XCPC Algorithm Template (II)

Jianly's Online version

v8.5 2024.06.01

WIDA

目录

1 杂类

1.1 int128 输出流自定义

```cpp
using i128 = __int128;

std::ostream &operator<<(std::ostream &os, i128 n) {
    std::string s;
    while (n) {
        s += '0' + n % 10;
        n /= 10;
    }
    std::reverse(s.begin(), s.end());
    return os << s;
}
```

1.2 常用库函数重载

```cpp
using i64 = long long;
using i128 = __int128;

i64 ceilDiv(i64 n, i64 m) {
    if (n >= 0) {
        return (n + m - 1) / m;
    } else {
        return n / m;
    }
}

i64 floorDiv(i64 n, i64 m) {
    if (n >= 0) {
        return n / m;
    } else {
        return (n - m + 1) / m;
    }
}

template<class T>
void chmax(T &a, T b) {
    if (a < b) {
        a = b;
    }
}

i128 gcd(i128 a, i128 b) {
    return b ? gcd(b, a % b) : a;
}
```

/END/

2 图与网络

2.1 强连通分量缩点（SCC）

```cpp
struct SCC {
    int n;
    std::vector<std::vector<int>> adj;
    std::vector<int> stk;
    std::vector<int> dfn, low, bel;
    int cur, cnt;

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

    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        dfn.assign(n, -1);
        low.resize(n);
        bel.assign(n, -1);
        stk.clear();
        cur = cnt = 0;
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
    }

    void dfs(int x) {
        dfn[x] = low[x] = cur++;
        stk.push_back(x);

        for (auto y : adj[x]) {
            if (dfn[y] == -1) {
                dfs(y);
                low[x] = std::min(low[x], low[y]);
            } else if (bel[y] == -1) {
                low[x] = std::min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
            cnt++;
        }
    }

    std::vector<int> work() {
        for (int i = 0; i < n; i++) {
            if (dfn[i] == -1) {
                dfs(i);
            }
        }
        return bel;
    }
```

```
};
```

2.2 割边与割边缩点（EBCC）

```cpp
std::set<std::pair<int, int>> E;

struct EBCC {
    int n;
    std::vector<std::vector<int>> adj;
    std::vector<int> stk;
    std::vector<int> dfn, low, bel;
    int cur, cnt;

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

    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        dfn.assign(n, -1);
        low.resize(n);
        bel.assign(n, -1);
        stk.clear();
        cur = cnt = 0;
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void dfs(int x, int p) {
        dfn[x] = low[x] = cur++;
        stk.push_back(x);

        for (auto y : adj[x]) {
            if (y == p) {
                continue;
            }
            if (dfn[y] == -1) {
                E.emplace(x, y);
                dfs(y, x);
                low[x] = std::min(low[x], low[y]);
            } else if (bel[y] == -1 && dfn[y] < dfn[x]) {
                E.emplace(x, y);
                low[x] = std::min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
            cnt++;
        }
    }

```

```
    std::vector<int> work() {
        dfs(0, -1);
        return bel;
    }

    struct Graph {
        int n;
        std::vector<std::pair<int, int>> edges;
        std::vector<int> siz;
        std::vector<int> cnte;
    };
    Graph compress() {
        Graph g;
        g.n = cnt;
        g.siz.resize(cnt);
        g.cnte.resize(cnt);
        for (int i = 0; i < n; i++) {
            g.siz[bel[i]]++;
            for (auto j : adj[i]) {
                if (bel[i] < bel[j]) {
                    g.edges.emplace_back(bel[i], bel[j]);
                } else if (i < j) {
                    g.cnte[bel[i]]++;
                }
            }
        }
        return g;
    }
};
```

2.3 二分图最大权匹配（MaxAssignment 基于KM）【久远】

```
template<class T>
struct MaxAssignment {
    public:
        T solve(int nx, int ny, std::vector<std::vector<T>> a) {
            assert(0 <= nx && nx <= ny);
            assert(int(a.size()) == nx);
            for (int i = 0; i < nx; ++i) {
                assert(int(a[i].size()) == ny);
                for (auto x : a[i])
                    assert(x >= 0);
            }

            auto update = [&](int x) {
                for (int y = 0; y < ny; ++y) {
                    if (lx[x] + ly[y] - a[x][y] < slack[y]) {
                        slack[y] = lx[x] + ly[y] - a[x][y];
                        slackx[y] = x;
                    }
                }
            };

            costs.resize(nx + 1);
            costs[0] = 0;
            lx.assign(nx, std::numeric_limits<T>::max());
            ly.assign(ny, 0);
            xy.assign(nx, -1);
            yx.assign(ny, -1);
            slackx.resize(ny);
            for (int cur = 0; cur < nx; ++cur) {
                std::queue<int> que;
```

```
                visx.assign(nx, false);
                visy.assign(ny, false);
                slack.assign(ny, std::numeric_limits<T>::max());
                p.assign(nx, -1);

                for (int x = 0; x < nx; ++x) {
                    if (xy[x] == -1) {
                        que.push(x);
                        visx[x] = true;
                        update(x);
                    }
                }

                int ex, ey;
                bool found = false;
                while (!found) {
                    while (!que.empty() && !found) {
                        auto x = que.front();
                        que.pop();
                        for (int y = 0; y < ny; ++y) {
                            if (a[x][y] == lx[x] + ly[y] && !visy[y]) {
                                if (yx[y] == -1) {
                                    ex = x;
                                    ey = y;
                                    found = true;
                                    break;
                                }
                                que.push(yx[y]);
                                p[yx[y]] = x;
                                visy[y] = visx[yx[y]] = true;
                                update(yx[y]);
                            }
                        }
                    }
                    if (found)
                        break;

                    T delta = std::numeric_limits<T>::max();
                    for (int y = 0; y < ny; ++y)
                        if (!visy[y])
                            delta = std::min(delta, slack[y]);
                    for (int x = 0; x < nx; ++x)
                        if (visx[x])
                            lx[x] -= delta;
                    for (int y = 0; y < ny; ++y) {
                        if (visy[y]) {
                            ly[y] += delta;
                        } else {
                            slack[y] -= delta;
                        }
                    }
                    for (int y = 0; y < ny; ++y) {
                        if (!visy[y] && slack[y] == 0) {
                            if (yx[y] == -1) {
                                ex = slackx[y];
                                ey = y;
                                found = true;
                                break;
                            }
                            que.push(yx[y]);
                            p[yx[y]] = slackx[y];
                            visy[y] = visx[yx[y]] = true;
                            update(yx[y]);
                        }
```

```
                    }
                }

                costs[cur + 1] = costs[cur];
                for (int x = ex, y = ey, ty; x != -1; x = p[x], y = ty) {
                    costs[cur + 1] += a[x][y];
                    if (xy[x] != -1)
                        costs[cur + 1] -= a[x][xy[x]];
                    ty = xy[x];
                    xy[x] = y;
                    yx[y] = x;
                }
            }
            return costs[nx];
        }
        std::vector<int> assignment() {
            return xy;
        }
        std::pair<std::vector<T>, std::vector<T>> labels() {
            return std::make_pair(lx, ly);
        }
        std::vector<T> weights() {
            return costs;
        }
    private:
        std::vector<T> lx, ly, slack, costs;
        std::vector<int> xy, yx, p, slackx;
        std::vector<bool> visx, visy;
};
```

2.4 一般图最大匹配（Graph 带花树算法）【久远】

```
struct Graph {
    int n;
    std::vector<std::vector<int>> e;
    Graph(int n) : n(n), e(n) {}
    void addEdge(int u, int v) {
        e[u].push_back(v);
        e[v].push_back(u);
    }
    std::vector<int> findMatching() {
        std::vector<int> match(n, -1), vis(n), link(n), f(n), dep(n);

        // disjoint set union
        auto find = [&](int u) {
            while (f[u] != u)
                u = f[u] = f[f[u]];
            return u;
        };

        auto lca = [&](int u, int v) {
            u = find(u);
            v = find(v);
            while (u != v) {
                if (dep[u] < dep[v])
                    std::swap(u, v);
                u = find(link[match[u]]);
            }
            return u;
        };

        std::queue<int> que;
```

```
        auto blossom = [&](int u, int v, int p) {
            while (find(u) != p) {
                link[u] = v;
                v = match[u];
                if (vis[v] == 0) {
                    vis[v] = 1;
                    que.push(v);
                }
                f[u] = f[v] = p;
                u = link[v];
            }
        };

        // find an augmenting path starting from u and augment (if exist)
        auto augment = [&](int u) {

            while (!que.empty())
                que.pop();

            std::iota(f.begin(), f.end(), 0);

            // vis = 0 corresponds to inner vertices, vis = 1 corresponds to outer vertices
            std::fill(vis.begin(), vis.end(), -1);

            que.push(u);
            vis[u] = 1;
            dep[u] = 0;

            while (!que.empty()){
                int u = que.front();
                que.pop();
                for (auto v : e[u]) {
                    if (vis[v] == -1) {

                        vis[v] = 0;
                        link[v] = u;
                        dep[v] = dep[u] + 1;

                        // found an augmenting path
                        if (match[v] == -1) {
                            for (int x = v, y = u, temp; y != -1; x = temp, y = x == -1 ? -1 : link[x]) {
                                temp = match[y];
                                match[x] = y;
                                match[y] = x;
                            }
                            return;
                        }

                        vis[match[v]] = 1;
                        dep[match[v]] = dep[u] + 2;
                        que.push(match[v]);

                    } else if (vis[v] == 1 && find(v) != find(u)) {
                        // found a blossom
                        int p = lca(u, v);
                        blossom(u, v, p);
                        blossom(v, u, p);
                    }
                }
            }

        };
```

```

        // find a maximal matching greedily (decrease constant)
        auto greedy = [&]() {

            for (int u = 0; u < n; ++u) {
                if (match[u] != -1)
                    continue;
                for (auto v : e[u]) {
                    if (match[v] == -1) {
                        match[u] = v;
                        match[v] = u;
                        break;
                    }
                }
            }
        };

        greedy();

        for (int u = 0; u < n; ++u)
            if (match[u] == -1)
                augment(u);

        return match;
    }
};
```

2.5 TwoSat (2-Sat)

```
struct TwoSat {
    int n;
    std::vector<std::vector<int>> e;
    std::vector<bool> ans;
    TwoSat(int n) : n(n), e(2 * n), ans(n) {}
    void addClause(int u, bool f, int v, bool g) {
        e[2 * u + !f].push_back(2 * v + g);
        e[2 * v + !g].push_back(2 * u + f);
    }
    bool satisfiable() {
        std::vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
        std::vector<int> stk;
        int now = 0, cnt = 0;
        std::function<void(int)> tarjan = [&](int u) {
            stk.push_back(u);
            dfn[u] = low[u] = now++;
            for (auto v : e[u]) {
                if (dfn[v] == -1) {
                    tarjan(v);
                    low[u] = std::min(low[u], low[v]);
                } else if (id[v] == -1) {
                    low[u] = std::min(low[u], dfn[v]);
                }
            }
            if (dfn[u] == low[u]) {
                int v;
                do {
                    v = stk.back();
                    stk.pop_back();
                    id[v] = cnt;
                } while (v != u);
                ++cnt;
            }
```

```
        };
        for (int i = 0; i < 2 * n; ++i) if (dfn[i] == -1) tarjan(i);
        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;
    }
    std::vector<bool> answer() { return ans; }
};
```

2.6 最大流

2.6.1 最大流（Flow 旧版其一，整数应用）

```
template<class T>
struct Flow {
    const int n;
    struct Edge {
        int to;
        T cap;
        Edge(int to, T cap) : to(to), cap(cap) {}
    };
    std::vector<Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<int> cur, h;
    Flow(int n) : n(n), g(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 maxFlow(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;
    }
};
```

2.6.2 最大流（Flow 旧版其二，浮点数应用）

```
template<class T>
struct Flow {
    const int n;
    struct Edge {
        int to;
        T cap;
        Edge(int to, T cap) : to(to), cap(cap) {}
    };
    std::vector<Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<int> cur, h;
    Flow(int n) : n(n), g(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;
        double res = 0;
        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));
                res += a;
                e[j].cap -= a;
                e[j ^ 1].cap += a;
                r -= a;
                if (r == 0) {
                    return f;
                }
            }
        }
        return res;
    }
    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 maxFlow(int s, int t) {
        T ans = 0;
        while (bfs(s, t)) {
            cur.assign(n, 0);
            ans += dfs(s, t, 1E100);
        }
        return ans;
    }
};
```

2.6.3 最大流（MaxFlow 新版）

```
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;
    }
};
```

2.7 费用流

2.7.1 费用流（MCFGraph 最小费用可行流）

```
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) {
        if (f < 0) {
            g[u].push_back(e.size());
            e.emplace_back(v, 0, f);
            g[v].push_back(e.size());
            e.emplace_back(u, c, -f);
        } else {
            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);
    }
};
```

2.7.2 费用流（MCFGraph 最小费用最大流）

代码同上，但是需要注释掉建边限制。以下为参考：

```
void addEdge(int u, int v, int c, int f) { // 可行流
    if (f < 0) {
        g[u].push_back(e.size());
        e.emplace_back(v, 0, f);
        g[v].push_back(e.size());
        e.emplace_back(u, c, -f);
    } else {
        g[u].push_back(e.size());
        e.emplace_back(v, c, f);
        g[v].push_back(e.size());
        e.emplace_back(u, 0, -f);
    }
}
```

```
void addEdge(int u, int v, int c, int f) { // 最大流
    g[u].push_back(e.size());
    e.emplace_back(v, c, f);
    g[v].push_back(e.size());
    e.emplace_back(u, 0, -f);
}
```

2.8 树链剖分（HLD）

```cpp
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 isAncester(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 (!isAncester(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 (!isAncester(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);
    }
};
```

/END/

3 数论、几何、多项式

3.1 快速幂

```cpp
int power(int a, i64 b, int p) {
    int res = 1;
    for (; b; b /= 2, a = 1LL * a * a % p) {
        if (b % 2) {
            res = 1LL * res * a % p;
        }
    }
    return res;
}
```

3.2 欧拉筛

```cpp
std::vector<int> minp, primes;

void sieve(int n) {
    minp.assign(n + 1, 0);
    primes.clear();

    for (int i = 2; i <= n; i++) {
        if (minp[i] == 0) {
            minp[i] = i;
            primes.push_back(i);
        }

        for (auto p : primes) {
            if (i * p > n) {
                break;
            }
            minp[i * p] = p;
            if (p == minp[i]) {
                break;
            }
        }
    }
}
```

3.3 莫比乌斯函数筛（莫比乌斯函数/反演）

```cpp
std::unordered_map<int, Z> fMu;

constexpr int N = 1E7;
std::vector<int> minp, primes;
std::vector<Z> mu;

void sieve(int n) {
    minp.assign(n + 1, 0);
    mu.resize(n);
    primes.clear();

    mu[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (minp[i] == 0) {
            mu[i] = -1;
            minp[i] = i;
            primes.push_back(i);
```

```
        }

        for (auto p : primes) {
            if (i * p > n) {
                break;
            }
            minp[i * p] = p;
            if (p == minp[i]) {
                break;
            }
            mu[i * p] = -mu[i];
        }
    }

    for (int i = 1; i <= n; i++) {
        mu[i] += mu[i - 1];
    }
}


Z sumMu(int n) {
    if (n <= N) {
        return mu[n];
    }
    if (fMu.count(n)) {
        return fMu[n];
    }
    if (n == 0) {
        return 0;
    }
    Z ans = 1;
    for (int l = 2, r; l <= n; l = r + 1) {
        r = n / (n / l);
        ans -= (r - l + 1) * sumMu(n / l);
    }
    return ans;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    sieve(N);

    int L, R;
    std::cin >> L >> R;
    L -= 1;

    Z ans = 0;
    for (int l = 1, r; l <= R; l = r + 1) {
        r = R / (R / l);
        if (l <= L) {
            r = std::min(r, L / (L / l));
        }

        ans += (power(Z(2), R / l - L / l) - 1) * (sumMu(r) - sumMu(l - 1));
    }

    std::cout << ans << "\n";

    return 0;
}
```

3.4 求解单个数的欧拉函数

```cpp
int phi(int n) {
    int res = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0) {
                n /= i;
            }
            res = res / i * (i - 1);
        }
    }
    if (n > 1) {
        res = res / n * (n - 1);
    }
    return res;
}
```

3.5 扩展欧几里得（exGCD）

```cpp
int exgcd(int a, int b, int &x, int &y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    int g = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return g;
}
```

3.6 组合数（Comb, with. MInt & MLong）

```cpp
struct Comb {
    int n;
    std::vector<Z> _fac;
    std::vector<Z> _invfac;
    std::vector<Z> _inv;

    Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
    Comb(int n) : Comb() {
        init(n);
    }

    void init(int m) {
        m = std::min(m, Z::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;
    }
```

```
    Z fac(int m) {
        if (m > n) init(2 * m);
        return _fac[m];
    }
    Z invfac(int m) {
        if (m > n) init(2 * m);
        return _invfac[m];
    }
    Z inv(int m) {
        if (m > n) init(2 * m);
        return _inv[m];
    }
    Z binom(int n, int m) {
        if (n < m || m < 0) return 0;
        return fac(n) * invfac(m) * invfac(n - m);
    }
} comb;
```

3.7 二项式（Binomial 任意模数计算）

```
std::vector<std::pair<int, int>> factorize(int n) {
    std::vector<std::pair<int, int>> factors;
    for (int i = 2; static_cast<long long>(i) * i <= n; i++) {
        if (n % i == 0) {
            int t = 0;
            for (; n % i == 0; n /= i)
                ++t;
            factors.emplace_back(i, t);
        }
    }
    if (n > 1)
        factors.emplace_back(n, 1);
    return factors;
}
constexpr int power(int base, i64 exp) {
    int res = 1;
    for (; exp > 0; base *= base, exp /= 2) {
        if (exp % 2 == 1) {
            res *= base;
        }
    }
    return res;
}
constexpr int power(int base, i64 exp, int mod) {
    int res = 1 % mod;
    for (; exp > 0; base = 1LL * base * base % mod, exp /= 2) {
        if (exp % 2 == 1) {
            res = 1LL * res * base % mod;
        }
    }
    return res;
}
int inverse(int a, int m) {
    int g = m, r = a, x = 0, y = 1;
    while (r != 0) {
        int q = g / r;
        g %= r;
        std::swap(g, r);
        x -= q * y;
        std::swap(x, y);
    }
```

```
    return x < 0 ? x + m : x;
}
int solveModuloEquations(const std::vector<std::pair<int, int>> &e) {
    int m = 1;
    for (std::size_t i = 0; i < e.size(); i++) {
        m *= e[i].first;
    }
    int res = 0;
    for (std::size_t i = 0; i < e.size(); i++) {
        int p = e[i].first;
        res = (res + 1LL * e[i].second * (m / p) * inverse(m / p, p)) % m;
    }
    return res;
}
constexpr int N = 1E5;
class Binomial {
    const int mod;
private:
    const std::vector<std::pair<int, int>> factors;
    std::vector<int> pk;
    std::vector<std::vector<int>> prod;
    static constexpr i64 exponent(i64 n, int p) {
        i64 res = 0;
        for (n /= p; n > 0; n /= p) {
            res += n;
        }
        return res;
    }
    int product(i64 n, std::size_t i) {
        int res = 1;
        int p = factors[i].first;
        for (; n > 0; n /= p) {
            res = 1LL * res * power(prod[i].back(), n / pk[i], pk[i]) % pk[i] *
prod[i][n % pk[i]] % pk[i];
        }
        return res;
    }
public:
    Binomial(int mod) : mod(mod), factors(factorize(mod)) {
        pk.resize(factors.size());
        prod.resize(factors.size());
        for (std::size_t i = 0; i < factors.size(); i++) {
            int p = factors[i].first;
            int k = factors[i].second;
            pk[i] = power(p, k);
            prod[i].resize(std::min(N + 1, pk[i]));
            prod[i][0] = 1;
            for (int j = 1; j < prod[i].size(); j++) {
                if (j % p == 0) {
                    prod[i][j] = prod[i][j - 1];
                } else {
                    prod[i][j] = 1LL * prod[i][j - 1] * j % pk[i];
                }
            }
        }
    }
    int operator()(i64 n, i64 m) {
        if (n < m || m < 0) {
            return 0;
        }
        std::vector<std::pair<int, int>> ans(factors.size());
        for (int i = 0; i < factors.size(); i++) {
            int p = factors[i].first;
            int k = factors[i].second;
```

```
            int e = exponent(n, p) - exponent(m, p) - exponent(n - m, p);
            if (e >= k) {
                ans[i] = std::make_pair(pk[i], 0);
            } else {
                int pn = product(n, i);
                int pm = product(m, i);
                int pd = product(n - m, i);
                int res = 1LL * pn * inverse(pm, pk[i]) % pk[i] * inverse(pd,
pk[i]) % pk[i] * power(p, e) % pk[i];
                ans[i] = std::make_pair(pk[i], res);
            }
        }
        return solveModuloEquations(ans);
    }
};
```

3.8 素数测试与因式分解 (Miller-Rabin & Pollard-Rho)

```
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; b >>= 1, a = mul(a, a, m))
        if (b & 1)
            res = mul(res, a, m);
    return res;
}
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;
}
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;
        }
        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;
}
```

3.9 平面几何

```
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 << ")";
    }
};

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

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

template<class T>
T square(Point<T> p) {
    return dot(p, p);
}

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));
}

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

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

template<class T>
Point<T> rotate(Point<T> a) {
    return Point(-a.y, a.x);
}

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

template<class T>
bool pointOnLineLeft(Point<T> p, Line<T> l) {
    return cross(l.b - l.a, p - l.a) > 0;
}

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;
        }
    }

    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;
}

```

```cpp
template<class T>
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());
}
```

3.10 静态凸包

3.10.1 静态凸包 (with. Point)

```cpp
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;
}
```

3.10.2 静态凸包 (with. std::complex)

```
using Point = std::complex<i64>;

#define x real
#define y imag

auto dot(const Point &a, const Point &b) {
    return (std::conj(a) * b).x();
}

auto cross(const Point &a, const Point &b) {
    return (std::conj(a) * b).y();
}

auto rot(const Point &p) {
    return Point(-p.y(), p.x());
}

auto complexHull(std::vector<Point> a) {
    std::sort(a.begin(), a.end(), [&](auto a, auto b) {
        if (a.x() != b.x()) {
            return a.x() < b.x();
        } else {
            return a.y() < b.y();
        }
    });

    std::vector<Point> l, h;

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

        while (h.size() > 1 && cross(h.back() - h[h.size() - 2], p - h.back()) >= 0)
{
            h.pop_back();
        }

        l.push_back(p);
        h.push_back(p);
    }

    std::reverse(h.begin(), h.end());

    h.insert(h.end(), l.begin() + 1, l.end() - 1);

```

```
    return h;
}

int sgn(Point p) {
    if (p.y() > 0 || (p.y() == 0 && p.x() < 0)) {
        return 0;
    } else {
        return 1;
    }
}
```

3.11 多项式相关

3.11.1 多项式相关 (Poly, with. Z)

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

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

    for (int i = 0; i < n; i++) {
        if (rev[i] < i) {
            std::swap(a[i], a[rev[i]]);
        }
    }
    if (int(roots.size()) < n) {
        int k = __builtin_ctz(roots.size());
        roots.resize(n);
        while ((1 << k) < n) {
            Z e = power(Z(3), (P - 1) >> (k + 1));
            for (int i = 1 << (k - 1); i < (1 << k); i++) {
                roots[2 * i] = roots[i];
                roots[2 * i + 1] = roots[i] * e;
            }
            k++;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                Z u = a[i + j];
                Z v = a[i + j + k] * roots[k + j];
                a[i + j] = u + v;
                a[i + j + k] = u - v;
            }
        }
    }
}
void idft(std::vector<Z> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    Z inv = (1 - P) / n;
    for (int i = 0; i < n; i++) {
```

```
        a[i] *= inv;
    }
}
struct Poly {
    std::vector<Z> a;
    Poly() {}
    explicit Poly(int size, std::function<Z(int)> f = [](int) { return 0; }) :
a(size) {
        for (int i = 0; i < size; i++) {
            a[i] = f(i);
        }
    }
    Poly(const std::vector<Z> &a) : a(a) {}
    Poly(const std::initializer_list<Z> &a) : a(a) {}
    int size() const {
        return a.size();
    }
    void resize(int n) {
        a.resize(n);
    }
    Z operator[](int idx) const {
        if (idx < size()) {
            return a[idx];
        } else {
            return 0;
        }
    }
    Z &operator[](int idx) {
        return a[idx];
    }
    Poly mulxk(int k) const {
        auto b = a;
        b.insert(b.begin(), k, 0);
        return Poly(b);
    }
    Poly modxk(int k) const {
        k = std::min(k, size());
        return Poly(std::vector<Z>(a.begin(), a.begin() + k));
    }
    Poly divxk(int k) const {
        if (size() <= k) {
            return Poly();
        }
        return Poly(std::vector<Z>(a.begin() + k, a.end()));
    }
    friend Poly operator+(const Poly &a, const Poly &b) {
        std::vector<Z> res(std::max(a.size(), b.size()));
        for (int i = 0; i < int(res.size()); i++) {
            res[i] = a[i] + b[i];
        }
        return Poly(res);
    }
    friend Poly operator-(const Poly &a, const Poly &b) {
        std::vector<Z> res(std::max(a.size(), b.size()));
        for (int i = 0; i < int(res.size()); i++) {
            res[i] = a[i] - b[i];
        }
        return Poly(res);
    }
    friend Poly operator-(const Poly &a) {
        std::vector<Z> res(a.size());
        for (int i = 0; i < int(res.size()); i++) {
            res[i] = -a[i];
        }
```

```
        return Poly(res);
    }
    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);
        }
        if (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;
        }
        int sz = 1, tot = a.size() + b.size() - 1;
        while (sz < tot) {
            sz *= 2;
        }
        a.a.resize(sz);
        b.a.resize(sz);
        dft(a.a);
        dft(b.a);
        for (int i = 0; i < sz; ++i) {
            a.a[i] = a[i] * b[i];
        }
        idft(a.a);
        a.resize(tot);
        return a;
    }
    friend Poly operator*(Z a, Poly b) {
        for (int i = 0; i < int(b.size()); i++) {
            b[i] *= a;
        }
        return b;
    }
    friend Poly operator*(Poly a, Z b) {
        for (int i = 0; i < int(a.size()); i++) {
            a[i] *= b;
        }
        return a;
    }
    Poly &operator+=(Poly b) {
        return (*this) = (*this) + b;
    }
    Poly &operator-=(Poly b) {
        return (*this) = (*this) - b;
    }
    Poly &operator*=(Poly b) {
        return (*this) = (*this) * b;
    }
    Poly &operator*=(Z b) {
        return (*this) = (*this) * b;
    }
    Poly deriv() const {
        if (a.empty()) {
            return Poly();
        }
        std::vector<Z> res(size() - 1);
        for (int i = 0; i < size() - 1; ++i) {
            res[i] = (i + 1) * a[i + 1];
```

```
        }
        return Poly(res);
    }
    Poly integr() const {
        std::vector<Z> res(size() + 1);
        for (int i = 0; i < size(); ++i) {
            res[i + 1] = a[i] / (i + 1);
        }
        return Poly(res);
    }
    Poly inv(int m) const {
        Poly x{a[0].inv()};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly{2} - modxk(k) * x)).modxk(k);
        }
        return x.modxk(m);
    }
    Poly log(int m) const {
        return (deriv() * inv(m)).integr().modxk(m);
    }
    Poly exp(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly{1} - x.log(k) + modxk(k))).modxk(k);
        }
        return x.modxk(m);
    }
    Poly pow(int k, int m) const {
        int i = 0;
        while (i < size() && a[i].val() == 0) {
            i++;
        }
        if (i == size() || 1LL * i * k >= m) {
            return Poly(std::vector<Z>(m));
        }
        Z v = a[i];
        auto f = divxk(i) * v.inv();
        return (f.log(m - i * k) * k).exp(m - i * k).mulxk(i * k) * power(v, k);
    }
    Poly sqrt(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2);
        }
        return x.modxk(m);
    }
    Poly mulT(Poly b) const {
        if (b.size() == 0) {
            return Poly();
        }
        int n = b.size();
        std::reverse(b.a.begin(), b.a.end());
        return ((*this) * b).divxk(n - 1);
    }
    std::vector<Z> eval(std::vector<Z> x) const {
        if (size() == 0) {
            return std::vector<Z>(x.size(), 0);
        }
```

```
        const int n = std::max(int(x.size()), size());
        std::vector<Poly> q(4 * n);
        std::vector<Z> 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;
                work(2 * p, l, m, num.mulT(q[2 * p + 1]).modxk(m - l));
                work(2 * p + 1, m, r, num.mulT(q[2 * p]).modxk(r - m));
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
    }
};
```

3.12 多项式相关 (Poly, with. MInt & MLong)

```
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 = 998244353>
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;
    }
    constexpr Poly deriv() const {
        if (this->empty()) {
            return Poly();
        }
        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 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 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;
                work(2 * p, l, m, num.mulT(q[2 * p + 1]).resize(m - l));
                work(2 * p + 1, m, r, num.mulT(q[2 * p]).resize(r - m));
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
    }
};

template<int P = 998244353>
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;
```

```cpp
}


template<int P = 998244353>
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];
}

struct Comb {
    int n;
    std::vector<Z> _fac;
    std::vector<Z> _invfac;
    std::vector<Z> _inv;

    Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
    Comb(int n) : Comb() {
        init(n);
    }

    void init(int m) {
        m = std::min(m, Z::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;
    }

    Z fac(int m) {
        if (m > n) init(2 * m);
        return _fac[m];
    }
    Z invfac(int m) {
        if (m > n) init(2 * m);
        return _invfac[m];
    }
    Z inv(int m) {
        if (m > n) init(2 * m);
        return _inv[m];
```

```
    }
    Z binom(int n, int m) {
        if (n < m || m < 0) return 0;
        return fac(n) * invfac(m) * invfac(n - m);
    }
} comb;

Poly<P> get(int n, int m) {
    if (m == 0) {
        return Poly(n + 1);
    }
    if (m % 2 == 1) {
        auto f = get(n, m - 1);
        Z p = 1;
        for (int i = 0; i <= n; i++) {
            f[n - i] += comb.binom(n, i) * p;
            p *= m;
        }
        return f;
    }
    auto f = get(n, m / 2);
    auto fm = f;
    for (int i = 0; i <= n; i++) {
        fm[i] *= comb.fac(i);
    }
    Poly pw(n + 1);
    pw[0] = 1;
    for (int i = 1; i <= n; i++) {
        pw[i] = pw[i - 1] * (m / 2);
    }
    for (int i = 0; i <= n; i++) {
        pw[i] *= comb.invfac(i);
    }
    fm = fm.mulT(pw);
    for (int i = 0; i <= n; i++) {
        fm[i] *= comb.invfac(i);
    }
    return f + fm;
}
```

/END/

4 数据结构

4.1 树状数组（Fenwick 新版）

```cpp
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) {
        for (int i = x + 1; i <= n; i += i & -i) {
            a[i - 1] = a[i - 1] + v;
        }
    }

    T sum(int x) {
        T ans{};
        for (int i = x; i > 0; i -= i & -i) {
            ans = ans + a[i - 1];
        }
        return ans;
    }

    T rangeSum(int l, int r) {
        return sum(r) - sum(l);
    }

    int select(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;
    }
};
```

4.2 并查集（DSU）

```cpp
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)];
    }
};
```

4.3 线段树

4.3.1 线段树（SegmentTree 基础区间加乘）

```
struct SegmentTree {
    int n;
    std::vector<int> tag, sum;
    SegmentTree(int n_) : n(n_), tag(4 * n, 1), sum(4 * n) {}

    void pull(int p) {
        sum[p] = (sum[2 * p] + sum[2 * p + 1]) % P;
    }

    void mul(int p, int v) {
        tag[p] = 1LL * tag[p] * v % P;
        sum[p] = 1LL * sum[p] * v % P;
    }

    void push(int p) {
        mul(2 * p, tag[p]);
        mul(2 * p + 1, tag[p]);
        tag[p] = 1;
    }

    int query(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return 0;
        }
        if (l >= x && r <= y) {
            return sum[p];
```

```
        }
        int m = (l + r) / 2;
        push(p);
        return (query(2 * p, l, m, x, y) + query(2 * p + 1, m, r, x, y)) % P;
    }

    int query(int x, int y) {
        return query(1, 0, n, x, y);
    }

    void rangeMul(int p, int l, int r, int x, int y, int v) {
        if (l >= y || r <= x) {
            return;
        }
        if (l >= x && r <= y) {
            return mul(p, v);
        }
        int m = (l + r) / 2;
        push(p);
        rangeMul(2 * p, l, m, x, y, v);
        rangeMul(2 * p + 1, m, r, x, y, v);
        pull(p);
    }

    void rangeMul(int x, int y, int v) {
        rangeMul(1, 0, n, x, y, v);
    }

    void add(int p, int l, int r, int x, int v) {
        if (r - l == 1) {
            sum[p] = (sum[p] + v) % P;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            add(2 * p, l, m, x, v);
        } else {
            add(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }

    void add(int x, int v) {
        add(1, 0, n, x, v);
    }
};
```

4.3.2 线段树（SegmentTree+Info 查找前驱后继）

```
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);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    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;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    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 >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
        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);
    }
};
struct Info {
    int cnt = 0;
    i64 sum = 0;
    i64 ans = 0;
};
Info operator+(Info a, Info b) {
    Info c;
    c.cnt = a.cnt + b.cnt;
    c.sum = a.sum + b.sum;
    c.ans = a.ans + b.ans + a.cnt * b.sum - a.sum * b.cnt;
    return c;
}
```

4.3.3 线段树（SegmentTree+Info+Merge 区间合并）

```
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);
    };
    build(1, 0, n);
}
void pull(int p) {
    info[p] = info[2 * p] + info[2 * p + 1];
}
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;
    if (x < m) {
        modify(2 * p, l, m, x, v);
    } else {
        modify(2 * p + 1, m, r, x, v);
    }
    pull(p);
}
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 >= y || r <= x) {
        return Info();
    }
    if (l >= x && r <= y) {
        return info[p];
    }
    int m = (l + r) / 2;
    return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
}
Info rangeQuery(int l, int r) {
    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);
    }
};

struct Info {
    int x = 0;
    int cnt = 0;
};

Info operator+(Info a, Info b) {
    if (a.x == b.x) {
        return {a.x, a.cnt + b.cnt};
    } else if (a.cnt > b.cnt) {
        return {a.x, a.cnt - b.cnt};
    } else {
        return {b.x, b.cnt - a.cnt};
    }
}
```

4.4 懒标记线段树

4.4.1 懒标记线段树（LazySegmentTree 基础区间修改）

```
template<class Info, class Tag>
struct LazySegmentTree {
    const int n;
    std::vector<Info> info;
    std::vector<Tag> tag;
    LazySegmentTree(int n) : n(n), info(4 << std::__lg(n)), tag(4 << std::__lg(n))
{}
    LazySegmentTree(std::vector<Info> init) : LazySegmentTree(init.size()) {
        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);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void apply(int p, const Tag &v) {
        info[p].apply(v);
```

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

```
        push(p);
        half(2 * p, l, m);
        half(2 * p + 1, m, r);
        pull(p);
    }
    void half() {
        half(1, 0, n);
    }
};

constexpr i64 inf = 1E18;

struct Tag {
    i64 add = 0;

    void apply(Tag t) {
        add += t.add;
    }
};

struct Info {
    i64 min = inf;
    i64 max = -inf;
    i64 sum = 0;
    i64 act = 0;

    void apply(Tag t) {
        min += t.add;
        max += t.add;
        sum += act * t.add;
    }
};

Info operator+(Info a, Info b) {
    Info c;
    c.min = std::min(a.min, b.min);
    c.max = std::max(a.max, b.max);
    c.sum = a.sum + b.sum;
    c.act = a.act + b.act;
    return c;
}
```

4.4.2 懒标记线段树（LazySegmentTree 查找前驱后继）

```
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(4 << std::__lg(n), Info());
        tag.assign(4 << std::__lg(n), 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);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void apply(int p, const Tag &v) {
        info[p].apply(v);
        tag[p].apply(v);
    }
    void push(int p) {
        apply(2 * p, tag[p]);
        apply(2 * p + 1, tag[p]);
        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);
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    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 >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        push(p);
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
        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;
        }
        if (l >= x && r <= y) {
            apply(p, v);
```

```
            return;
        }
        int m = (l + r) / 2;
        push(p);
        rangeApply(2 * p, l, m, x, y, v);
        rangeApply(2 * p + 1, m, r, x, y, v);
        pull(p);
    }
    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);
        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);
        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);
    }
};

struct Tag {
    i64 a = 0, b = 0;
    void apply(Tag t) {
        a = std::min(a, b + t.a);
        b += t.b;
    }
};

int k;

struct Info {
```

```
    i64 x = 0;
    void apply(Tag t) {
        x += t.a;
        if (x < 0) {
            x = (x % k + k) % k;
        }
        x += t.b - t.a;
    }
};
Info operator+(Info a, Info b) {
    return {a.x + b.x};
}
```

4.4.3 懒标记线段树（LazySegmentTree 二分修改）

```
constexpr int inf = 1E9 + 1;
template<class Info, class Tag>
struct LazySegmentTree {
    const int n;
    std::vector<Info> info;
    std::vector<Tag> tag;
    LazySegmentTree(int n) : n(n), info(4 << std::__lg(n)), tag(4 << std::__lg(n))
{}
    LazySegmentTree(std::vector<Info> init) : LazySegmentTree(init.size()) {
        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);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void apply(int p, const Tag &v) {
        info[p].apply(v);
        tag[p].apply(v);
    }
    void push(int p) {
        apply(2 * p, tag[p]);
        apply(2 * p + 1, tag[p]);
        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);
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
```

```
    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 >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        push(p);
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
        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;
        }
        if (l >= x && r <= y) {
            apply(p, v);
            return;
        }
        int m = (l + r) / 2;
        push(p);
        rangeApply(2 * p, l, m, x, y, v);
        rangeApply(2 * p + 1, m, r, x, y, v);
        pull(p);
    }
    void rangeApply(int l, int r, const Tag &v) {
        return rangeApply(1, 0, n, l, r, v);
    }
    void maintainL(int p, int l, int r, int pre) {
        if (info[p].difl > 0 && info[p].maxlowl < pre) {
            return;
        }
        if (r - l == 1) {
            info[p].max = info[p].maxlowl;
            info[p].maxl = info[p].maxr = l;
            info[p].maxlowl = info[p].maxlowr = -inf;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        maintainL(2 * p, l, m, pre);
        pre = std::max(pre, info[2 * p].max);
        maintainL(2 * p + 1, m, r, pre);
        pull(p);
    }
    void maintainL() {
        maintainL(1, 0, n, -1);
    }
    void maintainR(int p, int l, int r, int suf) {
        if (info[p].difr > 0 && info[p].maxlowr < suf) {
            return;
        }
        if (r - l == 1) {
            info[p].max = info[p].maxlowl;
            info[p].maxl = info[p].maxr = l;
            info[p].maxlowl = info[p].maxlowr = -inf;
            return;
        }
```

```
        int m = (l + r) / 2;
        push(p);
        maintainR(2 * p + 1, m, r, suf);
        suf = std::max(suf, info[2 * p + 1].max);
        maintainR(2 * p, l, m, suf);
        pull(p);
    }
    void maintainR() {
        maintainR(1, 0, n, -1);
    }
};

struct Tag {
    int add = 0;

    void apply(Tag t) & {
        add += t.add;
    }
};

struct Info {
    int max = -1;
    int maxl = -1;
    int maxr = -1;
    int difl = inf;
    int difr = inf;
    int maxlowl = -inf;
    int maxlowr = -inf;

    void apply(Tag t) & {
        if (max != -1) {
            max += t.add;
        }
        difl += t.add;
        difr += t.add;
    }
};

Info operator+(Info a, Info b) {
    Info c;
    if (a.max > b.max) {
        c.max = a.max;
        c.maxl = a.maxl;
        c.maxr = a.maxr;
    } else if (a.max < b.max) {
        c.max = b.max;
        c.maxl = b.maxl;
        c.maxr = b.maxr;
    } else {
        c.max = a.max;
        c.maxl = a.maxl;
        c.maxr = b.maxr;
    }

    c.difl = std::min(a.difl, b.difl);
    c.difr = std::min(a.difr, b.difr);
    if (a.max != -1) {
        c.difl = std::min(c.difl, a.max - b.maxlowl);
    }
    if (b.max != -1) {
        c.difr = std::min(c.difr, b.max - a.maxlowr);
    }

    if (a.max == -1) {
```

```
        c.maxlowl = std::max(a.maxlowl, b.maxlowl);
    } else {
        c.maxlowl = a.maxlowl;
    }
    if (b.max == -1) {
        c.maxlowr = std::max(a.maxlowr, b.maxlowr);
    } else {
        c.maxlowr = b.maxlowr;
    }
    return c;
}
```

4.5 取模类

4.5.1 取模类 (MLong & MInt)

```
constexpr int P = 998244353;
using i64 = long long;
// assume -P <= x < 2P
int norm(int x) {
    if (x < 0) {
        x += P;
    }
    if (x >= P) {
        x -= P;
    }
    return x;
}
template<class T>
T power(T a, i64 b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}
struct Z {
    int x;
    Z(int x = 0) : x(norm(x)) {}
    Z(i64 x) : x(norm(x % P)) {}
    int val() const {
        return x;
    }
    Z operator-() const {
        return Z(norm(P - x));
    }
    Z inv() const {
        assert(x != 0);
        return power(*this, P - 2);
    }
    Z &operator*=(const Z &rhs) {
        x = i64(x) * rhs.x % P;
        return *this;
    }
    Z &operator+=(const Z &rhs) {
        x = norm(x + rhs.x);
        return *this;
    }
    Z &operator-=(const Z &rhs) {
        x = norm(x - rhs.x);
```

```
        return *this;
    }
    Z &operator/=(const Z &rhs) {
        return *this *= rhs.inv();
    }
    friend Z operator*(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res *= rhs;
        return res;
    }
    friend Z operator+(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res += rhs;
        return res;
    }
    friend Z operator-(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res -= rhs;
        return res;
    }
    friend Z operator/(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res /= rhs;
        return res;
    }
    friend std::istream &operator>>(std::istream &is, Z &a) {
        i64 v;
        is >> v;
        a = Z(v);
        return is;
    }
    friend std::ostream &operator<<(std::ostream &os, const Z &a) {
        return os << a.val();
    }
};
```

4.5.2 取模类（MLong & MInt 新版）

根据输入内容动态修改 MOD 的方法：`Z::setMod(p);` 。

```
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<0LL>::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;
    }
    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();

```

```cpp
constexpr int P = 1000000007;
using Z = MInt<P>;
```

4.6 状压RMQ（RMQ）

```cpp
template<class T,
    class Cmp = std::less<T>>
struct RMQ {
    const Cmp cmp = Cmp();
    static constexpr unsigned B = 64;
    using u64 = unsigned long long;
    int n;
    std::vector<std::vector<T>> a;
    std::vector<T> pre, suf, ini;
    std::vector<u64> stk;
    RMQ() {}
    RMQ(const std::vector<T> &v) {
        init(v);
    }
    void init(const std::vector<T> &v) {
        n = v.size();
        pre = suf = ini = v;
        stk.resize(n);
        if (!n) {
            return;
        }
        const int M = (n - 1) / B + 1;
        const int lg = std::__lg(M);
        a.assign(lg + 1, std::vector<T>(M));
        for (int i = 0; i < M; i++) {
            a[0][i] = v[i * B];
            for (int j = 1; j < B && i * B + j < n; j++) {
                a[0][i] = std::min(a[0][i], v[i * B + j], cmp);
            }
        }
        for (int i = 1; i < n; i++) {
            if (i % B) {
                pre[i] = std::min(pre[i], pre[i - 1], cmp);
            }
        }
        for (int i = n - 2; i >= 0; i--) {
            if (i % B != B - 1) {
                suf[i] = std::min(suf[i], suf[i + 1], cmp);
            }
        }
        for (int j = 0; j < lg; j++) {
            for (int i = 0; i + (2 << j) <= M; i++) {
                a[j + 1][i] = std::min(a[j][i], a[j][i + (1 << j)], cmp);
            }
        }
        for (int i = 0; i < M; i++) {
            const int l = i * B;
            const int r = std::min(1U * n, l + B);
            u64 s = 0;
            for (int j = l; j < r; j++) {
                while (s && cmp(v[j], v[std::__lg(s) + l])) {
                    s ^= 1ULL << std::__lg(s);
                }
                s |= 1ULL << (j - l);
                stk[j] = s;
            }
        }
```

```
    }
    T operator()(int l, int r) {
        if (l / B != (r - 1) / B) {
            T ans = std::min(suf[l], pre[r - 1], cmp);
            l = l / B + 1;
            r = r / B;
            if (l < r) {
                int k = std::__lg(r - l);
                ans = std::min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
            }
            return ans;
        } else {
            int x = B * (l / B);
            return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + l];
        }
    }
};
```

4.6.1 Splay

```
struct Node {
    Node *l = nullptr;
    Node *r = nullptr;
    int cnt = 0;
    i64 sum = 0;
};

Node *add(Node *t, int l, int r, int p, int v) {
    Node *x = new Node;
    if (t) {
        *x = *t;
    }
    x->cnt += 1;
    x->sum += v;
    if (r - l == 1) {
        return x;
    }
    int m = (l + r) / 2;
    if (p < m) {
        x->l = add(x->l, l, m, p, v);
    } else {
        x->r = add(x->r, m, r, p, v);
    }
    return x;
}

int find(Node *tl, Node *tr, int l, int r, int x) {
    if (r <= x) {
        return -1;
    }
    if (l >= x) {
        int cnt = (tr ? tr->cnt : 0) - (tl ? tl->cnt : 0);
        if (cnt == 0) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
    }
    int m = (l + r) / 2;
    int res = find(tl ? tl->l : tl, tr ? tr->l : tr, l, m, x);
    if (res == -1) {
```

```
        res = find(tl ? tl->r : tl, tr ? tr->r : tr, m, r, x);
    }
    return res;
}

std::pair<int, i64> get(Node *t, int l, int r, int x, int y) {
    if (l >= y || r <= x || !t) {
        return {0, 0LL};
    }
    if (l >= x && r <= y) {
        return {t->cnt, t->sum};
    }
    int m = (l + r) / 2;
    auto [cl, sl] = get(t->l, l, m, x, y);
    auto [cr, sr] = get(t->r, m, r, x, y);
    return {cl + cr, sl + sr};
}

struct Tree {
    int add = 0;
    int val = 0;
    int id = 0;
    Tree *ch[2] = {};
    Tree *p = nullptr;
};

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

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

void push(Tree *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(Tree *t) {
    Tree *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(Tree *t) {
    std::vector<Tree *> s;
    for (Tree *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(Tree *&t, Tree *x, Tree *p = nullptr) {
    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(Tree *t) {
    if (!t) {
        return;
    }
    push(t);
    dfs(t->ch[0]);
    std::cerr << t->val << " ";
    dfs(t->ch[1]);
}

std::pair<Tree *, Tree *> split(Tree *t, int x) {
    if (!t) {
        return {t, t};
    }
    Tree *v = nullptr;
    Tree *j = t;
    for (Tree *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, nullptr};
    }

    splay(v);

    Tree *u = v->ch[0];
    if (u) {
        v->ch[0] = u->p = nullptr;
    }
    // std::cerr << "split " << x << "\n";
    // dfs(u);
```

```
    // std::cerr << "\n";
    // dfs(v);
    // std::cerr << "\n";
    return {u, v};
}

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

```
struct Node {
    Node *ch[2], *p;
    bool rev;
    int siz = 1;
    Node() : ch{nullptr, nullptr}, p(nullptr), rev(false) {}
};
void reverse(Node *t) {
    if (t) {
        std::swap(t->ch[0], t->ch[1]);
        t->rev ^= 1;
    }
}
void push(Node *t) {
    if (t->rev) {
        reverse(t->ch[0]);
        reverse(t->ch[1]);
        t->rev = false;
    }
}
void pull(Node *t) {
    t->siz = (t->ch[0] ? t->ch[0]->siz : 0) + 1 + (t->ch[1] ? t->ch[1]->siz : 0);
}
bool isroot(Node *t) {
    return t->p == nullptr || (t->p->ch[0] != t && t->p->ch[1] != t);
}
int pos(Node *t) {
    return t->p->ch[1] == t;
}
void pushAll(Node *t) {
    if (!isroot(t)) {
        pushAll(t->p);
    }
    push(t);
}
void rotate(Node *t) {
    Node *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 (!isroot(q)) {
        q->p->ch[pos(q)] = t;
    }
    t->ch[x] = q;
    q->p = t;
    pull(q);
}
void splay(Node *t) {
    pushAll(t);
    while (!isroot(t)) {
        if (!isroot(t->p)) {
            if (pos(t) == pos(t->p)) {
                rotate(t->p);
            } else {
                rotate(t);
            }
        }
        rotate(t);
    }
    pull(t);
}
void access(Node *t) {
    for (Node *i = t, *q = nullptr; i; q = i, i = i->p) {
        splay(i);
        i->ch[1] = q;
        pull(i);
    }
    splay(t);
}
void makeroot(Node *t) {
    access(t);
    reverse(t);
}
void link(Node *x, Node *y) {
    makeroot(x);
    x->p = y;
}
void split(Node *x, Node *y) {
    makeroot(x);
    access(y);
}
void cut(Node *x, Node *y) {
    split(x, y);
    x->p = y->ch[0] = nullptr;
    pull(y);
}
int dist(Node *x, Node *y) {
    split(x, y);
    return y->siz - 1;
}
```

```
struct Matrix : std::array<std::array<i64, 4>, 4> {
    Matrix(i64 v = 0) {
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                (*this)[i][j] = (i == j ? v : inf);
            }
        }
    }
};

```

```cpp
Matrix operator*(const Matrix &a, const Matrix &b) {
    Matrix c(inf);
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 4; k++) {
                c[i][k] = std::min(c[i][k], a[i][j] + b[j][k]);
            }
        }
        c[i][3] = std::min(c[i][3], a[i][3]);
    }
    c[3][3] = 0;
    return c;
}

struct Node {
    Node *ch[2], *p;
    i64 sumg = 0;
    i64 sumh = 0;
    i64 sumb = 0;
    i64 g = 0;
    i64 h = 0;
    i64 b = 0;
    Matrix mat;
    Matrix prd;
    std::array<i64, 4> ans{};
    Node() : ch{nullptr, nullptr}, p(nullptr) {}

    void update() {
        mat = Matrix(inf);
        mat[0][0] = b + h - g + sumg;
        mat[1][1] = mat[1][2] = mat[1][3] = h + sumh;
        mat[2][0] = mat[2][1] = mat[2][2] = mat[2][3] = b + h + sumb;
        mat[3][3] = 0;
    }
};
void push(Node *t) {

}
void pull(Node *t) {
    t->prd = (t->ch[0] ? t->ch[0]->prd : Matrix()) * t->mat * (t->ch[1] ? t->ch[1]->prd : Matrix());
}
bool isroot(Node *t) {
    return t->p == nullptr || (t->p->ch[0] != t && t->p->ch[1] != t);
}
int pos(Node *t) {
    return t->p->ch[1] == t;
}
void pushAll(Node *t) {
    if (!isroot(t)) {
        pushAll(t->p);
    }
    push(t);
}
void rotate(Node *t) {
    Node *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 (!isroot(q)) {
        q->p->ch[pos(q)] = t;
```

```
    }
    t->ch[x] = q;
    q->p = t;
    pull(q);
}
void splay(Node *t) {
    pushAll(t);
    while (!isroot(t)) {
        if (!isroot(t->p)) {
            if (pos(t) == pos(t->p)) {
                rotate(t->p);
            } else {
                rotate(t);
            }
        }
        rotate(t);
    }
    pull(t);
}

std::array<i64, 4> get(Node *t) {
    std::array<i64, 4> ans;
    ans.fill(inf);
    ans[3] = 0;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
            ans[i] = std::min(ans[i], t->prd[i][j]);
        }
    }
    return ans;
}

void access(Node *t) {
    std::array<i64, 4> old{};
    for (Node *i = t, *q = nullptr; i; q = i, i = i->p) {
        splay(i);
        if (i->ch[1]) {
            auto res = get(i->ch[1]);
            i->sumg += res[0];
            i->sumh += std::min({res[1], res[2], res[3]});
            i->sumb += std::min({res[0], res[1], res[2], res[3]});
        }
        i->ch[1] = q;
        i->sumg -= old[0];
        i->sumh -= std::min({old[1], old[2], old[3]});
        i->sumb -= std::min({old[0], old[1], old[2], old[3]});
        old = get(i);
        i->update();
        pull(i);
    }
    splay(t);
}
```

4.7 其他平衡树

```
struct Node {
    Node *l = nullptr;
    Node *r = nullptr;
    int sum = 0;
    int sumodd = 0;

    Node(Node *t) {
```

```
        if (t) {
            *this = *t;
        }
    }
};

Node *add(Node *t, int l, int r, int x, int v) {
    t = new Node(t);
    t->sum += v;
    t->sumodd += (x % 2) * v;
    if (r - l == 1) {
        return t;
    }
    int m = (l + r) / 2;
    if (x < m) {
        t->l = add(t->l, l, m, x, v);
    } else {
        t->r = add(t->r, m, r, x, v);
    }
    return t;
}

int query1(Node *t1, Node *t2, int l, int r, int k) {
    if (r - l == 1) {
        return l;
    }
    int m = (l + r) / 2;
    int odd = (t1 && t1->r ? t1->r->sumodd : 0) - (t2 && t2->r ? t2->r->sumodd : 0);
    int cnt = (t1 && t1->r ? t1->r->sum : 0) - (t2 && t2->r ? t2->r->sum : 0);
    if (odd > 0 || cnt > k) {
        return query1(t1 ? t1->r : t1, t2 ? t2->r : t2, m, r, k);
    } else {
        return query1(t1 ? t1->l : t1, t2 ? t2->l : t2, l, m, k - cnt);
    }
}

std::array<int, 3> query2(Node *t1, Node *t2, int l, int r, int k) {
    if (r - l == 1) {
        int cnt = (t1 ? t1->sumodd : 0) - (t2 ? t2->sumodd : 0);
        return {l, cnt, k};
    }
    int m = (l + r) / 2;
    int cnt = (t1 && t1->r ? t1->r->sumodd : 0) - (t2 && t2->r ? t2->r->sumodd : 0);
    if (cnt > k) {
        return query2(t1 ? t1->r : t1, t2 ? t2->r : t2, m, r, k);
    } else {
        return query2(t1 ? t1->l : t1, t2 ? t2->l : t2, l, m, k - cnt);
    }
}
```

```
struct Node {
    Node *l = nullptr;
    Node *r = nullptr;
    int cnt = 0;
};

Node *add(Node *t, int l, int r, int x) {
    if (t) {
        t = new Node(*t);
    } else {
        t = new Node;
    }
    t->cnt += 1;
```

```
    if (r - l == 1) {
        return t;
    }
    int m = (l + r) / 2;
    if (x < m) {
        t->l = add(t->l, l, m, x);
    } else {
        t->r = add(t->r, m, r, x);
    }
    return t;
}

int query(Node *t1, Node *t2, int l, int r, int x) {
    int cnt = (t2 ? t2->cnt : 0) - (t1 ? t1->cnt : 0);
    if (cnt == 0 || l >= x) {
        return -1;
    }
    if (r - l == 1) {
        return l;
    }
    int m = (l + r) / 2;
    int res = query(t1 ? t1->r : t1, t2 ? t2->r : t2, m, r, x);
    if (res == -1) {
        res = query(t1 ? t1->l : t1, t2 ? t2->l : t2, l, m, x);
    }
    return res;
}
```

```
struct Info {
    int imp = 0;
    int id = 0;
};

Info operator+(Info a, Info b) {
    return {std::max(a.imp, b.imp), 0};
}

struct Node {
    int w = rng();
    Info info;
    Info sum;
    int siz = 1;
    Node *l = nullptr;
    Node *r = nullptr;
};

void pull(Node *t) {
    t->sum = t->info;
    t->siz = 1;
    if (t->l) {
        t->sum = t->l->sum + t->sum;
        t->siz += t->l->siz;
    }
    if (t->r) {
        t->sum = t->sum + t->r->sum;
        t->siz += t->r->siz;
    }
}

std::pair<Node *, Node *> splitAt(Node *t, int p) {
    if (!t) {
        return {t, t};
    }
```

```
    if (p <= (t->l ? t->l->siz : 0)) {
        auto [l, r] = splitAt(t->l, p);
        t->l = r;
        pull(t);
        return {l, t};
    } else {
        auto [l, r] = splitAt(t->r, p - 1 - (t->l ? t->l->siz : 0));
        t->r = l;
        pull(t);
        return {t, r};
    }
}

void insertAt(Node *&t, int p, Node *x) {
    if (!t) {
        t = x;
        return;
    }
    if (x->w < t->w) {
        auto [l, r] = splitAt(t, p);
        t = x;
        t->l = l;
        t->r = r;
        pull(t);
        return;
    }
    if (p <= (t->l ? t->l->siz : 0)) {
        insertAt(t->l, p, x);
    } else {
        insertAt(t->r, p - 1 - (t->l ? t->l->siz : 0), x);
    }
    pull(t);
}

Node *merge(Node *a, Node *b) {
    if (!a) {
        return b;
    }
    if (!b) {
        return a;
    }

    if (a->w < b->w) {
        a->r = merge(a->r, b);
        pull(a);
        return a;
    } else {
        b->l = merge(a, b->l);
        pull(b);
        return b;
    }
}

int query(Node *t, int v) {
    if (!t) {
        return 0;
    }
    if (t->sum.imp < v) {
        return t->siz;
    }
    int res = query(t->r, v);
    if (res != (t->r ? t->r->siz : 0)) {
        return res;
    }
```

```
    if (t->info.imp > v) {
        return res;
    }
    return res + 1 + query(t->l, v);
}

void dfs(Node *t) {
    if (!t) {
        return;
    }
    dfs(t->l);
    std::cout << t->info.id << " ";
    dfs(t->r);
}
```

```
struct Node {
    Node *l = nullptr;
    Node *r = nullptr;
    int cnt = 0;
    int cntnew = 0;
};

Node *add(int l, int r, int x, int isnew) {
    Node *t = new Node;
    t->cnt = 1;
    t->cntnew = isnew;
    if (r - l == 1) {
        return t;
    }
    int m = (l + r) / 2;
    if (x < m) {
        t->l = add(l, m, x, isnew);
    } else {
        t->r = add(m, r, x, isnew);
    }
    return t;
}

struct Info {
    Node *t = nullptr;
    int psum = 0;
    bool rev = false;
};

void pull(Node *t) {
    t->cnt = (t->l ? t->l->cnt : 0) + (t->r ? t->r->cnt : 0);
    t->cntnew = (t->l ? t->l->cntnew : 0) + (t->r ? t->r->cntnew : 0);
}

std::pair<Node *, Node *> split(Node *t, int l, int r, int x, bool rev) {
    if (!t) {
        return {t, t};
    }
    if (x == 0) {
        return {nullptr, t};
    }
    if (x == t->cnt) {
        return {t, nullptr};
    }
    if (r - l == 1) {
        Node *t2 = new Node;
        t2->cnt = t->cnt - x;
        t->cnt = x;
```

```
        return {t, t2};
    }
    Node *t2 = new Node;
    int m = (l + r) / 2;
    if (!rev) {
        if (t->l && x <= t->l->cnt) {
            std::tie(t->l, t2->l) = split(t->l, l, m, x, rev);
            t2->r = t->r;
            t->r = nullptr;
        } else {
            std::tie(t->r, t2->r) = split(t->r, m, r, x - (t->l ? t->l->cnt : 0),
rev);
        }
    } else {
        if (t->r && x <= t->r->cnt) {
            std::tie(t->r, t2->r) = split(t->r, m, r, x, rev);
            t2->l = t->l;
            t->l = nullptr;
        } else {
            std::tie(t->l, t2->l) = split(t->l, l, m, x - (t->r ? t->r->cnt : 0),
rev);
        }
    }
    pull(t);
    pull(t2);
    return {t, t2};
}

Node *merge(Node *t1, Node *t2, int l, int r) {
    if (!t1) {
        return t2;
    }
    if (!t2) {
        return t1;
    }
    if (r - l == 1) {
        t1->cnt += t2->cnt;
        t1->cntnew += t2->cntnew;
        delete t2;
        return t1;
    }
    int m = (l + r) / 2;
    t1->l = merge(t1->l, t2->l, l, m);
    t1->r = merge(t1->r, t2->r, m, r);
    delete t2;
    pull(t1);
    return t1;
}
```

4.8 分数四则运算（Frac）

```
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;
        }
    }
};
```

4.9 线性基 (Basis)

```
struct Basis {
    int a[20] {};
    int t[20] {};

    Basis() {
        std::fill(t, t + 20, -1);
    }

    void add(int x, int y = 1E9) {
        for (int i = 0; i < 20; i++) {
            if (x >> i & 1) {
                if (y > t[i]) {
                    std::swap(a[i], x);
                    std::swap(t[i], y);
                }
                x ^= a[i];
            }
        }
    }

    bool query(int x, int y = 0) {
        for (int i = 0; i < 20; i++) {
            if ((x >> i & 1) && t[i] >= y) {
                x ^= a[i];
            }
        }
        return x == 0;
    }
};
```

/END/

5 字符串

5.1 马拉车（Manacher 新版）

```cpp
std::vector<int> manacher(std::vector<int> s) {
    std::vector<int> t{0};
    for (auto c : s) {
        t.push_back(c);
        t.push_back(0);
    }
    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;
}
```

5.2 Z函数

```cpp
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::max(0, 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;
}
```

5.3 后缀数组（SA）

```cpp
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;
            }
        }
    }
};
```

5.4 后缀自动机

5.4.1 后缀自动机（SuffixAutomaton 旧版）

```
struct SuffixAutomaton {
    static constexpr int ALPHABET_SIZE = 26, N = 5e5;
    struct Node {
        int len;
        int link;
        int next[ALPHABET_SIZE];
        Node() : len(0), link(0), next{} {}
    } t[2 * N];
    int cntNodes;
    SuffixAutomaton() {
        cntNodes = 1;
        std::fill(t[0].next, t[0].next + ALPHABET_SIZE, 1);
        t[0].len = -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 = ++cntNodes;
            t[r].len = t[p].len + 1;
```

```
            t[r].link = t[q].link;
            std::copy(t[q].next, t[q].next + ALPHABET_SIZE, t[r].next);
            t[q].link = r;
            while (t[p].next[c] == q) {
                t[p].next[c] = r;
                p = t[p].link;
            }
            return r;
        }
        int cur = ++cntNodes;
        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;
    }
};
```

5.4.2 后缀自动机（SAM 新版）

```
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();
    }
};
```

5.5 回文自动机（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;
```

5.6 AC自动机

5.6.1 AC自动机（AC 旧版）

```
constexpr int N = 3e5 + 30, A = 26;

struct Node {
    int fail;
    int sum;
    int next[A];
    Node() : fail(-1), sum(0) {
        std::memset(next, -1, sizeof(next));
    }
} node[N];

int cnt = 0;
int bin[N];
int nBin = 0;

```

```cpp
int newNode() {
    int p = nBin > 0 ? bin[--nBin] : cnt++;
    node[p] = Node();
    return p;
}

struct AC {
    std::vector<int> x;
    AC(AC &&a) : x(std::move(a.x)) {}
    AC(std::vector<std::string> s, std::vector<int> w) {
        x = {newNode(), newNode()};
        std::fill(node[x[0]].next, node[x[0]].next + A, x[1]);
        node[x[1]].fail = x[0];

        for (int i = 0; i < int(s.size()); i++) {
            int p = x[1];
            for (int j = 0; j < int(s[i].length()); j++) {
                int c = s[i][j] - 'a';
                if (node[p].next[c] == -1) {
                    int u = newNode();
                    x.push_back(u);
                    node[p].next[c] = u;
                }
                p = node[p].next[c];
            }
            node[p].sum += w[i];
        }

        std::queue<int> que;
        que.push(x[1]);
        while (!que.empty()) {
            int u = que.front();
            que.pop();
            node[u].sum += node[node[u].fail].sum;
            for (int c = 0; c < A; c++) {
                if (node[u].next[c] == -1) {
                    node[u].next[c] = node[node[u].fail].next[c];
                } else {
                    node[node[u].next[c]].fail = node[node[u].fail].next[c];
                    que.push(node[u].next[c]);
                }
            }
        }
    }
    ~AC() {
        for (auto p : x) {
            bin[nBin++] = p;
        }
    }
    i64 query(const std::string &s) const {
        i64 ans = 0;
        int p = x[1];
        for (int i = 0; i < int(s.length()); i++)  {
            int c = s[i] - 'a';
            p = node[p].next[c];
            ans += node[p].sum;
        }
        return ans;
    }
};
```

5.6.2 AC自动机（AhoCorasick 新新版）

```cpp
struct AhoCorasick {
    static constexpr int ALPHABET = 26;
    struct Node {
        int len;
        int link;
        std::array<int, ALPHABET> next;
        Node() : len{0}, link{0}, next{} {}
    };

    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::string &a) {
        int p = 1;
        for (auto c : a) {
            int x = c - '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];
        }
        return p;
    }

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

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

            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];
                    q.push(t[x].next[i]);
                }
            }
        }
    }

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

```
62
63      int link(int p) {
64          return t[p].link;
65      }
66
67      int len(int p) {
68          return t[p].len;
69      }
70
71      int size() {
72          return t.size();
73      }
74  };
```

5.7 随机生成模底 字符串哈希（例题）

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

using i64 = long long;

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;
}

using Hash = std::array<int, 2>;

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

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

    const int P = findPrime(rng() % 900000000 + 100000000);

    std::string s, x;
    std::cin >> s >> x;

    int n = s.length();
    int m = x.length();

    std::vector<int> h(n + 1), p(n + 1);
    for (int i = 0; i < n; i++) {
        h[i + 1] = (10LL * h[i] + s[i] - '0') % P;
    }
    p[0] = 1;
    for (int i = 0; i < n; i++) {
        p[i + 1] = 10LL * p[i] % P;
```

```cpp
    }

    auto get = [&](int l, int r) {
        return (h[r] + 1LL * (P - h[l]) * p[r - l]) % P;
    };

    int px = 0;
    for (auto c : x) {
        px = (10LL * px + c - '0') % P;
    }

    for (int i = 0; i <= n - 2 * (m - 1); i++) {
        if ((get(i, i + m - 1) + get(i + m - 1, i + 2 * m - 2)) % P == px) {
            std::cout << i + 1 << " " << i + m - 1 << "\n";
            std::cout << i + m << " " << i + 2 * m - 2 << "\n";
            return 0;
        }
    }

    std::vector<int> z(m + 1), f(n + 1);
    z[0] = m;

    for (int i = 1, j = -1; i < m; i++) {
        if (j != -1) {
            z[i] = std::max(0, std::min(j + z[j] - i, z[i - j]));
        }
        while (z[i] + i < m && x[z[i]] == x[z[i] + i]) {
            z[i]++;
        }
        if (j == -1 || i + z[i] > j + z[j]) {
            j = i;
        }
    }
    for (int i = 0, j = -1; i < n; i++) {
        if (j != -1) {
            f[i] = std::max(0, std::min(j + f[j] - i, z[i - j]));
        }
        while (f[i] + i < n && f[i] < m && x[f[i]] == s[f[i] + i]) {
            f[i]++;
        }
        if (j == -1 || i + f[i] > j + f[j]) {
            j = i;
        }
    }

    for (int i = 0; i + m <= n; i++) {
        int l = std::min(m, f[i]);

        for (auto j : { m - l, m - l - 1 }) {
            if (j <= 0) {
                continue;
            }
            if (j <= i && (get(i - j, i) + get(i, i + m)) % P == px) {
                std::cout << i - j + 1 << " " << i << "\n";
                std::cout << i + 1 << " " << i + m << "\n";
                return 0;
            }
            if (i + m + j <= n && (get(i, i + m) + get(i + m, i + m + j)) % P ==
px) {
                std::cout << i + 1 << " " << i + m << "\n";
                std::cout << i + m + 1 << " " << i + m + j << "\n";
                return 0;
            }
        }
```

```
110     }
111
112     return 0;
113 }
```

Edited by *Wida*

Version: 8.5 (2024.06.01)

github.com/hh2048
www.cnblogs.com/WIDA