(int p, char c, char offset = 'a') {
 return next(p, c - 'a');
}

int link(int p) {
 return t[p].link;
}

int len(int p) {
 return t[p].len;
}

int& val(int p) {
 return t[p].val;
}

int top (int p) {
 return t[p].top;
}

int size() {
 return t.size();
}

int& d (int p) {
 return t[p].d;
}

void apply (auto& val) {
 val = 0 ;
}
};

```

## 字符串哈希

```

std::mt19937 rng(std::chrono::steady_clock::now().time_since_epoch().count());

bool isprime(int n) {
 if (n <= 1) return false;
 for (int i = 2; i * i <= n; i++)
 if (n % i == 0)
 return false;
 return true;
}

int findPrime(int n) {
 while (!isprime(n))
 n++;
 return n;
}

template<int N>
struct StringHash {
 static array<int, N> mod;
 static array<int, N> base;
 vector<array<int, N>> p, h;

```

```

StringHash() = default;
StringHash(const string& s) {
 int n = s.size();
 p.resize(n);
 h.resize(n);
 fill(p[0].begin(), p[0].end(), 1);
 for (int i = 0; i < n; i++)
 for (int j = 0; j < N; j++) {
 p[i][j] = 111 * (i == 0 ? 111 : p[i - 1][j]) * base[j] % mod[j];
 h[i][j] = (111 * (i == 0 ? 011 : h[i - 1][j]) * base[j] + s[i]) %
mod[j];
 }
 }

 array<int, N> query(int l, int r) {
 assert(r >= l - 1);
 array<int, N> ans{};
 if (l > r) return {0, 0};
 for (int i = 0; i < N; i++) {
 ans[i] = (h[r][i] - 111 * (l == 0 ? 011 : h[l - 1][i]) * (r - l + 1
== 0 ? 111 : p[r - 1][i]) % mod[i] + mod[i]) % mod[i];
 }
 return ans;
 }
};

constexpr int HN = 2;
template<>
array<int, 2> StringHash<HN>::mod =
 {findPrime(rng() % 900000000 + 100000000),
 findPrime(rng() % 900000000 + 100000000)};
template<>
array<int, 2> StringHash<HN>::base {13331, 131};
using Hashing = StringHash<HN>;

```

## 后缀数组

```

using i64 = long long;
struct SuffixArray {
 int n;
 std::vector<int> sa, rk, lc;
 SuffixArray(const std::string &s) {
 n = s.length();
 sa.resize(n);
 lc.resize(n - 1);
 rk.resize(n);
 std::iota(sa.begin(), sa.end(), 0);
 std::sort(sa.begin(), sa.end(), [&](int a, int b) {return s[a] < s[b];});
 rk[sa[0]] = 0;
 for (int i = 1; i < n; ++i)
 rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
 int k = 1;
 std::vector<int> tmp, cnt(n);
 tmp.reserve(n);
 while (rk[sa[n - 1]] < n - 1) {
 tmp.clear();

```

```

 for (int i = 0; i < k; ++i)
 tmp.push_back(n - k + i);
 for (auto i : sa)
 if (i >= k)
 tmp.push_back(i - k);
 std::fill(cnt.begin(), cnt.end(), 0);
 for (int i = 0; i < n; ++i)
 ++cnt[rk[i]];
 for (int i = 1; i < n; ++i)
 cnt[i] += cnt[i - 1];
 for (int i = n - 1; i >= 0; --i)
 sa[--cnt[rk[tmp[i]]]] = tmp[i];
 std::swap(rk, tmp);
 rk[sa[0]] = 0;
 for (int i = 1; i < n; ++i)
 rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] || sa[i - 1] + k == n || tmp[sa[i - 1] + k] < tmp[sa[i] + k]);
 k *= 2;
 }
 for (int i = 0, j = 0; i < n; ++i) {
 if (rk[i] == 0) {
 j = 0;
 } else {
 for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j < n && s[i + j] == s[sa[rk[i] - 1] + j];)
 ++j;
 lc[rk[i] - 1] = j;
 }
 }
}
};

```

## KMP

```

struct KMP{
 int n;
 std::vector<int> pi;
 std::vector<vector<int>> aut;

 KMP(const std::string &s) {
 n = (int)s.length();
 prefix_function(s);
 compute_automaton(s);
 }

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

```



```

void compute_automaton(string s) {
 aut.resize(n, vector<int>(26));
 for (int i = 0; i < n; i++) {
 for (int c = 0; c < 26; c++) {
 if (i > 0 && 'a' + c != s[i])
 aut[i][c] = aut[pi[i - 1]][c];
 else
 aut[i][c] = i + ('a' + c == s[i]);
 }
 }
}
};

```

## Trie

```

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;
 }
};

constexpr int N = 2e5;

struct node;
using Trie = u32_p<node>;

struct node {
 array<Trie, 2> ch{};
 int x; int sum;
};

```

```
};
```

## Manacher

```
std::vector<int> manacher(std::string s) {
 std::string t = "#";
 for (auto c : s) {
 t += c;
 t += '#';
 }
 int n = t.size();
 std::vector<int> r(n);
 for (int i = 0, j = 0; i < n; i++) {
 if (2 * j - i >= 0 && j + r[j] > i) {
 r[i] = std::min(r[2 * j - i], j + r[j] - i);
 }
 while (i - r[i] >= 0 && i + r[i] < n && t[i - r[i]] == t[i + r[i]]) {
 r[i] += 1;
 }
 if (i + r[i] > j + r[j]) {
 j = i;
 }
 }
 return r;
}
```

## Z函数

```
std::vector<int> zFunction(std::string s) {
 int n = s.size();
 std::vector<int> z(n + 1);
 z[0] = n;
 for (int i = 1, j = 1; i < n; i++) {
 z[i] = std::min(j + z[j] - i, z[i - j]);
 while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
 z[i]++;
 }
 if (i + z[i] > j + z[j]) {
 j = i;
 }
 }
 return z;
}
```

## PAM

```
struct PAM {
 static constexpr int ALPHABET_SIZE = 28;
 struct Node {
 int len;
 int link;
 int cnt;
 std::array<int, ALPHABET_SIZE> next;
 };
};
```

```

Node() : len{}, link{}, cnt{}, next{} {}
};
std::vector<Node> t;
int suff;
std::string s;
PAM() { init(); }
void init() {
 t.assign(2, Node());
 t[0].len = -1;
 suff = 1;
 s.clear();
}
int newNode() {
 t.emplace_back();
 return t.size() - 1;
}

bool add(char c, char offset = 'a') {
 int pos = s.size();
 s += c;
 int let = c - offset;
 int cur = suff, curlen = 0;

 while (true) {
 curlen = t[cur].len;
 if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])
 break;
 cur = t[cur].link;
 }
 if (t[cur].next[let]) {
 suff = t[cur].next[let];
 return false;
 }

 int num = newNode();
 suff = num;
 t[num].len = t[cur].len + 2;
 t[cur].next[let] = num;

 if (t[num].len == 1) {
 t[num].link = 1;
 t[num].cnt = 1;
 return true;
 }

 while (true) {
 cur = t[cur].link;
 curlen = t[cur].len;
 if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
 t[num].link = t[cur].next[let];
 break;
 }
 }

 t[num].cnt = 1 + t[t[num].link].cnt;
}

```

```

 return true;
 }
};

PAM pam;
// 应用:
// 1: 求s本质不同回文串个数: 自动机状态数
// 2: 求所有回文子串分别出现次数: 插入的时候cnt[last]++, 然后查询的时候倒推
cnt[fail[i]]+=cnt[i]
// 3: 以第i个位置为结尾的回文串个数, cnt[i]=cnt[fail[i]]+1, 边加边查cnt[last]

```

## SAM

```

struct SAM {
 static constexpr int ALPHABET_SIZE = 26;
 struct Node {
 int len;
 int link;
 std::array<int, ALPHABET_SIZE> next;
 Node() : len{}, link{}, next{} {}
 };
 std::vector<Node> t;
 SAM() { init(); }
 void init() {
 t.assign(2, Node());
 t[0].next.fill(1);
 t[0].len = -1;
 }
 int newNode() {
 t.emplace_back();
 return t.size() - 1;
 }
 int extend(int p, int c) {
 if (t[p].next[c]) {
 int q = t[p].next[c];
 if (t[q].len == t[p].len + 1) {
 return q;
 }
 int r = newNode();
 t[r].len = t[p].len + 1;
 t[r].link = t[q].link;
 t[r].next = t[q].next;
 t[q].link = r;
 while (t[p].next[c] == q) {
 t[p].next[c] = r;
 p = t[p].link;
 }
 return r;
 }
 int cur = newNode();
 t[cur].len = t[p].len + 1;
 while (!t[p].next[c]) {
 t[p].next[c] = cur;
 p = t[p].link;
 }
 }
};

```

```

 }
 t[cur].link = extend(p, c);
 return cur;
}
// int extend(int p, char c, char offset = 'a') {
// return extend(p, c - offset);
// }

int next(int p, int x) { return t[p].next[x]; }

// int next(int p, char c, char offset = 'a') { return next(p, c - 'a'); }

int link(int p) { return t[p].link; }

int len(int p) { return t[p].len; }

int size() { return t.size(); }

string lcs(const string& s, char offset = 'a') {
 int p = 1, l = 0;
 int pos = 0, len = 0;
 int cnt = 0;
 for (auto i : s) {
 while (p != 1 && (next(p, i - offset) == 0)) {
 p = link(p);
 l = t[p].len;
 }
 if (next(p, i - offset)) {
 p = next(p, i - offset);
 l++;
 }
 if (l > len) {
 len = l;
 pos = cnt;
 }
 cnt++;
 }
 return s.substr(pos - len + 1, len);
};
};

// 应用：
// 1: 检查字符串是否出现
// 给一个文本串 T 和多个模式串 P，我们要检查字符串 P 是否作为 T
// 的一个子串出现。 我们在
// O(T)
// 的时间内对文本串 T 构造后缀自动机。为了检查模式串 P 是否在 T
// 中出现，我们沿转移（边）从 t0
// 开始根据 P 的字符进行转移。如果在某个点无法转移下去，则模式串 P 不是 T
// 的一个子串。如果我们能够这样处理完整个字符串 P，那么模式串在 T 中出现过。
// 对于每个字符串 P，算法的时间复杂度为 O(P)
// 此外，这个算法还找到了模式串 P 在文本串中出现的最大前缀长度。

// 2: 出现次数
// 对于一个给定的文本串 T，有多组询问，每组询问给一个模式串 P，
// 回答模式串 P 在字符串 T 中作为子串出现了多少次。

```

```

// 对文本串 T 构造后缀自动机。
// 接下来做预处理：对于自动机中的每个状态 v ，预处理 cnt_v
// 使之等于 $endpos(v)$ 集合的大小。事实上，对应同一状态 v 的所有子串在文本串 T
// 中的出现次数相同，这相当于集合 $endpos$ 中的位置数。
// 然而我们不能明确的构造集合 $endpos$ ，因此我们只考虑它们的大小 cnt
// 为了计算这些值，我们进行以下操作。对于每个状态，
// 如果它不是通过复制创建的（且它不是初始状态 t_0 ），
// 我们将它的 cnt 初始化为 1。然后我们按它们的长度 len 降序遍历所有状态，
// 并将当前的 cnt_v 的值加到后缀链接指向的状态上，即：
// $cnt_{link(v)} += cnt_v$
// 最后回答询问只需要查找值 cnt_t ，其中 t 为模式串对应的状态，
// 如果该模式串不存在答案就为 0。单次查询的时间复杂度为 $O(P)$ ，预处理复杂度 $O(|T|)$

// 3: LCS
// 对 S 构造后缀自动机，处理 T 串

```

## 子序列自动机

```

auto get_nxt(string s) {
 int n = (int)s.size() - 1;
 vector<vector<int>> nxt(n + 2, vector<int>(26, n + 1));
 for (int i = n; i >= 0; i--) {
 for (int j = 0; j < 26; j++) {
 if (i == n)
 nxt[i][j] = n + 1;
 else
 nxt[i][j] = nxt[i + 1][j];
 }
 if (i != n)
 nxt[i][s[i + 1] - 'a'] = i + 1;
 }
 return nxt;
}

auto jump(string s, vector<vector<int>> &nxt) {
 int now = 0;
 for (int i = 0; i < s.size(); i++) {
 now = nxt[now][s[i] - 'a'];
 }
 return now;
}

```

# 动态规划

## dp优化

### 斜率优化

板子：x 单调, k 单调

```
// k层dp, 每层n位
int n, k;
vector<ll> f(n + 1), g(n + 1); // 滚动数组

// 斜率优化, 点(x, y), 斜率k,
auto x = [&](int i) { return 1; }; //
auto y = [&](int i) { return 1; }; //
auto k = [&](int i) { return 1; }; //

// 计算斜率
auto slope = [&](int i, int j) -> long double{
 if(x(j) == x(i)) return (y(j) >= y(i) ? 1e18 : -1e18);
 else {
 return (long double)(y(j) - y(i)) / (x(j) - x(i));
 }
};

// 队列存凸包
vector<int> q(n + 3);

for(int i = 1; i <= n; ++i) {
 // g[i] = ... ;
 // 初始化k = 1, 一般可以直接计算
}

// 下凸包为例
for(int c = 2; c <= k; ++c) {
 int head = 1, tail = 0;
 q[++tail] = 0;
 for(int i = 1; i <= n; ++i) {
 while(head < tail && slope(q[head], q[head + 1]) <= k(i)) ++head;
 ll B = y(q[head]) - k(i) * x(q[head]);
 // f[i] = B + ...; f[i] 与 B 之间的式子
 while(head < tail && i != n && slope(q[tail - 1], q[tail]) >=
slope(q[tail], i)) --tail;
 q[++tail] = i;
 }
 std::swap(f, g);
}
cout << g[n];
```

## 板子：x 单调, k 不单调

```
int n;
vector<ll> dp(n + 1, 1e18);
dp[0] = 0;

// 斜率优化, 点(x, Y), 斜率K,
auto x = [&](int i) { return 1; }; //
auto Y = [&](int i) { return 1; }; //
auto K = [&](int i) { return 1; }; //

// 计算斜率
auto slope = [&](int i, int j) -> long double{
 if(x(j) == x(i)) return (Y(j) >= Y(i) ? 1e18 : -1e18);
 else {
 return (long double)(Y(j) - Y(i)) / (x(j) - x(i));
 }
};

// 队列维护凸包
vector<int> q(n + 5);
int head = 1, tail = 0;
q[++tail] = 0;

// 二分最优策略点, 下凸包为例
auto ask = [&](ll k) {
 int l = head, r = tail;
 while(l < r) {
 int mid = (l + r) >> 1;
 if(slope(q[mid], q[mid + 1]) >= k) r = mid;
 else l = mid + 1;
 }
 return q[l];
};

// 下凸包为例
for(int i = 1; i <= n; ++i) {
 int j = ask(K(i));
 ll B = Y(j) - K(i) * x(j);
 // dp[i] = B + ... ;
 while(head < tail && i != n && slope(q[tail - 1], q[tail]) >= slope(q[tail],
i)) --tail;
 q[++tail] = i;
}

cout << dp[n];
```

## 板子：x 不单调, k 不单调

```
// CDQ板子, 以下凸包为例
using ll = long long;
const int maxn = 1e5 + 5;

struct node {
```



```

 int id;
 ll x, y, k;
};
// a表示原数组, b为归并辅助数组
vector<node> a(maxn), b(maxn);

ll X(int i) { return a[i].x; }
ll Y(int i) { return a[i].y; }
ll K(int i) { return a[i].k; }
long double slope(int i, int j) {
 if(X(j) == X(i)) return (Y(j) >= Y(i) ? 1e20 : -1e20);
 else {
 return (long double)(Y(j) - Y(i)) / (X(j) - X(i));
 }
};

// dp数组, 切记f[]的初始化
vector<ll> f(maxn, 1e18);

// 按照x进行归并
void merge(int L, int mid, int R)
{
 int p1 = L, p2 = mid + 1;
 int tp = L;
 while(p1 <= mid && p2 <= R) {
 if(a[p1].x <= a[p2].x) b[tp++] = a[p1++];
 else b[tp++] = a[p2++];
 }
 while(p1 <= mid) b[tp++] = a[p1++];
 while(p2 <= R) b[tp++] = a[p2++];
 for(int i = L; i <= R; ++i) a[i] = b[i];
}

void cdq(int L, int R) {
 if(L == R) {
 int pos = a[L].id;
 // f[pos] = ...; //视情况而修改, 有些求解为前缀最优, 则在此处修改。
 // 例f[pos] = max(f[pos], f[pos - 1]);
 // a[L].x = ;
 // a[L].y = ;
 return ;
 }

 int mid = (L + R) >> 1;
 // 分为左右两边
 int p1 = L, p2 = mid + 1;
 for(int i = L; i <= R; ++i) {
 if(a[i].id <= mid) b[p1++] = a[i];
 else b[p2++] = a[i];
 }
 for(int i = L; i <= R; ++i) a[i] = b[i];

 cdq(L, mid);

 // 下凸包, 上凸包则需要改成 slope() <= slope()
 vector<int> q(R - L + 3);

```

```

int head = 1, tail = 0;
for(int i = L; i <= mid; ++i) {
 while(head < tail && slope(q[tail - 1], q[tail]) >= slope(q[tail], i)) --
tail;
 q[++tail] = i;
}

// 下凸包, 上凸包则需要改成 slope() >= K(), 同时f[pos] = max(f[pos], B ...)
for(int i = mid + 1; i <= R; ++i) {
 while(head < tail && slope(q[head], q[head + 1]) <= K(i)) ++head;
 ll B = Y(q[head]) - K(i) * X(q[head]);
 int pos = a[i].id;
 // f[pos] = min(f[pos], B ...);
}

cdq(mid + 1, R);

merge(L, mid, R);
}

void solve() {
 int n;
 cin >> n;
 for(int i = 1; i <= n; ++i) {
 // a[i].id = i;
 // a[i].k = 2 * h[i];
 }

 // 下凸包, 上凸包修改为 x.k > y.k;
 sort(a.begin() + 1, a.begin() + n + 1, [&](node &x, node &y){
 return x.k < y.k;
 });

 f[1] = 0; // 视情况而初始化
 cdq(1, n);
 cout << f[n];
}

```

# 计算几何

## 二维计算几何基础

```

//#include <bits/stdc++.h>
//
//using namespace std;
//#define IOS ios::sync_with_stdio(false),cin.tie(nullptr),cout.tie(nullptr);
//#define int long long
//
//
template<class T>
struct Point {
 T x;
 T y;
}

```

```

Point(T x_ = 0, T y_ = 0) : x(x_), y(y_) {}

template<class U>
operator Point<U>() {
 return Point<U>(U(x), U(y));
}

Point &operator+=(Point p) &{
 x += p.x;
 y += p.y;
 return *this;
}

Point &operator-=(Point p) &{
 x -= p.x;
 y -= p.y;
 return *this;
}

Point &operator*=(T v) &{
 x *= v;
 y *= v;
 return *this;
}

Point operator-() const {
 return Point(-x, -y);
}

friend Point operator+(Point a, Point b) {
 return a += b;
}

friend Point operator-(Point a, Point b) {
 return a -= b;
}

friend Point operator*(Point a, T b) {
 return a *= b;
}

friend Point operator*(T a, Point b) {
 return b *= a;
}

friend bool operator==(Point a, Point b) {
 return a.x == b.x && a.y == b.y;
}

friend std::istream &operator>>(std::istream &is, Point &p) {
 return is >> p.x >> p.y;
}

friend std::ostream &operator<<(std::ostream &os, Point p) {
 return os << "(" << p.x << ", " << p.y << ")";
}

```

```

};

//点乘
template<class T>
T dot(Point<T> a, Point<T> b) {
 return a.x * b.x + a.y * b.y;
}

//叉乘
template<class T>
T cross(Point<T> a, Point<T> b) {
 return a.x * b.y - a.y * b.x;
}

//template<class T>
////ca 与 cb 叉乘
//T cross(Point<T> a, Point<T> b, Point<T> c) {
// Point<T> pa = {b.x - a.x, b.y - a.y};
// Point<T> pb = {c.x - a.x, c.y - b.y};
// return cross(pa, pb);
//}

//点到原点距离的平方
template<class T>
T square(Point<T> p) {
 return dot(p, p);
}

//点到原点距离
template<class T>
double length(Point<T> p) {
 return std::sqrt(double(square(p)));
}

long double length(Point<long double> p) {
 return std::sqrt(square(p));
}

//斜率
template<class T>
double slope(Point<T> p) {
 return (double) p.y / (double) p.x;
}

long double slope(Point<long double> p) {
 return (double) p.y / (double) p.x;
}

template<class T>
Point<T> rotate(Point<T> a) {
 return Point(-a.y, a.x);
} // 逆时针旋转90°

```

```

template<class T>
int sgn(Point<T> a) {
 return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
}

template<class T>
int Quadrant(Point<T> a) {
 //象限排序, 注意包含四个坐标轴
 if (a.x > 0 && a.y >= 0) return 1;
 if (a.x <= 0 && a.y > 0) return 2;
 if (a.x < 0 && a.y <= 0) return 3;
 if (a.x >= 0 && a.y < 0) return 4;
}

//极角序
template<class T>
bool cmp(Point<T> a, Point<T> b) {
 Point<T> c(0, 0); //原点
 if (cross(c, a, b) == 0) //计算叉积, 函数在上面有介绍, 如果叉积相等, 按照x从小到大排序
 return a.x < b.x;
 else return cross(c, a, b) > 0;
}

template<class T>
struct Line {
 Point<T> a;
 Point<T> b;

 Line(Point<T> a_ = Point<T>(), Point<T> b_ = Point<T>()) : a(a_), b(b_) {}
};

template<class T>
Point<T> getprojection(Line<T> l, Point<T> c) {
 auto a = l.a;
 auto b = l.b;
 if (a == b) {
 return a;
 }
 long double x1 = a.x, x2 = b.x, x0 = c.x, y1 = a.y, y2 = b.y, y0 = c.y;
 long double k = -((x1 - x0) * (x2 - x1) + (y1 - y0) * (y2 - y1)) / ((x1 - x2)
* (x1 - x2) + (y1 - y2) * (y1 - y2));
 long double xf = k * (x2 - x1) + x1;
 long double yf = k * (y2 - y1) + y1;
 return Point<T>(xf, yf);
}

template<class T>
Point<T> getreflection(Line<T> l, Point<T> c) {
 auto pf = getprojection(l, c);
 long double xf = pf.x;
 long double yf = pf.y;
 return Point<T>(2 * xf - c.x, 2 * yf - c.y);
}

```

```

template<class T>
Point<T> lineIntersection(Line<T> l1, Line<T> l2) {
 return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b -
l2.a, l1.a - l1.b));
}

template<class T>
bool pointOnSegment(Point<T> p, Line<T> l) {
 return cross(p - l.a, l.b - l.a) == 0 && std::min(l.a.x, l.b.x) <= p.x && p.x
<= std::max(l.a.x, l.b.x)
 && std::min(l.a.y, l.b.y) <= p.y && p.y <= std::max(l.a.y, l.b.y);
}

template<class T>
bool pointInPolygon(Point<T> a, std::vector<Point<T>> p) {
 int n = p.size();
 for (int i = 0; i < n; i++) {
 if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
 return true;
 }
 }
 //先检查是否边上

 int t = 0;
 for (int i = 0; i < n; i++) {
 auto u = p[i];
 auto v = p[(i + 1) % n];
 if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
 t ^= 1;
 }
 if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
 t ^= 1;
 }
 }

 return t == 1;
}

// 0 : not intersect
// 1 : strictly intersect
// 2 : overlap
// 3 : intersect at endpoint
template<class T>
std::tuple<int, Point<T>, Point<T>> segmentIntersection(Line<T> l1, Line<T> l2) {
 if (std::max(l1.a.x, l1.b.x) < std::min(l2.a.x, l2.b.x)) {
 return {0, Point<T>(), Point<T>()};
 }
 if (std::min(l1.a.x, l1.b.x) > std::max(l2.a.x, l2.b.x)) {
 return {0, Point<T>(), Point<T>()};
 }
 if (std::max(l1.a.y, l1.b.y) < std::min(l2.a.y, l2.b.y)) {
 return {0, Point<T>(), Point<T>()};
 }
 if (std::min(l1.a.y, l1.b.y) > std::max(l2.a.y, l2.b.y)) {
 return {0, Point<T>(), Point<T>()};
 }

```

```

}
if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
 if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
 return {0, Point<T>(), Point<T>()};
 } else {
 auto maxx1 = std::max(l1.a.x, l1.b.x);
 auto minx1 = std::min(l1.a.x, l1.b.x);
 auto maxy1 = std::max(l1.a.y, l1.b.y);
 auto miny1 = std::min(l1.a.y, l1.b.y);
 auto maxx2 = std::max(l2.a.x, l2.b.x);
 auto minx2 = std::min(l2.a.x, l2.b.x);
 auto maxy2 = std::max(l2.a.y, l2.b.y);
 auto miny2 = std::min(l2.a.y, l2.b.y);
 Point<T> p1(std::max(minx1, minx2), std::max(miny1, miny2));
 Point<T> p2(std::min(maxx1, maxx2), std::min(maxy1, maxy2));
 if (!pointOnSegment(p1, l1)) {
 std::swap(p1.y, p2.y);
 }
 if (p1 == p2) {
 return {3, p1, p2};
 } else {
 return {2, p1, p2};
 }
 }
}

auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);

if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) ||
 (cp3 < 0 && cp4 < 0)) {
 return {0, Point<T>(), Point<T>()};
}

Point p = lineIntersection(l1, l2);
if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
 return {1, p, p};
} else {
 return {3, p, p};
}
}

template<class T>
bool segmentInPolygon(Line<T> l, std::vector<Point<T>> p) {
 int n = p.size();
 if (!pointInPolygon(l.a, p)) {
 return false;
 }
 if (!pointInPolygon(l.b, p)) {
 return false;
 }
 for (int i = 0; i < n; i++) {
 auto u = p[i];
 auto v = p[(i + 1) % n];
 auto w = p[(i + 2) % n];
 }
}

```

```

auto [t, p1, p2] = segmentIntersection(l, Line(u, v));

if (t == 1) {
 return false;
}
if (t == 0) {
 continue;
}
if (t == 2) {
 if (pointOnSegment(v, l) && v != l.a && v != l.b) {
 if (cross(v - u, w - v) > 0) {
 return false;
 }
 }
} else {
 if (p1 != u && p1 != v) {
 if (pointOnLineLeft(l.a, Line(v, u))
 || pointOnLineLeft(l.b, Line(v, u))) {
 return false;
 }
 } else if (p1 == v) {
 if (l.a == v) {
 if (pointOnLineLeft(u, l)) {
 if (pointOnLineLeft(w, l)
 && pointOnLineLeft(w, Line(u, v))) {
 return false;
 }
 } else {
 if (pointOnLineLeft(w, l)
 || pointOnLineLeft(w, Line(u, v))) {
 return false;
 }
 }
 } else if (l.b == v) {
 if (pointOnLineLeft(u, Line(l.b, l.a))) {
 if (pointOnLineLeft(w, Line(l.b, l.a))
 && pointOnLineLeft(w, Line(u, v))) {
 return false;
 }
 } else {
 if (pointOnLineLeft(w, Line(l.b, l.a))
 || pointOnLineLeft(w, Line(u, v))) {
 return false;
 }
 }
 }
 } else {
 if (pointOnLineLeft(u, l)) {
 if (pointOnLineLeft(w, Line(l.b, l.a))
 || pointOnLineLeft(w, Line(u, v))) {
 return false;
 }
 } else {
 if (pointOnLineLeft(w, l)
 || pointOnLineLeft(w, Line(u, v))) {
 return false;
 }
 }
 }
}

```



```

 }
 }
 }
}

return true;
}

using Vec = Point<int>; //注意类型
//using Vec = Point<double>

//
//template<class T>
/////半平面交
//std::vector<Point<T>> hp(std::vector<Line<T>> lines) {
// std::sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
// auto d1 = l1.b - l1.a;
// auto d2 = l2.b - l2.a;
//
// if (sgn(d1) != sgn(d2)) {
// return sgn(d1) == 1;
// }
//
// return cross(d1, d2) > 0;
// });
//
// std::deque<Line<T>> ls;
// std::deque<Point<T>> ps;
// for (auto l: lines) {
// if (ls.empty()) {
// ls.push_back(l);
// continue;
// }
//
// while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
// ps.pop_back();
// ls.pop_back();
// }
//
// while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
// ps.pop_front();
// ls.pop_front();
// }
//
// if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
// if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
//
// if (!pointOnLineLeft(ls.back().a, l)) {
// assert(ls.size() == 1);
// ls[0] = l;
// }
// continue;
// }
// return {};
// }
// }

```

```

//
// ps.push_back(lineIntersection(ls.back(), l));
// ls.push_back(l);
// }
//
// while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
// ps.pop_back();
// ls.pop_back();
// }
// if (ls.size() <= 2) {
// return {};
// }
// ps.push_back(lineIntersection(ls[0], ls.back()));
//
// return std::vector(ps.begin(), ps.end());
//}
template<class T>
struct Frac {
 T num;
 T den;
 Frac(T num_, T den_) : num(num_), den(den_) {
 if (den < 0) {
 den = -den;
 num = -num;
 }
 }
 Frac() : Frac(0, 1) {}
 Frac(T num_) : Frac(num_, 1) {}
 explicit operator double() const {
 return 1. * num / den;
 }
 Frac &operator+=(const Frac &rhs) {
 num = num * rhs.den + rhs.num * den;
 den *= rhs.den;
 return *this;
 }
 Frac &operator-=(const Frac &rhs) {
 num = num * rhs.den - rhs.num * den;
 den *= rhs.den;
 return *this;
 }
 Frac &operator*=(const Frac &rhs) {
 num *= rhs.num;
 den *= rhs.den;
 return *this;
 }
 Frac &operator/=(const Frac &rhs) {
 num *= rhs.den;
 den *= rhs.num;
 if (den < 0) {
 num = -num;
 den = -den;
 }
 return *this;
 }
}
friend Frac operator+(Frac lhs, const Frac &rhs) {

```

```

 return lhs += rhs;
 }
 friend Frac operator-(Frac lhs, const Frac &rhs) {
 return lhs -= rhs;
 }
 friend Frac operator*(Frac lhs, const Frac &rhs) {
 return lhs *= rhs;
 }
 friend Frac operator/(Frac lhs, const Frac &rhs) {
 return lhs /= rhs;
 }
 friend Frac operator-(const Frac &a) {
 return Frac(-a.num, a.den);
 }
 friend bool operator==(const Frac &lhs, const Frac &rhs) {
 return lhs.num * rhs.den == rhs.num * lhs.den;
 }
 friend bool operator!=(const Frac &lhs, const Frac &rhs) {
 return lhs.num * rhs.den != rhs.num * lhs.den;
 }
 friend bool operator<(const Frac &lhs, const Frac &rhs) {
 return lhs.num * rhs.den < rhs.num * lhs.den;
 }
 friend bool operator>(const Frac &lhs, const Frac &rhs) {
 return lhs.num * rhs.den > rhs.num * lhs.den;
 }
 friend bool operator<=(const Frac &lhs, const Frac &rhs) {
 return lhs.num * rhs.den <= rhs.num * lhs.den;
 }
 friend bool operator>=(const Frac &lhs, const Frac &rhs) {
 return lhs.num * rhs.den >= rhs.num * lhs.den;
 }
 friend std::ostream &operator<<(std::ostream &os, Frac x) {
 T g = std::gcd(x.num, x.den);
 if (x.den == g) {
 return os << x.num / g;
 } else {
 return os << x.num / g << "/" << x.den / g;
 }
 }
};

```

## 凸包

```

struct Point {
 i64 x;
 i64 y;
 Point(i64 x = 0, i64 y = 0) : x(x), y(y) {}
};

bool operator==(const Point &a, const Point &b) {
 return a.x == b.x && a.y == b.y;
}

```

```

Point operator+(const Point &a, const Point &b) {
 return Point(a.x + b.x, a.y + b.y);
}

Point operator-(const Point &a, const Point &b) {
 return Point(a.x - b.x, a.y - b.y);
}

i64 dot(const Point &a, const Point &b) {
 return a.x * b.x + a.y * b.y;
}

i64 cross(const Point &a, const Point &b) {
 return a.x * b.y - a.y * b.x;
}

void norm(std::vector<Point> &h) {
 int i = 0;
 for (int j = 0; j < int(h.size()); j++) {
 if (h[j].y < h[i].y || (h[j].y == h[i].y && h[j].x < h[i].x)) {
 i = j;
 }
 }
 std::rotate(h.begin(), h.begin() + i, h.end());
}

int sgn(const Point &a) {
 return a.y > 0 || (a.y == 0 && a.x > 0) ? 0 : 1;
}

std::vector<Point> getHull(std::vector<Point> p) {
 std::vector<Point> h, l;
 std::sort(p.begin(), p.end(), [&](auto a, auto b) {
 if (a.x != b.x) {
 return a.x < b.x;
 } else {
 return a.y < b.y;
 }
 });
 p.erase(std::unique(p.begin(), p.end()), p.end());
 if (p.size() <= 1) {
 return p;
 }

 for (auto a : p) {
 while (h.size() > 1 && cross(a - h.back(), a - h[h.size() - 2]) <= 0) {
 h.pop_back();
 }
 while (l.size() > 1 && cross(a - l.back(), a - l[l.size() - 2]) >= 0) {
 l.pop_back();
 }
 l.push_back(a);
 h.push_back(a);
 }

 l.pop_back();

```

```
std::reverse(h.begin(), h.end());
h.pop_back();
l.insert(l.end(), h.begin(), h.end());
return l;
}
```



