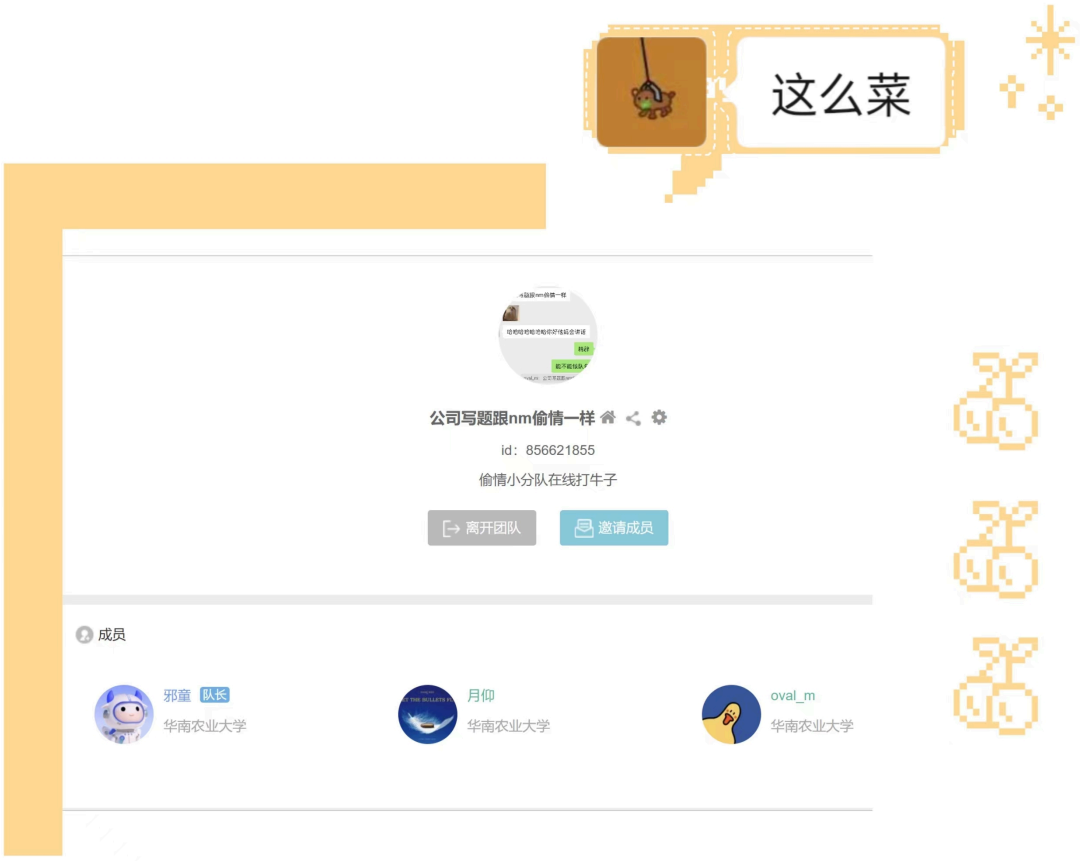
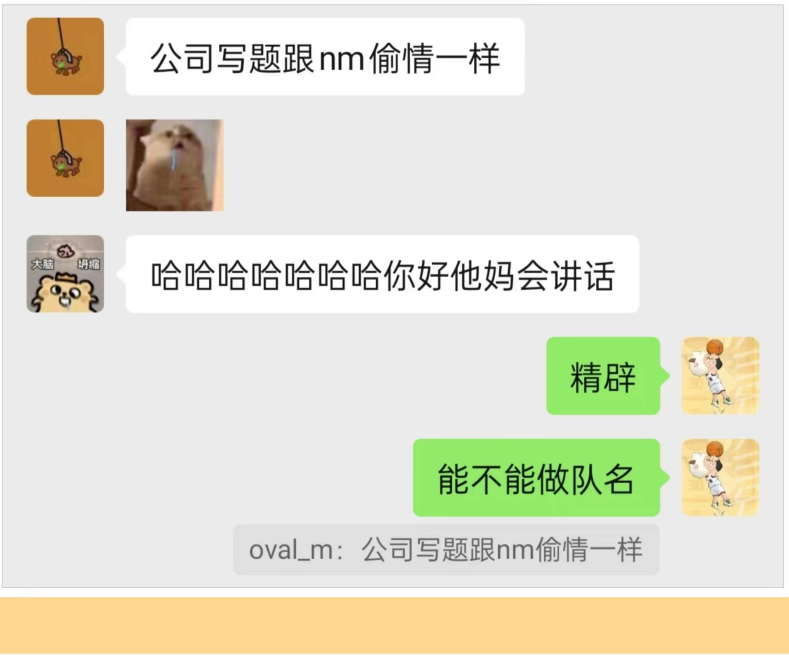
邪童 の 偷情笔记


这么菜
公司写题跟nm偷情一样
id：856621855
偷情小分队在线打牛子
离开团队
邀请成员
成员
邪童 队长
华南农业大学
月仰
华南农业大学
oval_m
华南农业大学

yummy


公司写题跟nm偷情一样
哈哈哈哈哈哈哈你好他妈会讲话
精辟
能不能做队名
oval_m：公司写题跟nm偷情一样

目录

jiangly 板子

    - 04A - 懒标记线段树（LazySegmentTree 基础区间修改）
    - 04B - 懒标记线段树（LazySegmentTree 查找前驱后继）
    - 04C - 懒标记线段树（LazySegmentTree 二分修改）
    - 05A - 取模类（MLong & MInt）
    - 05B - 取模类（MLong & MInt 新版）
    - 06 - 状压RMQ（RMQ）
    - 07 - Splay
    - 08 - 其他平衡树
    - 09 - 分数四则运算（Frac）
    - 10 - 线性基（Basis）
- 五、字符串
    - 01 - 马拉车（Manacher）
    - 02 - Z函数
    - 03 - 后缀数组（SA）
    - 04A - 后缀自动机（SuffixAutomaton 旧版）
    - 04B - 后缀自动机（SAM 新版）
    - 05 - 回文自动机（PAM）
    - 06A - AC自动机（AC 旧版）
    - 06B - AC自动机（AhoCorasick 新版）
    - 07 - 随机生成模底 字符串哈希（例题）

其它

- BigInt
- 点分治
- 树分治
- 网络流封装
- bitset bfs
- 封装 multiset 对顶堆
- Miller-Rabin 大素数判定
- Pollard rho 大整数分解
- 双模哈希封装
- 哈希 + 数据结构
- 树与图上的计数问题
- SA 封装
- 排列组合公式
- Let it Rot

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;
}
```

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;
}
```

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;
    }
};
```

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;
    }
};
```

二分图最大权匹配（MaxAssignment 基于KM）

```cpp
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;
};
```

一般图最大匹配（Graph 带花树算法）

```cpp
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;
    }
};
```

TwoSat（2-Sat）

```cpp
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; }
};
```

最大流（Flow 旧版其一，整数应用）

```cpp
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;
    }
};
```

最大流（Flow 旧版其二，浮点数应用）

```cpp
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;
    }
};
```

最大流（MaxFlow 新版）

```cpp
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);
        }
```

```
109         return a;
110     }
111 };
```

费用流（MCFGraph 最小费用可行流）

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

费用流（MCFGraph 最小费用最大流）

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

```
                e[pre[i]].c -= aug;
                e[pre[i] ^ 1].c += aug;
            }
            flow += aug;
            cost += i64(aug) * h[t];
        }
        return std::make_pair(flow, cost);
    }
};
```

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);
    }
};
```

快速幂

```
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;
}
```

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;
            }
        }
    }
}
```

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);
    }
```

```cpp
    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;
}
```

求解单个数的欧拉函数

```
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;
}
```

扩展欧几里得（exGCD）

```
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;
}
```

组合数（Comb+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;
```

二项式（Binomial 任意模数计算）

```cpp
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;
```

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

素数测试与因式分解（Miller-Rabin & Pollard-Rho）

```cpp
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;
}
```

平面几何

```cpp
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;
    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;
}

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());
}
```

静态凸包

```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;
}
```

多项式相关（Poly 旧版）

```cpp
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;
    }
};
```

多项式相关（Poly+MInt & MLong 新版）

```cpp
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;
}


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++) {
```

```
438             pw[i] = pw[i - 1] * (m / 2);
439         }
440         for (int i = 0; i <= n; i++) {
441             pw[i] *= comb.invfac(i);
442         }
443         fm = fm.multT(pw);
444         for (int i = 0; i <= n; i++) {
445             fm[i] *= comb.invfac(i);
446         }
447         return f + fm;
448     }
```

树状数组（Fenwick 旧版）

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

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

    void init(int n) {
        this->n = n;
        a.assign(n, T());
    }

    void add(int x, T v) {
        for (int i = x + 1; i <= n; i += i & -i) {
            a[i - 1] += v;
        }
    }

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

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

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

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;
    }
};
```

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)];
    }
};
```

线段树（SegmentTree 基础区间加乘）

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

线段树（SegmentTree+Info 查找前驱后继）

```cpp
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;
}
```

线段树（SegmentTree+Info+Merge 区间合并）

```cpp
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};
    }
}
```

懒标记线段树（LazySegmentTree 基础区间修改）

```cpp
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;
}
```

懒标记线段树（LazySegmentTree 查找前驱后继）

```cpp
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};
}
```

懒标记线段树（LazySegmentTree 二分修改）

```cpp
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;
}
```

取模类（MLong & MInt）

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

取模类（MLong & MInt 新版）

根据输入内容动态修改 MOD 的方法：`Z::setMod(p)`

```cpp
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();

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

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];
        }
    }
};
```

Splay

```cpp
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;
}
```

```cpp
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;
}
```

```cpp
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);
            }
        }
    }
};

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);
}
```

其他平衡树

```cpp
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);
    }
}
```

```cpp
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;
}
```

```cpp
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);
}
```

```cpp
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;
}
```

分数四则运算（Frac）

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

线性基（Basis）

```cpp
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;
    }
};
```

马拉车（Manacher）

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

Z函数

```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;
}
```

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;
            }
        }
    }
};
```

后缀自动机（SuffixAutomaton 旧版）

```cpp
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;
    }
};
```

后缀自动机（SAM 新版）

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

回文自动机（PAM）

```cpp
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;
```

AC自动机（AC 旧版）

```cpp
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;

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;
    }
};
```

AC自动机（AhoCorasick 新版）

```cpp
struct AhoCorasick {
    static constexpr int ALPHABET = 26;
    struct Node {
        int len;
        int link;
        std::array<int, ALPHABET> next;
        Node() : link{}, 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::vector<int> &a) {
        int p = 1;
        for (auto x : a) {
            if (t[p].next[x] == 0) {
                t[p].next[x] = newNode();
                t[t[p].next[x]].len = t[p].len + 1;
            }
            p = t[p].next[x];
        }
        return p;
    }

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

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

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

```
            q.pop();

            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];
    }

    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();
    }
};
```

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;
    }

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

```
52
53      int px = 0;
54      for (auto c : x) {
55          px = (10LL * px + c - '0') % P;
56      }
57
58      for (int i = 0; i <= n - 2 * (m - 1); i++) {
59          if ((get(i, i + m - 1) + get(i + m - 1, i + 2 * m - 2)) % P == px)
{
60              std::cout << i + 1 << " " << i + m - 1 << "\n";
61              std::cout << i + m << " " << i + 2 * m - 2 << "\n";
62              return 0;
63          }
64      }
65
66      std::vector<int> z(m + 1), f(n + 1);
67      z[0] = m;
68
69      for (int i = 1, j = -1; i < m; i++) {
70          if (j != -1) {
71              z[i] = std::max(0, std::min(j + z[j] - i, z[i - j]));
72          }
73          while (z[i] + i < m && x[z[i]] == x[z[i] + i]) {
74              z[i]++;
75          }
76          if (j == -1 || i + z[i] > j + z[j]) {
77              j = i;
78          }
79      }
80      for (int i = 0, j = -1; i < n; i++) {
81          if (j != -1) {
82              f[i] = std::max(0, std::min(j + f[j] - i, z[i - j]));
83          }
84          while (f[i] + i < n && f[i] < m && x[f[i]] == s[f[i] + i]) {
85              f[i]++;
86          }
87          if (j == -1 || i + f[i] > j + f[j]) {
88              j = i;
89          }
90      }
91
92      for (int i = 0; i + m <= n; i++) {
93          int l = std::min(m, f[i]);
94
95          for (auto j : { m - l, m - l - 1 }) {
96              if (j <= 0) {
97                  continue;
98              }
99              if (j <= i && (get(i - j, i) + get(i, i + m)) % P == px) {
100                 std::cout << i - j + 1 << " " << i << "\n";
101                 std::cout << i + 1 << " " << i + m << "\n";
102                 return 0;
103             }
104             if (i + m + j <= n && (get(i, i + m) + get(i + m, i + m + j)) %
P == px) {
105                 std::cout << i + 1 << " " << i + m << "\n";
```

```
                std::cout << i + m + 1 << " " << i + m + j << "\n";
                return 0;
            }
        }
    }

    return 0;
}
```

BigInt

```
#ifdef UseFFT
/***** FFT 板子 *******/
namespace FFT{
// 精度保证：系数总和小于 1e15 开longduoble
// 精度够(maybe总和小于1e10)可以不开longdouble
// 先调用fft_init()
// FFT_MAXN = 2^k
// fft_init() to precalc FFT_MAXN-th roots
typedef long double db;
const int FFT_MX=2097152, N =4001000;
const db pi=acosl(-1.);
struct cp{
    db a,b;
    cp operator+(const cp&y)const{return (cp){a+y.a,b+y.b};}
    cp operator-(const cp&y)const{return (cp){a-y.a,b-y.b};}
    cp operator*(const cp&y)const{return (cp){a*y.a-b*y.b,a*y.b+b*y.a};}
    cp operator!()const{return (cp){a,-b};};
}nw[FFT_MX+1];
int bitrev[FFT_MX];

void dft(cp*a,int n,int flag=1){
    int d=0;
    while((1<<d)*n!=FFT_MX) d++;
    For(i,0,n-1) if(i < (bitrev[i]>>d))
        swap(a[i], a[bitrev[i]>>d]);
    for (int l=2;l<=n;l<<=1){
        int del=FFT_MX/l*flag;
        for (int i=0;i<n;i+=l){
            cp *le=a+i, *ri=a+i+(l>>1), *w=(flag==1) ? nw : nw+FFT_MX;
            For(k,0,(l>>1)-1){
                cp ne=*ri**w;
                *ri=*le-ne, *le=*le+ne;
                le++, ri++, w+=del;
            }
        }
    }
    if(flag!=1) For(i,0,n-1)
        a[i].a/=n, a[i].b/=n;
}
void fft_init(){
    int L=0;
    while((1<<L)!=FFT_MX) L++;
    bitrev[0]=0;
    For(i,1,FFT_MX-1)
        bitrev[i] = bitrev[i>>1]>>1 | ((i&1)<<(L-1));
    nw[0]=nw[FFT_MX]=(cp){1,0};
    For(i,0,FFT_MX)nw[i] = (cp){cosl(2*pi/FFT_MX*i), sinl(2*pi/FFT_MX*i)};
//very slow
}
// n, m 分别为a, b的最高次幂, 数组a的范围为[0, n], b为[0, m], c转整数四舍五入
void polymul(db *a, int n, db *b, int m, db *c) {
    static cp f[FFT_MX>>1], g[FFT_MX>>1], t[FFT_MX>>1];
```

```
    int N=2;
    while(N<=n+m)N<<=1;
    For(i,0,N-1)    // 此N非全局的N
        if(i&1){
            f[i>>1].b=(i<=n)?a[i]:0.0;
            g[i>>1].b=(i<=m)?b[i]:0.0;
        }else{
            f[i>>1].a=(i<=n)?a[i]:0.0;
            g[i>>1].a=(i<=m)?b[i]:0.0;
        }
    dft(f,N>>1); dft(g,N>>1);
    int del=FFT_MX/(N>>1);
    cp qua=(cp){0,0.25}, one=(cp){1,0}, four=(cp){4,0}, *w=nw;
    For(i,0,(N>>1)-1){
        int j=i?(N>>1)-i:0;
        t[i] = (four*!(f[j]*g[j])-(!f[j]-f[i])*(!g[j]-g[i])*(one+*w))*qua;
        w+=del;
    }
    dft(t,N>>1,-1ll);
    For(i,0,n+m)c[i]=(i&1) ? t[i>>1].a : t[i>>1].b;
}
// 这实际抄的时候可以和上面结合一下 少个复制的常数
db A[N], B[N], C[N];
vector<ll> vectorMul(const vector<ll>& a, const vector<ll>& b){
    int n = a.size(), m = b.size();
    if(n==0 || m==0)return {};
    n--, m--;
    For(i,0,n)A[i] = a[i];
    For(i,0,m)B[i] = b[i];
    polymul(A, n, B, m, C);
    vector<ll> res(n+m+1);
    For(i,0,n+m) res[i] = C[i]+0.5;
    return res;
}
}
#endif
```

```cpp
namespace BigIntSP{
// 十进制高精度板子, 将 D 个十进制位压到一起
// FFT 时最好<=5
const int D = 1;
const int B = pow(10, D);
struct BigInt
{
    // ========================== 初始化部分 ==========================
    // int, string 转 BigInt; BigInt 转 string 输出; 取绝对值, 取反; 高位推进

    int sign = 0;
    vector<ll> v;
    BigInt(ll x = 0){
        set(x);
    }
    BigInt(const string &s) {
        set(s);
    }
    void set(ll x){
        v.clear();
        sign = 0;
        if (x < 0) x *= -1, sign = 1;
        while (x) {
            v.push_back(x % B);
            x /= B;
        }
    }
    void set(const string& s){
        sign = 0;
        v.clear();
        int beg = 0;
        if (s[0] == '-')
            beg++, sign = 1;
        int add = 0, cnt = 0, base = 1;
        for (int i = s.size()-1; i >= beg; i--) {
            if (cnt == D) {
                v.push_back(add);
                cnt = add = 0;
                base = 1;
            }
            add = (s[i] - '0') * base + add;
            cnt++;
            base *= 10;
        }
        if (add) v.push_back(add);
    }
    BigInt operator-() const {
        BigInt res = *this;
        res.sign ^= 1;
        return res;
    }
    BigInt abs() const {
        BigInt res = *this;
        res.sign = 0;
```

```
        return res;
    }
    ll& operator[](const int i) { return v[i]; }
    int size() const { return v.size(); }
    void norm() {       // 向高位推一遍进位
        For(i,0,(ll)v.size()-2) {
            if (v[i] >= 0) {
                v[i + 1] += v[i] / B;
                v[i] %= B;
            } else {
                int c = (-v[i] + B - 1) / B;
                v[i] += c * B;
                v[i + 1] -= c;
            }
        }
        while (!v.empty() && v.back() >= B) {
            int c = v.back() / B;
            v.back() %= B;
            v.push_back(c);
        }
        while (!v.empty() && v.back() == 0) v.pop_back();
    }
    string to_str() const {
        string res;
        if (v.empty()) return "0";
        if (sign) res += '-';
        res += to_string(v.back());
        for (int i = (ll)v.size() - 2; i >= 0; i--) {
            string add;
            int w = v[i];
            For(_,1,D){
                add += ('0' + (w % 10));
                w /= 10;
            }
            reverse(all(add));
            res += add;
        }
        return res;
    }
    friend istream& operator>>(istream &is, BigInt &x) {
        string tmp;
        is >> tmp;
        x = BigInt(tmp);
        return is;
    }
    friend ostream& operator<<(ostream &os, BigInt x) {
        os << x.to_str();
        return os;
    }


    // ========================= O(n) 运算部分 =========================
    // 高精度加法，高精度减法，高精度乘低精度，高精度除低精度

    BigInt& operator+=(const BigInt &x) {
        if (sign != x.sign) {
```

```
            *this -= (-x);
            return *this;
        }
        if ((int)v.size() < (int)x.size())
            v.resize(x.size(), 0);
        For(i,0,(ll)x.size()-1)
            v[i] += x.v[i];
        norm();
        return *this;
    }
    BigInt& operator-=(const BigInt &x) {
        if (sign != x.sign) {
            *this += (-x);
            return *this;
        }
        if (abs() < x.abs()) {
            *this = x - (*this);
            sign ^= 1;
            return *this;
        }
        For(i,0,(ll)x.size()-1)
            v[i] -= x.v[i];
        norm();
        return *this;
    }
    BigInt operator*(ll x) const {  // 注意爆ll (D<=9)
        BigInt res(*this);
        if (x < 0) res.sign ^= 1, x *= -1;
        for (int i = (ll)res.v.size()-1; i >= 0; i--)
            res.v[i] *= x;
        res.norm();
        return res;
    }
    BigInt& operator/=(ll x) {
        if (x < 0) sign ^= 1, x *= -1;
        for (int i = (ll)v.size()-1; i >= 0; i--){
            if (v[i] % x != 0 && i != 0) {
                v[i - 1] += B * (v[i] % x);
            }
            v[i] /= x;
        }
        norm();
        return *this;
    }

    BigInt operator+(const BigInt &x) const { return BigInt(*this) += x; }
    BigInt operator-(const BigInt &x) const { return BigInt(*this) -= x; }
    BigInt operator*=(const int &x) { return (*this) = (*this)*x; }
    BigInt operator/(const int &x) const { return BigInt(*this) /= x; }


    // ========================= 比较函数部分 =========================

    bool gtZer(){return sign==0 && v.size();}   // return BitInt(x) > 0

    bool operator<(const BigInt &x) const {
```

```
        if (sign != x.sign) return sign > x.sign;
        if (v.size() != x.size()) {
            if (sign) return (int)x.size() < (int)v.size();
            else return (int)v.size() < (int)x.size();
        }
        Rep(i, (ll)v.size()-1, 0) if (v[i] != x.v[i]) {
            if (sign) return x.v[i] < v[i];
            else return v[i] < x.v[i];
        }
        return false;
    }
    bool operator>(const BigInt &x) const { return x < *this; }
    bool operator<=(const BigInt &x) const { return !(x < *this); }
    bool operator>=(const BigInt &x) const { return !(*this < x); }
    bool operator==(const BigInt &x) const { return !(*this < x) && !(x < *this); }
    bool operator!=(const BigInt &x) const { return !(*this == x); }


    // ========================== O(n^2) 运算部分 ========================
    // 高精度乘高精度, 高精度除高精度, 高精度模高精度, 高精度取模, 高精度平方

    BigInt operator * (const BigInt &x) const {
        #ifndef UseFFT
        return n2Mul(*this, x);
        #else
        return FFTMul(*this, x);
        #endif
        // return karatsubaMul(x);
    }
    BigInt& operator/=(BigInt x){
        int lstSign = sign ^ x.sign;
        sign = x.sign = 0;
        if ((*this) < x)
            return *this = BigInt();
        if (x == BigInt(1)){
            sign = lstSign;
            return *this;
        }
        int d = v.size() - x.size() + 1;
        BigInt inv(1LL * B * B / x.v.back()), pre(0), c;
        int cur = 2, bcur = 1;
        while (inv != pre || bcur < x.size()) {
            bcur = min(bcur << 1, x.size());
            c.v = vector<ll>(x.v.end()-bcur, x.v.end());
            pre = inv;
            inv *= ((BigInt(2) << (cur + bcur - 1)) - inv * c);
            cur = min(cur << 1, d);
            inv.v = vector<ll>(inv.v.end()-cur, inv.v.end());
        }
        inv.v = vector<ll>(inv.v.end() - d, inv.v.end());
        BigInt res = (*this) * inv;
        res >>= (v.size());
        BigInt tt = (*this) - res * x;
        while (x <= tt) {
            res += BigInt(1);
```

```
            tt -= x;
        }
        v = res.v;
        return *this;
    }
    BigInt& operator%=(const BigInt &x) {
        BigInt div = (*this) / x;
        (*this) -= div * x;
        return *this;
    }

    void divMod(const BigInt& x, BigInt& divRes, BigInt& modRes) const {
        divRes = (*this) / x;
        modRes = (*this) - divRes*x;
    }

    BigInt square() {
        BigInt res = *this;
        res.sign = 0;
        #ifdef UseFFT
        auto v1 = FFT::vectorMul(v, v);
        #else
        auto v1 = n2Mul(*this, *this);
        #endif
        res.v.assign(v1.size(), 0);
        For(i, 0, (ll)v1.size()-1) {
            ll val = v1[i];
            for (int j = i; val; j++) {
                if (j == (int)res.v.size())
                    res.v.push_back(0);
                res.v[j] += val % B;
                val /= B;
            }
        }
        res.norm();
        return res;
    }

    BigInt operator*=(const BigInt &x) { return (*this) = (*this)*x; }
    BigInt operator/(const BigInt &x) const { return BigInt(*this) /= x; }
    BigInt operator%(const BigInt &x) const { return BigInt(*this) %= x; }

private:

    /*****************三种乘法**********************/
    static BigInt n2Mul(const BigInt& a, const BigInt& b) {
        BigInt res;
        res.v.resize(a.size()+b.size(), 0);
        res.sign = a.sign ^ b.sign;
        for(int i = 0; i < a.size(); i++)
            for(int j = 0; j < b.size(); j++)
                res[i+j] += a.v[i]*b.v[j];
        res.norm();
        return res;
    }
    // BigInt& karatsubaMul(const BigInt& x){
```

```
    //     // todo
    // }
    #ifdef UseFFT
    static BigInt FFTMul(const BigInt& a, const BigInt& b) {
        if(a.size()+b.size() < 1000)return n2Mul(a, b);

        BigInt res;
        res.sign = a.sign ^ b.sign;
        auto v1 = FFT::vectorMul(a.v, b.v);
        res.v.assign(v1.size(), 0);
        for(int i = 0; i < v1.size(); i++){
            ll val = v1[i];
            for (int j = i; val; j++) {
                if (j == (int)res.v.size())
                    res.v.push_back(0);
                res.v[j] += val % B;
                val /= B;
            }
        }
        res.norm();
        return res;
    }
    #endif

    /*** 这的左右移不是正常的左右移 做除法时用 ***/
    BigInt& operator<<=(const int& x){
        if(!v.empty()) {
            vector<ll> add(x, 0);
            v.insert(v.begin(), all(add));
        }
        return *this;
    }
    BigInt& operator>>=(const int& x){
        v = vector<ll>(v.begin()+min(x,(int)v.size()), v.end());
        return *this;
    }
    BigInt operator<<(const int& x)const{return BigInt(*this)<<=x;}
    BigInt operator>>(const int& x)const{return BigInt(*this)>>=x;}
};
typedef BigInt Bigint;
Bigint qmi(Bigint m, int k)
{
    Bigint res(1);
    while (k) {
        if (k & 1) res *= m;
        m *= m;
        k >>= 1;
    }
    return res;
}
}
using BigIntSP::BigInt;
using BigIntSP::qmi;
```

点分治

```
int n, k;
vector<pii> G[N];
// rt是重心
int vis[N];
int rt, sz[N], wt[N];   // wt[u]是u作为rt时的重儿子的sz
void findRt(int u, int fa, int subSZ){  // subSZ是当前子树大小
    sz[u] = 1, wt[u] = 0;
    for(auto [v, w]:G[u])if(v != fa && !vis[v]){
        findRt(v, u, subSZ);
        sz[u] += sz[v];
        wt[u] = max(wt[u], sz[v]);
    }
    wt[u] = max(wt[u], subSZ - sz[u]);
    if(wt[rt] > wt[u])rt = u;   // 找个重儿子最小的作为根
}
int arr[N], cnt;
void dfs(int u, int fa, int dep){
    arr[++cnt] = dep;
    for(auto [v, w]:G[u])if(!vis[v] && v != fa)
        dfs(v, u, dep+w);
}
// 算答案部分，点分治之后，算u子树内经过u的路径的答案
int calc(int u, int val){   // 计算子树内 路径+val <= k的对数, 路径可能是非法的
    cnt = 0;
    dfs(u, 0, val);
    sort(arr+1, arr+1+cnt);
    int res = 0;
    for(int l = 1, r = cnt; l <= r; l++){
        while(l<=r && arr[l]+arr[r]>k)r--;
        if(l>r)break;
        // l -> u -> (l,r]  有r-l条
        res += r-l;
    }
    return res;
}
int ans;
void DFS(int u){
    ans += calc(u, 0);
    vis[u] = 1;     // vis设为1，做子树的时候就不会dfs出来了
    for(auto [v, w]:G[u])if(!vis[v]){
        ans -= calc(v, w);  // 容斥 减掉前面calc(u, 0)多算的部分
        rt = 0;
        findRt(v, 0, sz[v]);   // 重新在v子树找rt
        DFS(rt);
    }
}
void solve()
{
    cin >> n;
    ans = rt = 0;
    wt[rt] = LINF;
    For(i,1,n-1){
```

```
        int u, v, w;
        cin >> u >> v >> w;
        G[u].push_back({v, w});
        G[v].push_back({u, w});
    }
    cin >> k;
    findRt(1, 0, n);
    DFS(rt);
    cout << ans << "\n";
}
```

树分治

树分治版题 (P6329)

给一棵树，每个点有点权, 多次操作(强制在线)

1. 修改一个点的点权
2. 查询距离x小于等于k的所有点权和

复杂度$O(n\log^2 n)$

```
// G1原树 G重构树
vector<int> G1[N], G[N];
BIT<ll> T[2][N];    // BIT用vector动态开时 查询修改记得取min
int fa[N][21], dep[N];
int cfa[N];      // 重构树上的fa
int n, a[N], q;
int rt, sz[N], vis[N], wt[N];   // wt[u]是u作为rt时的重儿子的sz
namespace LCASP{
void dfs0(int u, int f){
    fa[u][0] = f;
    For(i,1,20)fa[u][i] = fa[fa[u][i-1]][i-1];
    for(auto v:G1[u])if(v != f){
        dep[v] = dep[u]+1;
        dfs0(v, u);
    }
}
int lca(int u, int v){
    if(dep[u] < dep[v])swap(u, v);
    int cha = dep[u] - dep[v];
    Rep(i,20,0)if(cha>>i&1)
        u = fa[u][i];
    if(u==v)return u;
    Rep(i,20,0) if(fa[u][i] != fa[v][i]){
        u = fa[u][i], v = fa[v][i];
    }
    return fa[u][0];
}
int dis(int u, int v){
    int lc = lca(u, v);
    return dep[u]+dep[v]-2*dep[lc];
}
}
using namespace LCASP;
void findRt(int u, int fa, int subSZ){  // subSZ是当前子树大小
    sz[u] = 1, wt[u] = 0;
    for(auto v:G1[u])if(v != fa && !vis[v]){
        findRt(v, u, subSZ);
        sz[u] += sz[v];
        wt[u] = max(wt[u], sz[v]);
    }
    wt[u] = max(wt[u], subSZ - sz[u]);
    if(wt[rt] > wt[u])rt = u;   // 找个重儿子最小的作为根
}
```

```cpp
int tsz[N];
void getsz(int u, int fa){
    tsz[u] = 1;
    for(auto v:G1[u])if(v != fa && !vis[v]){
        getsz(v, u);
        tsz[u] += tsz[v];
    }
}
void dfs(int u) {
    getsz(u, 0);    // 用findRt的sz是不准的(只是在点分治时负责度对)
    T[0][u].tre.resize(tsz[u]+2, 0);
    T[1][u].tre.resize(tsz[u]+2, 0);
    vis[u] = 1;
    for(auto v:G1[u])if(!vis[v]) {
        rt = 0;
        findRt(v, u, tsz[v]);
        cfa[rt] = u;
        G[u].push_back(rt); // 建重构树，这题没用到
        dfs(rt);
    }
}
void solve()
{
    cin >> n >> q;
    For(i,1,n)cin >> a[i];
    For(i,2,n) {
        int u, v;
        cin >> u >> v;
        G1[u].push_back(v);
        G1[v].push_back(u);
    }
    dfs0(1, 0);

    rt = 0;
    wt[0] = LINF;
    findRt(1, 0, n);
    dfs(rt);

    auto add = [&](int u, int val){
        for(int c = u; c; c=cfa[c]){
            T[0][c].add(dis(u, c)+1, val);
        }
        for(int c = u; cfa[c]; c=cfa[c])T[1][c].add(dis(u, cfa[c])+1, val);
    };
    For(i,1,n) add(i, a[i]);

    int ans = 0;
    while(q--){
        int op, x, y;
        cin >> op >> x >> y;
        x ^= ans, y ^= ans;
        if(op == 0){
            ans = T[0][x].query(y+1);
            for(int u = x; cfa[u]; u = cfa[u]){
                int xdis = dis(cfa[u], x);
                if(y-xdis < 0)continue; // 这不能break 可能上面的点离x更近
```

```
                ans += T[0][cfa[u]].query(y-xdis+1);
                ans -= T[1][u].query(y-xdis+1);
            }
            cout << ans << "\n";
        } else {
            add(x, y-a[x]);
            a[x] = y;
        }
    }
}
```

网络流封装

Dinic 最大流

```cpp
template<typename T> struct Flow_ {
    const int n;
    const T inf = numeric_limits<T>::max();
    struct Edge {
        int to;
        T w;
        Edge(int to, T w) : to(to), w(w) {}
    };
    vector<Edge> ver;
    vector<vector<int>> h;
    vector<int> cur, d;

    Flow_(int n) : n(n), h(n + 1) {}
    void add(int u, int v, T c)
    {
        h[u].push_back(ver.size());
        ver.emplace_back(v, c);
        h[v].push_back(ver.size());
        ver.emplace_back(u, 0);
    }
    bool bfs(int s, int t)
    {
        d.assign(n + 1, -1);
        d[s] = 0;
        queue<int> q;
        q.push(s);
        while(!q.empty())
        {
            auto x = q.front();
            q.pop();
            for(auto it: h[x])
            {
                auto [y, w] = ver[it];
                if(w && d[y] == -1)
                {
                    d[y] = d[x] + 1;
                    if(y == t) return true;
                    q.push(y);
                }
            }
        }
        return false;
    }
    T dfs(int u, int t, T f)
    {
        if(u == t) return f;
        auto r = f;
        for(int &i = cur[u]; i < h[u].size(); i++)
        {
            auto j = h[u][i];
```

```
            auto &[v, c] = ver[j];
            auto &[u, rc] = ver[j ^ 1];
            if(c && d[v] == d[u] + 1)
            {
                auto a = dfs(v, t, std::min(r, c));
                c -= a;
                rc += a;
                r -= a;
                if(!r) return f;
            }
        }
        return f - r;
    }
    T work(int s, int t)
    {
        T ans = 0;
        while(bfs(s, t))
        {
            cur.assign(n + 1, 0);
            ans += dfs(s, t, inf);
        }
        return ans;
    }
};
using Flow = Flow_<int>;
```

ISAP 最大流

```cpp
template<typename T> struct Flow_ {
    const int n;
    const T inf = numeric_limits<T>::max();
    struct Edge {
        int to;
        T cap, flow;
        Edge(int _to = 0, T _cap = 0) :
            to(_to), cap(_cap), flow(0) {}
    };
    vector<Edge> ver;
    vector<vector<int>> h;
    vector<int> dep, gap, cur;
    vector<int> stk, que;

    Flow_(int n) : n(n), h(n + 1) {}
    void addedge(int u, int v, int w)
    {
        h[u].push_back(ver.size());
        ver.emplace_back(v, w);
        h[v].push_back(ver.size());
        ver.emplace_back(u, 0);
    }
    void bfs(int s, int t)
    {
        dep.assign(n + 1, -1);
        gap.assign(n + 2, 0);
        que.assign(1, 0);
        gap[0] = 1;
        int hh = 0, tt = 0;
        dep[t] = 0, que[0] = t;
        while(hh <= tt)
        {
            int u = que[hh ++];
            for(int i = 0; i < h[u].size(); i++)
            {
                int j = h[u][i];
                auto &[v, c, f] = ver[j];
                if(dep[v] != -1) continue;
                if((int)que.size() == tt + 1) que.push_back(0);
                que[++ tt] = v;
                dep[v] = dep[u] + 1;
                gap[dep[v]] ++;
            }
        }
    }
    T work(int s, int t)
    {
        bfs(s, t);
        cur.assign(n + 1, 0);
        int u = s, top = 0;
        T ans = 0;
        while(dep[s] < n)
```

```
        {
            if(u == t)
            {
                T Min = inf;
                int inser;
                for(int i = 0; i < top; i++)
                    if(Min > ver[stk[i]].cap - ver[stk[i]].flow)
                    {
                        Min = ver[stk[i]].cap - ver[stk[i]].flow;
                        inser = i;
                    }
                for(int i = 0; i < top; i++)
                {
                    ver[stk[i]].flow += Min;
                    ver[stk[i] ^ 1].flow -= Min;
                }
                ans += Min;
                top = inser;
                u = ver[stk[top] ^ 1].to;
                continue;
            }
            bool flag = false;
            int _v;
            for(int &i = cur[u]; i < h[u].size(); i++)
            {
                auto j = h[u][i];
                auto &[v, c, f] = ver[j];
                if(c - f && dep[v] + 1 == dep[u])
                {
                    flag = true;
                    _v = v;
                    break;
                }
            }
            if(flag)
            {
                if(stk.size() == top) stk.push_back(0);
                stk[top ++] = h[u][cur[u]];
                u = _v;
                continue;
            }
            int Min = n;
            for(int i = 0; i < h[u].size(); i++)
            {
                auto j = h[u][i];
                auto &[v, c, f] = ver[j];
                if(c - f && dep[v] < Min)
                {
                    Min = dep[v];
                    cur[u] = i;
                }
            }
            gap[dep[u]] --;
            if(!gap[dep[u]]) return ans;
            dep[u] = Min + 1;
            gap[dep[u]] ++;
```

```
            if(u != s) u = ver[stk[-- top] ^ 1].to;
        }
        return ans;
    }
};
using Flow = Flow_<int>;
```

HLPP 预流推进

```cpp
template<typename T> struct PushRelabel {
    const int inf = 0x3f3f3f3f;
    const T INF = 0x3f3f3f3f3f3f3f3f;
    struct Edge {
        int to, cap, flow, anti;
        Edge(int v = 0, int w = 0, int id = 0) :
            to(v), cap(w), flow(0), anti(id) {}
    };
    vector<vector<Edge>> e;
    vector<vector<int>> gap;
    vector<T> ex;
    vector<bool> ingap;
    vector<int> h;
    int n, gobalcnt, maxH = 0;
    T maxflow = 0;

    PushRelabel(int n) : n(n), e(n + 1), ex(n + 1), gap(n + 1) {}
    void addedge(int u, int v, int w)
    {
        e[u].push_back({v, w, (int)e[v].size()});
        e[v].push_back({u, 0, (int)e[u].size() - 1});
    }
    void PushEdge(int u, Edge &edge)
    {
        int v = edge.to, d = min(ex[u], 1ll * edge.cap - edge.flow);
        ex[u] -= d;
        ex[v] += d;
        edge.flow += d;
        e[v][edge.anti].flow -= d;
        if(h[v] != inf && d > 0 && ex[v] == d && !ingap[v])
        {
            ++ gobalcnt;
            gap[h[v]].push_back(v);
            ingap[v] = 1;
        }
    }
    void PushPoint(int u)
    {
        for(auto k = e[u].begin(); k != e[u].end(); k++)
        {
            if(h[k->to] + 1 == h[u] && k->cap > k->flow)
            {
                PushEdge(u, *k);
                if(!ex[u]) break;
            }
        }
        if(!ex[u]) return;
        if(gap[h[u]].empty())
        {
            for(int i = h[u] + 1; i <= min(maxH, n); i++)
            {
                for(auto v: gap[i])
```

```
                    ingap[v] = 0;
                gap[i].clear();
            }
        }
        h[u] = inf;
        for(auto [to, cap, flow, anti]: e[u])
        {
            if(cap > flow)
                h[u] = min(h[u], h[to] + 1);
        }
        if(h[u] >= n) return;
        maxH = max(maxH, h[u]);
        if(!ingap[u])
        {
            gap[h[u]].push_back(u);
            ingap[u] = 1;
        }
    }
    void init(int t, bool f = 1)
    {
        ingap.assign(n + 1, 0);
        for(int i = 1; i <= maxH; i++)
            gap[i].clear();
        gobalcnt = 0, maxH = 0;
        queue<int> q;
        h.assign(n + 1, inf);
        h[t] = 0, q.push(t);
        while(q.size())
        {
            int u = q.front();
            q.pop(), maxH = h[u];
            for(auto &[v, cap, flow, anti]: e[u])
            {
                if(h[v] == inf && e[v][anti].cap > e[v][anti].flow)
                {
                    h[v] = h[u] + 1;
                    q.push(v);
                    if(f)
                    {
                        gap[h[v]].push_back(v);
                        ingap[v] = 1;
                    }
                }
            }
        }
    }
    T work(int s, int t)
    {
        init(t, 0);
        if(h[s] == inf) return maxflow;
        h[s] = n;
        ex[s] = INF;
        ex[t] = -INF;
        for(auto k = e[s].begin(); k != e[s].end(); k++)
            PushEdge(s, *k);
        while(maxH > 0)
```

```
        {
            if(gap[maxH].empty())
            {
                maxH --;
                continue;
            }
            int u = gap[maxH].back();
            gap[maxH].pop_back();
            ingap[u] = 0;
            PushPoint(u);
            if(gobalcnt >= 10 * n)
                init(t);
        }
        ex[s] -= INF;
        ex[t] += INF;
        return maxflow = ex[t];
    }
};
```

EK 费用流

```cpp
struct MinCostFlow {
    using LL = long long;
    using PII = pair<LL,int>;
    const LL INF = numeric_limits<LL>::max();
    struct Edge {
        int v, c, f;
        Edge(int v, int c, int f) : v(v), c(c), f(f) {}
    };
    const int n;
    vector<Edge> e;
    vector<vector<int>> g;
    vector<LL> h, dis;
    vector<int> pre;

    MinCostFlow(int n) : n(n), g(n + 1) {}
    void add(int u, int v, int c, int f) // c 流量, f 费用
    {
        g[u].push_back(e.size());
        e.emplace_back(v, c, f);
        g[v].push_back(e.size());
        e.emplace_back(u, 0, -f);
    }
    bool dijkstra(int s, int t)
    {
        dis.assign(n + 1, INF);
        pre.assign(n + 1, -1);
        priority_queue<PII, vector<PII>, greater<PII>> que;
        dis[s] = 0;
        que.emplace(0, s);
        while(!que.empty())
        {
            auto [d, u] = que.top();
            que.pop();
            if(dis[u] < d) continue;
            for(int i: g[u])
            {
                auto [v, c, f] = e[i];
                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] != INF;
    }
    pair<int,LL> flow(int s, int t)
    {
        int flow = 0;
        LL cost = 0;
        h.assign(n + 1, 0);
```

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

Bitset Bfs

bitset 相关函数

```
bitset<N> foo;
```

- `foo.size()` 返回大小 (位数)
- `foo.count()` 返回 1 的个数
- `foo.any()` 返回是否有 1
- `foo.none()` 返回是否没有 1
- `foo.set()` 全部置为 1
- `foo.set(p)` 将第 $p+1$ 位置为 1
- `foo.set(p, x)` 将第 $p+1$ 位置为 x
- `foo.reset()` 全部置为 0
- `foo.reset(p)` 将第 $p+1$ 位置为 0
- `foo.flip` 全部取反
- `foo.flip(p)` 将第 $p+1$ 位取反
- `foo.to_ulong()` 返回它转换为 `unsigned long` 的结果, 如果超出范围则报错
- `foo.to_ullong()` 返回它转换为 `unsigned long long` 的结果, 如果超出范围则报错
- `foo.to_string()` 返回它转换为 `string` 的结果

AcWing 164. 可达性统计

题意 : 给定一张 n 个点 m 条边的有向无环图 $(1 \leq n, m \leq 30000)$, 分别统计从每个点出发能到达的点的数量

解法1 : 拓扑排序 + bitset

`bitset<N> f[N]` 维护每个点能到达的点集, 拓扑排序后倒序枚举每个点 `f[u] |= f[v]`

```
int n, m, d[N];
bitset<N> f[N];
vector<int> edges[N];
vector<int> top_que; // 拓扑序

void topsort()
{
    queue<int> q;
    for(int i = 1; i <= n; i++)
        if(!d[i]) q.push(i);

    while(q.size())
    {
        int u = q.front();
        q.pop();
```

```
        top_que.push_back(u);
        for(auto v: edges[u])
            if(!(--d[v])) q.push(v);
    }
}

int main()
{
    cin >> n >> m;
    for(int i = 0; i < m; i++)
    {
        int u, v;
        cin >> u >> v;
        edges[u].push_back(v);
        d[v] ++;
    }

    topsort();
    for(int i = n - 1; i >= 0; i--)
    {
        int u = top_que[i];
        f[u][u] = 1;
        for(auto v: edges[u])
            f[u] |= f[v];
    }
    for(int i = 1; i <= n; i++)
        cout << f[i].count() << '\n';
}
```

解法2：记忆化搜索

```
int n, m;
bitset<N> f[N];
vector<int> edges[N];
bool vis[N];

bitset<N> dp(int u)
{
    if(vis[u]) return f[u];
    vis[u] = true;

    f[u][u] = 1;
    for(auto v: edges[u])
        f[u] |= dp(v);
    return f[u];
}

int main()
{
    cin >> n >> m;
    for(int i = 0; i < m; i++)
    {
        int u, v;
        cin >> u >> v;
        edges[u].push_back(v);
```

```
    }

    for(int i = 1; i <= n; i++)
        cout << dp(i).count() << '\n';
}
```

牛客练习赛14. 无向图中的最短距离

题意：有一个 n 个点，m 条边的无向图，每条边的边权为 1，q 个询问，每次询问给出若干个 (x_i, y_i)，求至少与一个点对存在 $dist(x_i, v) \leq y_i$ 的点 v 的个数 $(n \leq 1000, m \leq 100000, q \leq 100000)$

题解：`bitset` + `bfs` + `可达性`

`bitset<N> dp[i][j]` 维护 i 为起点，距离 $\leq j$ 能到达的点集

`cur` 表示前 $j-1$ 步的可达点集，`nxt` 表示前 j 步的可达点集，`tmp` 表示仅第 j 步可达的点集

`tmp` 可以由 `tem = cur ^ nxt` 得到，当 `cur == nxt` 时，表示第 j 步无法再扩展，此时 `break` 即可

lowBit 实现返回一段 bitset 内的 lowbit

```
const int maxn = 1010, maxd = 1 << 16 | 1;
int lbt[maxd];
void init(){
    //lbt[i]表示i的二进制表示中lowbit在右起第几位(0-index) 用于lowbit函数的初始化
    lbt[0] = -1;
    for(int i = 1; i < maxd; ++i)
        lbt[i] = i & 1 ? 0 : lbt[i >> 1] + 1;
}
//lowbit(bitset,low,upp) 求该bitset的[low,upp]内的lowbit
int lowBit(bitset<maxn> const &msk, size_t const &low, size_t const &upp) {
    typedef unsigned long long _WordT;
    _WordT *seq = (_WordT *)&msk;
    size_t pL = low >> 6, pR = upp >> 6;
    size_t qL = low & 63, qR = upp & 63;
    for(size_t i = pL; i <= pR; ++i) {
        _WordT val = seq[i];
        if(i == pR && qR < 63)
            val &= (static_cast<_WordT>(1) << (qR + 1)) - 1;
        if(i == pL)
            val = (val >> qL) << qL;
        if(val != static_cast<_WordT>(0)) {
            size_t ret = i << 6;
            if((val & ((static_cast<_WordT>(1) << 32) - 1)) ==
static_cast<_WordT>(0)) {
                val >>= 32;
                ret |= 32;
            }
            if((val & ((static_cast<_WordT>(1) << 16) - 1)) ==
static_cast<_WordT>(0)) {
                val >>= 16;
                ret |= 16;
            }
```

```
            return ret + lbt[static_cast<int>(val & ((static_cast<_WordT>(1) << 16) - 1))];
        }
    }
    return -1;
}

int n, m, q;
bitset<maxn> e[maxn], dp[maxn][maxn], cur, nxt, tmp;

void solve()
{
    cin >> n >> m >> q;
    while(m --)
    {
        int u, v;
        cin >> u >> v;
        u --, v --;
        e[u].set(v);
        e[v].set(u);
    }

    init();
    for(int i = 0; i < n; i++)
    {
        dp[i][0].set(i);
        cur = dp[i][0];
        nxt = cur | e[i];
        for(int j = 1; j <= n; j++)
        {
            tmp = nxt ^ cur;    // 仅第 j 步可达
            cur = dp[i][j] = nxt;   // 前 j 步可达

            for(int u = lowBit(tmp, 0, n - 1); u != -1; u = lowBit(tmp, u + 1, n - 1))
                nxt |= e[u];    // 由仅第 j 步可达的点进行扩展
            if(cur == nxt)
            {
                for(int k = j + 1; k <= n; k++)
                    dp[i][k] = cur;
                break;
            }
        }
    }

    while(q --)
    {
        int c, u, v;
        cin >> c;
        bitset<maxn> res;
        while(c --)
        {
            cin >> u >> v;
            u --;
            res |= dp[u][v];
        }
```

```
85          cout << res.count() << '\n';
86      }
87  }
```

封装 multiset 对顶堆

```cpp
struct Set{
    const int inf = 0x3f3f3f3f;
    multiset<int> small, big;
    Set() {
        small.clear(), big.clear();
        small.insert(-inf), big.insert(inf);
    }
    void adjust() {
        while(small.size() > big.size() + 1)
        {
            multiset<int>::iterator it = (--small.end());
            big.insert(*it);
            small.erase(it);
        }
        while(big.size() > small.size())
        {
            multiset<int>::iterator it = big.begin();
            small.insert(*it);
            big.erase(it);
        }
    }
    void add(int val) {
        if(val <= *big.begin()) small.insert(val);
        else big.insert(val);
        adjust();
    }
    void del(int val) {
        multiset<int>::iterator it = small.lower_bound(val);
        if(it != small.end()) small.erase(it);
        else
        {
            it = big.lower_bound(val);
            big.erase(it);
        }
        adjust();
    }
    int get_middle() {
        return *small.rbegin();
    }
};
```

Miller-Rabin 大素数判定

Miller-Rabin 素性测试

- 作用: 快速判定一个数是否是质数
- 核心idea :

定理 : 设 p 是奇质数, 存在正整数 s 与正奇数 d 使得 $p = 2^s d + 1$. 对于任意 $1 \leq a < p$, 满足要么 $a^d \equiv 1 \ (mod \ n)$, 要么存在一个 $0 \leq r < s$ 使得 $a^{2^r d} \equiv -1 \ (mod \ n)$

- 对于检验一个 $n \geq 3$ 的奇数是否为质数, 考虑一个随机算法: 随机一个 $1 \leq a < n$, 假如 a 满足上述定理的结论, 则称 a 是 n 的一个证人. 可以证明对于合数 n , 最多有 $n/4$ 个 "说谎的" 证人, 因此假设检验 t 次都符合结论, 则 n 是合数的概率最多为 $(1/4)^t$
- 时间复杂度: $O(t \log n)$

```cpp
#include <bits/stdc++.h>
using namespace std;
using i128 = __int128;

inline i128 read()
{
    i128 x = 0, f = 1;
    char ch = getchar();
    while(ch < '0' || ch > '9')
    {
        if(ch == '-') f = -1;
        ch = getchar();
    }
    while(ch >= '0' && ch <= '9')
    {
        x = (x << 1) + (x << 3) + (ch - '0');
        ch = getchar();
    }
    return x * f;
}

void print(i128 x)
{
    if(x < 0)
    {
        putchar('-');
        x = -x;
    }
    if(x > 9) print(x / 10);
    putchar(x % 10 + '0');
}

i128 mul(i128 a, i128 b, i128 p)
{
    a %= p, b %= p;
    i128 ans = 0;
    while(b)
    {
```

```cpp
        if(b & 1) ans = (ans + a) % p;
        a = (a + a) % p;
        b >>= 1;
    }
    return ans;
}

i128 qmi(i128 a, i128 b, i128 p)
{
    i128 ans = 1;
    while(b)
    {
        if(b & 1) ans = mul(ans, a, p);
        a = mul(a, a, p);
        b >>= 1;
    }
    return ans;
}

bool Miller_Rabin(i128 n)
{
    if(n < 3 || n % 2 == 0) return n == 2;
    if(n % 3 == 0) return n == 3;
    i128 u = n - 1, t = 0;
    while(u % 2 == 0) u /= 2, t++;

    // test_time 为测试次数, 建议设为不小于 8 的整数
    int test_time = 10;
    for(int i = 0; i < test_time; i++)
    {
        // 0, 1, n-1 可以直接通过测试, a 取值范围 [2, n-2]
        i128 a = rand() % (n - 3) + 2, v = qmi(a, u, n);
        if(v == 1) continue;
        int s;
        for(s = 0; s < t; s++)
        {
            if(v == n - 1) break;
            v = mul(v, v, n);
        }
        if(s == t) return 0;
    }
    return 1;
}

signed main()
{
    i128 n = read();
    if(Miller_Rabin(n)) puts("Yes");
    else puts("No");
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
using i128 = __int128;

namespace Miller_Rabin {
    inline i128 read()
    {
        i128 x = 0, f = 1;
        char ch = getchar();
        while(ch < '0' || ch > '9')
        {
            if(ch == '-') f = -1;
            ch = getchar();
        }
        while(ch >= '0' && ch <= '9')
        {
            x = (x << 1) + (x << 3) + (ch - '0');
            ch = getchar();
        }
        return x * f;
    }

    void print(i128 x)
    {
        if(x < 0)
        {
            putchar('-');
            x = -x;
        }
        if(x > 9) print(x / 10);
        putchar(x % 10 + '0');
    }

    i128 mul(i128 a, i128 b, i128 p)
    {
        a %= p, b %= p;
        i128 ans = 0;
        while(b)
        {
            if(b & 1) ans = (ans + a) % p;
            a = (a + a) % p;
            b >>= 1;
        }
        return ans;
    }

    i128 qmi(i128 a, i128 b, i128 p)
    {
        i128 ans = 1;
        while(b)
        {
            if(b & 1) ans = mul(ans, a, p);
            a = mul(a, a, p);
            b >>= 1;
```

```cpp
        }
        return ans;
    }

    // 2, 7, 61 适用 4e9
    // 2, 3, 5, 7, 11, 13, 17 适用 3e14
    // 2, 3, 7, 61, 24251 适合 1e16 特判 46856248255981
    static constexpr int primelist[] = {2, 325, 9375, 28178, 450775,
9780504, 1795265022};
    bool miller_rabin(i128 n)
    {
        if (n < 3 || n % 2 == 0) return n == 2;
        i128 a = n - 1, b = 0;
        while (a % 2 == 0) a >>= 1, b++;

        for (int i : primelist)
        {
            i128 v = qmi(i, a, n);
            if(v == 1) continue;
            for (int j = 0; j < b || (b = -1, false); j++)
            {
                if(v == n - 1) break;
                v = mul(v, v, n);
            }
            if (b < 0) return false;
        }
        return true;
    }
}
using namespace Miller_Rabin;

signed main()
{
    i128 n = read();
    if(miller_rabin(n)) puts("Yes");
    else puts("No");
}
```

Pollard rho 大整数分解

Pollard rho 算法

- 作用: 快速地分解一个整数
- 假设当前有一个合数 n
- 核心 idea:

idea : 给定一个模 n 意义下的多项式 $g(x)$, (比如选取 $g(x) = (x^2 + c) \ mod \ n$) , 它充当一个伪随机数生成器. 选定任一 $0 \leq x_0 < n$, 用 $g(x)$ 生成一个序列 $x_i = g(x_{i-1})$, $(i > 0)$. 序列 x_i 会循环, 假设 p 是 n 的一个因子, 则 $\{x_i \ mod \ p\}$ 也会成环, 且环长只可能更短

- 复杂度: $O(n^{1/4})$

```
namespace Let_it_Rot {
    typedef long long ll;
    using f64 = long double;

    ll p;
    f64 invp;

    void setmod(ll x)
    {
        p = x, invp = (f64) 1 / x;
    }

    ll mul(ll a, ll b)
    {
        ll z = a * invp * b + 0.5;
        ll res = a * b - z * p;
        return res + (res >> 63 & p);
    }

    ll pow(ll a, ll x, ll res = 1)
    {
        for(; x; x >>= 1, a = mul(a, a))
            if(x & 1) res = mul(res, a);
        return res;
    }

    bool checkprime(ll p)
    {
        if(p == 1) return 0;
        setmod(p);
        ll d = __builtin_ctzll(p - 1), s = (p - 1) >> d;
        for(ll a: {2, 3, 5, 7, 11, 13, 82, 373})
        {
            if(a % p == 0) continue;
            ll x = pow(a, s), y;
            for(int i = 0; i < d; i++, x = y)
            {
                y = mul(x, x);
                if(y == 1 && x != 1 && x != p - 1)
```

```
                    return 0;
            }
            if(x != 1) return 0;
        }
        return 1;
    }

    ll rho(ll n)
    {
        if(!(n & 1)) return 2;
        static std:: mt19937_64 gen(((size_t)"evilboy"));
        ll x = 0, y = 0, prod = 1;
        auto f = [&](ll o) {return mul(o, o) + 1;};
        setmod(n);
        for(int t = 30, z = 0; t % 64 || std::__gcd(prod, n) == 1; t++)
        {
            if(x == y) x = ++z, y = f(x);
            if(ll q = mul(prod, x + n - y)) prod = q;
            x = f(x), y = f(f(y));
        }
        return std::__gcd(prod, n);
    }
    std::vector<ll> factor(ll x)
    {
        std::vector<ll> res;
        auto f = [&](auto f, ll x)
        {
            if(x == 1) return;
            if(checkprime(x)) return res.push_back(x);
            ll y = rho(x);
            f(f, y), f(f, x / y);
        };
        f(f, x), sort(res.begin(), res.end());
        return res;
    }
}
using namespace Let_it_Rot;
```

双模哈希封装

```cpp
namespace Hash {
    const int N = 1000010;
    typedef unsigned long long ull;
    typedef pair<ull,ull> puu;
    ull base = 131;
    ull mod1 = 25165843, mod2 = 1610612741;
    ull bs1[N], bs2[N];

    void init() {
        bs1[0] = bs2[0] = 1;
        for(int i = 1; i < N; i++) {
            bs1[i] = bs1[i - 1] * base % mod1;
            bs2[i] = bs2[i - 1] * base % mod2;
        }
    }

    struct String
    {
        int n;
        string s;
        vector<ull> h1, h2;

        String(string _s) {
            n = _s.size();
            s = " " + _s;
            h1.resize(n + 1, 0), h2.resize(n + 1, 0);
            for(int i = 1; i <= n; i++) {
                h1[i] = (h1[i - 1] * base % mod1 + (ull)s[i]) % mod1;
                h2[i] = (h2[i - 1] * base % mod2 + (ull)s[i]) % mod2;
            }
        }
        ull get_hash1(int l, int r) {
            return (h1[r] - h1[l - 1] * bs1[r - l + 1] % mod1 + mod1) %
mod1;
        }
        ull get_hash2(int l, int r) {
            return (h2[r] - h2[l - 1] * bs2[r - l + 1] % mod2 + mod2) %
mod2;
        }
        puu get_hash(int l, int r) {
            return {get_hash1(l, r), get_hash2(l, r)};
        }
        puu get_hash(void) {
            return get_hash(1, n);
        }
    };
}
using namespace Hash;
```

哈希 + 数据结构 (卡自然溢出)

哈希类

```
namespace Hash {
    typedef unsigned long long ull;
    const int N = 500010;
    ull base = 13131, bas[N];

    inline void init_base() {
        bas[0] = 1;
        for(int i = 1; i < N; i++)
            bas[i] = bas[i - 1] * base;
    }

    struct HashNode {
        ull val;
        int len;
        HashNode(){}
        HashNode(ull a, int b): val(a), len(b) {}
    };
    inline HashNode operator + (const HashNode& a, const HashNode &b) {
        HashNode res;
        res.val = a.val * bas[b.len] + b.val;
        res.len = a.len + b.len;
        return res;
    }
    inline HashNode operator - (const HashNode& a, const HashNode &b) {
        HashNode res;
        res.val = a.val - b.val * bas[a.len - b.len];
        res.len = a.len - b.len;
        return res;
    }
    inline bool operator == (const HashNode& a, const HashNode& b) {
        return (a.len == b.len) && (a.val == b.val);
    }
}
```

动态字符串哈希 (线段树维护字符串哈希)

$O(\log n)$ 单点修改, $O(\log n)$ 区间查询

```
namespace Hash {
    typedef unsigned long long ull;
    const int N = 500010;
    ull base = 13131, bas[N];

    inline void init_base() {
        bas[0] = 1;
        for(int i = 1; i < N; i++)
            bas[i] = bas[i - 1] * base;
```

```
    }

    struct HashNode {
        ull val;
        int len;
        HashNode(){}
        HashNode(ull a, int b): val(a), len(b) {}
    };
    inline HashNode operator + (const HashNode& a, const HashNode &b) {
        HashNode res;
        res.val = a.val * bas[b.len] + b.val;
        res.len = a.len + b.len;
        return res;
    }
    inline HashNode operator - (const HashNode& a, const HashNode &b) {
        HashNode res;
        res.val = a.val - b.val * bas[a.len - b.len];
        res.len = a.len - b.len;
        return res;
    }
    inline bool operator == (const HashNode& a, const HashNode& b) {
        return (a.len == b.len) && (a.val == b.val);
    }

    HashNode tr[N << 2];
    inline void pushup(int u) {
        tr[u] = tr[u << 1] + tr[u << 1 | 1];
    }
    void build(int u, int l, int r, const string& str) {
        if(l == r) {
            tr[u] = HashNode(str[l] - 'a' + 1, 1);
            return;
        }
        int mid = l + r >> 1;
        build(u << 1, l, mid, str), build(u << 1 | 1, mid + 1, r, str);
        pushup(u);
    }
    void update(int u, int l, int r, int x, char ch) {
        if(l == r) {
            tr[u] = HashNode(ch - 'a' + 1, 1);
            return;
        }
        int mid = l + r >> 1;
        if(x <= mid) update(u << 1, l, mid, x, ch);
        else update(u << 1 | 1, mid + 1, r,  x, ch);
        pushup(u);
    }
    HashNode query(int u, int l, int r, int L, int R) {
        if(l >= L && r <= R) return tr[u];
        int mid = l + r >> 1;
        HashNode res(0, 0);
        if(L <= mid) res = res + query(u << 1, l, mid, L, R);
        if(R > mid) res = res + query(u << 1 | 1, mid + 1, r, L, R);
        return res;
    }
}
```

线段树维护正反串哈希

$O(\log n)$ 单点修改, $O(\log n)$ 区间查询. 用于判断区间是否为回文串

```cpp
namespace Hash {
    typedef unsigned long long ull;
    const int N = 1000010;
    ull base = 13131, bas[N];

    inline void init_base() {
        bas[0] = 1;
        for(int i = 1; i < N; i++)
            bas[i] = bas[i - 1] * base;
    }

    struct HashNode {
        ull pval, sval; // 正串哈希, 反串哈希
        int len;
        HashNode(){}
        HashNode(ull a, ull b, int c): pval(a), sval(b), len(c) {}
    };
    inline HashNode operator + (const HashNode& a, const HashNode &b) {
        HashNode res;
        res.pval = a.pval * bas[b.len] + b.pval;
        res.sval = b.sval * bas[a.len] + a.sval;
        res.len = a.len + b.len;
        return res;
    }

    HashNode tr[N << 2];
    inline void pushup(int u) {
        tr[u] = tr[u << 1] + tr[u << 1 | 1];
    }
    void build(int u, int l, int r, const string& str) {
        if(l == r) {
            tr[u] = HashNode(str[l] - 'a' + 1, str[l] - 'a' + 1, 1);
            return;
        }
        int mid = l + r >> 1;
        build(u << 1, l, mid, str), build(u << 1 | 1, mid + 1, r, str);
        pushup(u);
    }
    void update(int u, int l, int r, int x, char ch) {
        if(l == r) {
            tr[u] = HashNode(ch - 'a' + 1, ch - 'a' + 1, 1);
            return;
        }
        int mid = l + r >> 1;
        if(x <= mid) update(u << 1, l, mid, x, ch);
        else update(u << 1 | 1, mid + 1, r,  x, ch);
        pushup(u);
    }
```

```
49      HashNode query(int u, int l, int r, int L, int R) {
50          if(l >= L && r <= R) return tr[u];
51          int mid = l + r >> 1;
52          HashNode res(0, 0, 0);
53          if(L <= mid) res = res + query(u << 1, l, mid, L, R);
54          if(R > mid) res = res + query(u << 1 | 1, mid + 1, r, L, R);
55          return res;
56      }
57  }
```

树与图上的计数问题

Prufer Code

n 个点的有标号无根树可以与一个长度为 $n-2$ 的 $Prufer$ 序列对应

从树到 Prufer 序列

- f 为空序列
- 如果当前树上多于两个节点, 假设当前标号最小的叶子为 x , 与 x 相连的节点标号为 y , 把节点 x 从树上删掉, 把 y 放到序列 f 的末尾
- 重复上述操作, 直到树上只有两个节点

```
vector<int> tree_to_prufer(vector<set<int>> e)
{
    int n = e.size();
    vector<int> f;
    priority_queue<int,vector<int>,greater<int>> leaves;
    for(int i = 0; i < n; i++)
        if(e[i].size() == 1)
            leaves.push(i);

    while(f.size() < n - 2)
    {
        int u = leaves.top();
        leaves.pop();

        int v = *e[u].begin();
        e[v].erase(u);
        f.push_back(v);
        if(e[v].size() == 1) leaves.push(v);
    }
    return f;
}
```

从 Prufer 序列到树

- 找到当前不在 f 中且还没被使用过的最小的元素 x
- x 与当前 f 的第一个元素连边, 把 x 标记为已使用过
- 删除 f 的第一个元素, 如果 f 非空重复上述操作
- 最终有两个元素没被使用, 将它们连边

```
vector<vector<int>> prufer_to_tree(vector<int> f)
{
    int n = f.size() + 2;
    vector<vector<int>> edges(n, vector<int> ());
    vector<int> cnt(n, 0);
    for(int u: f) cnt[u] ++;
    priority_queue<int,vector<int>,greater<int>> q;
    for(int i = 0; i < n; i++) if(!cnt[i]) q.push(i);

    for(int u: f)
```

```
    {
        int v = q.top();
        q.pop();
        edges[u].push_back(v);
        edges[v].push_back(u);
        cnt[u] --;
        if(!cnt[u]) q.push(u);
    }
    int u = q.top(); q.pop();
    int v = q.top(); q.pop();
    edges[u].push_back(v);
    edges[v].push_back(u);
    return edges;
}
```

上述两个映射定义了 n 个点有标号无根树的集合到长度为 $n-2$, 元素在 1 到 n 之间的序列的一个一一映射

定理 (Cayley 定理)

n 个点的有标号无根树有 n^{n-2} 种

matrix-tree theorem

定理

设无向图 $G(V,E)$, $D=diag(d(1),d(2),\cdots,d(n))$, $d(i)$ 是节点 i 的度数, G 的拉普拉斯矩阵 $L=D-E$. G 的生成树个数为 $det(L_0)$, 其中 L_0 是 L 去掉第 i 行第 i 列 (i 任选)

SA 封装

```cpp
namespace SA {
    const int N = 1000010;
    int n, m; //字符串长度、不同字符数量
    string s;
    //sa[i] 表示排名第 i 的是第几个后缀
    //x[i] 表示第 i 个后缀的第一关键字
    //y[i] 表示第 i 个后缀的第二关键字
    //c[i] 表示关键字为 i 的数的个数
    //rk[i] 表示第 i 个后缀的排名
    //height[i] 表示排名第 i 的后缀和排名第 i - 1 的后缀的最长公共前缀
    int sa[N], x[N], y[N], c[N], rk[N], height[N];

    void get_sa() //预处理 sa
    {
        //将所有后缀按照首字母从小到大排序（基数排序）
        for(int i = 1; i <= n; i++) c[x[i] = s[i]]++; //记录每个关键字出现的次数
        for(int i = 2; i <= m; i++) c[i] += c[i - 1]; //求前缀和
        for(int i = n; i >= 1; i--) sa[c[x[i]]--] = i; //为每个数安排位置

        //每一轮将后缀按照前 2k 个字符排序
        for(int k = 1; k <= n; k <<= 1)
        {
            //先将所有后缀按照第二关键字排序
            int num = 0;
            for(int i = n - k + 1; i <= n; i++) y[++num] = i;
            for(int i = 1; i <= n; i++)
                if(sa[i] > k)
                    y[++num] = sa[i] - k;

            //再将所有后缀按照第一关键字排序（基数排序）
            for(int i = 1; i <= m; i++) c[i] = 0;
            for(int i = 1; i <= n; i++) c[x[i]]++;
            for(int i = 2; i <= m; i++) c[i] += c[i - 1];
            for(int i = n; i >= 1; i--) sa[c[x[y[i]]]--] = y[i], y[i] = 0;
//y 数组清空，用于后面存储当前的第一关键字

            //将排序后的所有后缀再按照前 2k 个字符离散化
            swap(x, y);
            x[sa[1]] = 1, num = 1;
            for(int i = 2; i <= n; i++)
                x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + k] ==
y[sa[i - 1] + k]) ? num : ++num;

            if(num == n) break; //如果 n 个后缀的排名都不同，说明已经排好序
            m = num; //更新不同字符的个数
        }
    }

    void get_height() //预处理 height
    {
        for(int i = 1; i <= n; i++) rk[sa[i]] = i; //预处理 rk
        //预处理 height
```

```
        for(int i = 1, k = 0; i <= n; i++) //k 记录当前的 h[i]
        {
            if(rk[i] == 1) continue; //height[1] 默认为 0
            if(k) k--; //应该从 h[i - 1] - 1 开始枚举，也就是 k - 1
            int j = sa[rk[i] - 1]; //记录第 i 个后缀前一个排名的后缀
            //如果 i 和 j 的第 k 位相同，则最长公共前缀长度 + 1
            while(i + k <= n && j + k <= n && s[i + k] == s[j + k]) k++;
            height[rk[i]] = k; //height[rk[i]] = k
        }
    }

    void init()
    {
        s = " " + s;
        n = s.size() - 1, m = 122; //最大字符 'z' 的 ASCII 码是 122
        get_sa(); //预处理 sa
        get_height(); //预处理 height
    }
};
using namespace SA;
```

排列组合公式

排列数公式

$$A_n^m = n(n-1)(n-2)\cdots(n-m+1) = \frac{n!}{(n-m)!}$$

排列数性质

$$A_n^m = nA_{n-1}^{m-1}$$

$$A_n^m = mA_{n-1}^{m-1} + A_{n-1}^m$$

组合数公式

$$C_n^m = \frac{n(n-1)(n-2)\cdots(n-m+1)}{m!} = \frac{n!}{m!(n-m)!}$$

$$C_n^0 = C_n^n = 1$$

组合数性质

$$C_n^m = C_n^{n-m}$$

$$C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$$

组合数求和公式

$$C_n^0 + C_n^1 + C_n^2 + \cdots + C_n^n = 2^n$$

$$C_n^0 + C_n^2 + C_n^4 + \cdots = C_n^1 + C_n^3 + C_n^5 + \cdots = 2^{n-1}$$

二项式定理

$$(1+x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k = \binom{n}{0} + \binom{n}{1}x + \cdots + \binom{n}{n}x^n$$

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k}y^k$$

二项式系数

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

$$\binom{n}{0} = 1\,,\ \binom{0}{k} = 0$$

有关二项式系数的恒等式

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$$

$$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$$

$$\binom{n-1}{k} - \binom{n-1}{k-1} = \frac{n-2k}{n}\binom{n}{k}$$

$$\binom{n}{i}\binom{i}{m} = \binom{n}{m}\binom{n-m}{i-m}$$

$$\sum_{r=0}^{n} \binom{n}{r} = 2^n$$

$$\sum_{r=0}^{n-k}\frac{(-1)^r(n+1)}{k+r+1}\binom{n-k}{r}=\binom{n}{k}^{-1}$$

$$\sum_{r=0}^{n}\binom{dn}{dr}=\frac{1}{d}\sum_{r=1}^{d}(1+e^{\frac{2\pi ri}{d}})^{dn}$$

$$\sum_{i=m}^{n}\binom{a+i}{i}=\binom{a+n+1}{n}-\binom{a+m}{m-1}$$

$$\binom{a+m}{m-1}+\binom{a+m}{m}+\binom{a+m+1}{m+1}+\cdots+\binom{a+n}{n}=\binom{a+n+1}{n}$$

$$F_n=\sum_{i=0}^{\infty}\binom{ni}{i}$$

$$F_{n-1}+F_n=\sum_{i=0}^{\infty}\binom{n-1-i}{i}+\sum_{i=0}^{\infty}\binom{n-i}{i}=1+\sum_{i=1}^{\infty}\binom{n-i}{i-1}+\sum_{i=1}^{\infty}\binom{n-i}{i}=1$$

$$\sum_{i=0}^{\infty}\binom{n+1-i}{i}=F_{n+1}$$

朱世杰恒等式

$$\sum_{i=m}^{n}\binom{i}{a}=\binom{n+1}{a+1}-\binom{m}{a+1}$$

$$\binom{m}{a+1}+\binom{m}{a}+\binom{m+1}{a}+\cdots+\binom{n}{a}=\binom{n+1}{a+1}$$

二阶求和公式

$$\sum_{r=0}^{n}\binom{n}{r}^2=\binom{2n}{n}$$

$$\sum_{i=0}^{r}\binom{r_1+n-1-i}{r_1-1}\binom{r_2+i-1}{r_2-1}=\binom{r_1+r_2+n-1}{r_1+r_2-1}$$

$$(1-x)^{-r_1}(1-x)^{-r_2}=(1-x)^{-r_1-r_2}$$

$$(1-x)^{-r_1}(1-x)^{-r_2}=(\sum_{n=0}^{\infty}\binom{r_1+n-1}{r_1-1}x^n)(\sum_{n=0}^{\infty}\binom{r_2+n-1}{r_2-1}x^n)=\sum_{n=0}^{\infty}(\sum_{i=0}^{n}\binom{r_1+n-1-i}{r_1-1}\binom{r_2+i-1}{r_2-1})x^n$$

$$(1-x)^{-r_1-r_2}=\sum_{n=0}^{\infty}\binom{r_1+r_2+n-1}{r_1+r_2-1}x^n$$

范德蒙恒等式

$$\sum_{i=0}^{k}\binom{n}{i}\binom{m}{k-i}=\binom{n+m}{k}$$

三阶求和公式 李善兰恒等式

$$\binom{n+k}{k}^2=\sum_{j=0}^{k}\binom{k}{j}^2\binom{n+2k-j}{2k}$$

北京大学

PEKING UNIVERSITY

Let it Rot

Coach

张勤健 Qinjian Zhang

罗国杰 Guojie Luo

Contestant

钱易 Yi Qian

彭博 Bo Peng

冯施源 Shiyuan Feng

ICPC World Finals Luxor

Contents

1 图论

1.1 欧拉回路

```
namespace Euler {
  | bool directed;
  | vector<pii>V[sz];
```

```
  vector<int>ans; // reverse ans in the end
  int vis[sz];
  int dfs(int x) {
    vector<int>t;
    while (V[x].size()) {
      auto [to,id]=V[x].back();
      V[x].pop_back();
      if (!vis[abs(id)])
        ↪ vis[abs(id)]=1,t.push_back(dfs(to)),ans.push_back(id);
    }
    rep(i,1,(int)t.size()-1) if (t[i]!=x) ans.clear();
    return t.size()?t[0]:x;
  }
  int n,m;
  pii e[sz];
  int deg[sz],vv[sz];
  void clr() {
    rep(i,1,n) V[i].clear(),deg[i]=vv[i]=0;
    rep(i,1,m) vis[i]=0;
    ans.clear();
    n=m=0;
  }
  void addedge(int x,int y) {
    chkmax(n,x),chkmax(n,y); ++m;
    e[m]={x,y};
    if (directed) {
      V[x].push_back({y,m});
      ++deg[x],--deg[y],vv[x]=vv[y]=1;
    }
    else {
      V[x].push_back({y,m});
      V[y].push_back({x,-m});
      ++deg[x],++deg[y],vv[x]=vv[y]=1;
    }
  }
  using vi=vector<int>;
  pair<vi,vi> work() {
    if (!m) return clr(),pair<vi,vi>{{1},{}};
    int S=1;
    rep(i,1,n) if (vv[i]) S=i;
    rep(i,1,n) if (deg[i]>0&&deg[i]%2==1) S=i;
    dfs(S);
    if ((int)ans.size()!=m) return clr(),pair<vi,vi>();
    reverse(ans.begin(),ans.end());
    vi ver,edge=ans;
    if (directed) {
      ver={e[ans[0]].fir};
      for (auto t:ans) ver.push_back(e[t].sec);
    }
    else {
      ver={ans[0]>0?e[ans[0]].fir:e[-ans[0]].sec};
      for (auto t:ans) ver.push_back(t>0?e[t].sec:e[-t].fir);
    }
    clr();
    return {ver,edge};
  }
}
```

1.2 二分图匹配 | 最小边覆盖

```
// 匈牙利，左到右单向边，O(M|match|)
std::vector<int> edge[N];
bool dfs(int x, std::vector<int> & vis, std::vector<int> & match) {
  for(int y : edge[x]) if(!vis[y])
    if(vis[y] = 1, !match[y] || dfs(match[y], vis, match))
      return match[y] = x, 1;
  return 0;
}
std::vector<int> match(int nl, int nr) {
  std::vector<int> vis(nr + 1), match(nr + 1), ret(nl + 1);
  for(int i = 1;i <= nl;++i) if(dfs(i, vis, match))
    memset(vis.data(), 0, vis.size() << 2);
  for(int i = 1;i <= nr;++i) ret[match[i]] = i;
  return ret[0] = 0, ret;
}
// 最小边覆盖
std::pair<std::vector<int>, std::vector<int>> minedgecover(int nl, int nr) {
  std::vector<int> vis(nr + 1), match(nr + 1), ret(nl + 1);
  for(int i = 1;i <= nl;++i) if(dfs(i, vis, match))
    memset(vis.data(), 0, vis.size() << 2);
  for(int i = 1;i <= nr;++i) ret[match[i]] = i;
  ret[0] = 0;
  for(int i = 1;i <= nl;++i) if(!ret[i]) dfs(i, vis, match);
  std::vector<int> le, ri;
  for(int i = 1;i <= nl;++i) if(ret[i] && !vis[ret[i]]) le.push_back(i);
  for(int i = 1;i <= nr;++i) if(vis[i]) ri.push_back(i);
  return std::make_pair(le, ri);
}
// 匈牙利，左到右单向边，bitset，O(n^2/w|match|)
using set = std::bitset<N>;
set edge[N];
bool dfs(int x, set & unvis, std::vector<int> & match) {
  for(set z = edge[x];;) {
    z &= unvis;
    int y = z._Find_first();
    if(y == N) return 0;
    if(unvis.reset(y), !match[y] || dfs(match[y], unvis, match))
      return match[y] = x, 1;
  }
}
std::vector<int> match(int nl, int nr) {
  set unvis; unvis.set();
  std::vector<int> match(nr + 1), ret(nl + 1);
  for(int i = 1;i <= nl;++i)
    if(dfs(i, unvis, match))
      unvis.set();
  for(int i = 1;i <= nr;++i) ret[match[i]] = i;
```

```
  | return ret[0] = 0, ret;
}
// HK, 左到右单向边, O(M√|match|)
std::vector<int> e[N];
std::vector<int> matchl, matchr, a, p;
std::vector<int> match(int nl, int nr) {
  | matchl.assign(nl + 1, 0), matchr.assign(nr + 1, 0);
  | for(;;) {
  |   | a.assign(nl + 1, 0), p.assign(nl + 1, 0);
  |   | static std::queue<int> Q;
  |   | for(int i = 1;i <= nl;++i)
  |   |   | if(!matchl[i]) a[i] = p[i] = i, Q.push(i);
  |   | int succ = 0;
  |   | for(;Q.size();) {
  |   |   | int x = Q.front(); Q.pop();
  |   |   | if(matchl[a[x]]) continue;
  |   |   | for(int y : e[x]) {
  |   |   |   | if(!matchr[y]) {
  |   |   |   |   | for(succ = 1;y;x = p[x])
  |   |   |   |   |   | matchr[y] = x, std::swap(matchl[x], y);
  |   |   |   |   | break;
  |   |   |   | }
  |   |   |   | if(!p[matchr[y]])
  |   |   |   |   | Q.push(y = matchr[y]), p[y] = x, a[y] = a[x];
  |   |   | }
  |   | }
  |   | if(!succ) break;
  | }
  | return matchl;
} // matchl 是左边每个点匹配的右边点编号
std::pair<std::vector<int>, std::vector<int>> minedgecover(int nl, int nr) {
  | match(nl, nr);
  | std::vector<int> l, r;
  | for(int i = 1;i <= nl;++i) if(!a[i]) l.push_back(i);
  | for(int i = 1;i <= nr;++i) if(a[matchr[i]]) r.push_back(i);
  | return {l, r};
}
```

1.3 网络最大流 | dinic

```
// S 编号最小, T 最大, 或者改一下清空
struct Dinic {
  | struct T {
  |   | int to, nxt, v;
  | } e[N << 3];
  | int h[N], head[N], num = 1;
  | void link(int x, int y, int v) {
  |   | e[++num] = {y, h[x], v}, h[x] = num;
  |   | e[++num] = {x, h[y], 0}, h[y] = num; // !!!
  | }
  | int dis[N];
  | bool bfs(int s, int t) {
  |   | for(int i = s;i <= t;++i) dis[i] = -1, head[i] = h[i]; //如果编号不是
  |   |   ↪ [S,T], 只要改这里
  |   | for(Q.push(t), dis[t] = 0;!Q.empty();) {
  |   |   | int x = Q.front(); Q.pop();
  |   |   | for(int i = h[x];i;i = e[i].nxt) if(e[i ^ 1].v && dis[e[i].to] < 0)
  |   |   |   ↪ {
  |   |   |   | dis[e[i].to] = dis[x] + 1, Q.push(e[i].to);
  |   |   | }
  |   | }
  |   | return dis[s] >= 0;
  | }
  | int dfs(int s, int t, int lim) {
  |   | if(s == t || !lim) return lim;
  |   | int ans = 0, mn;
  |   | for(int & i = head[s];i;i = e[i].nxt) {
  |   |   | if(dis[e[i].to] + 1 == dis[s] && (mn = dfs(e[i].to, t, std::min(lim,
  |   |   |   ↪ e[i].v)))) {
  |   |   |   | e[i].v -= mn, e[i ^ 1].v += mn;
  |   |   |   | ans += mn, lim -= mn;
  |   |   |   | if(!lim) break;
  |   |   | }
  |   | }
  |   | return ans;
  | }
  | int flow(int s, int t) {
  |   | int ans = 0;
  |   | for(;bfs(s, t);) ans += dfs(s, t, 1e9);
  |   | return ans;
  | }
} G;
```

1.4 最小费用流

```
// S 编号最小, T 最大, 或者改一下清空
namespace mcmf {
  | using pr = std::pair<ll, int>;
  | const int N = 10005, M = 1e6 + 10;
  | struct edge {
  |   | int to, nxt, v, f;
  | } e[M << 1];
  | int h[N], num = 1;
  | void link(int x, int y, int v, int f) {
  |   | e[++num] = {y, h[x], v, f}, h[x] = num;
  |   | e[++num] = {x, h[y], 0, -f}, h[y] = num;
  | }
  | ll d[N], dis[N];
  | int vis[N], fr[N];
  | bool spfa(int s, int t) {
  |   | std::queue<int> Q;
  |   | std::fill(d + s, d + t + 1, 1e18); // CHECK
  |   | for(d[s] = 0, Q.push(s);!Q.empty();) {
  |   |   | int x = Q.front(); Q.pop(); vis[x] = 0;
  |   |   | for(int i = h[x];i;i = e[i].nxt)
  |   |   |   | if(e[i].v && d[e[i].to] > d[x] + e[i].f) {
```

```
        d[e[i].to] = d[x] + e[i].f;
        fr[e[i].to] = i;
        if(!vis[e[i].to]) vis[e[i].to] = 1, Q.push(e[i].to);
      }
    }
    return d[t] < 1e17;
  }
  bool dijkstra(int s, int t) { // 正常题目不需要 dijk
    std::priority_queue<pr, std::vector<pr>, std::greater<pr>> Q;
    for(int i = s;i <= t;++i) dis[i] = d[i], d[i] = 1e18, vis[i] = fr[i] =
      ↪0; // CHECK
    for(Q.emplace(d[s] = 0, s);!Q.empty();) {
      int x = Q.top().second; Q.pop();
      if(vis[x]) continue;
      vis[x] = 1;
      for(int i = h[x];i;i = e[i].nxt) {
        const ll v = e[i].f + dis[x] - dis[e[i].to];
        if(e[i].v && d[e[i].to] > d[x] + v) {
          fr[e[i].to] = i;
          Q.emplace(d[e[i].to] = d[x] + v, e[i].to);
        }
      }
    }
    for(int i = s;i <= t;++i) d[i] += dis[i]; // CHECK
    return d[t] < 1e17;
  }
  std::pair<ll, ll> EK(int s, int t) {
    spfa(s, t); // 如果初始有负权且要 dijk
    ll f = 0, c = 0;
    for(;dijkstra(s, t);) { // 正常可以用 spfa
      ll fl = 1e18;
      for(int i = fr[t];i;i = fr[e[i ^ 1].to]) fl = std::min<ll>(e[i].v,
        ↪fl);
      for(int i = fr[t];i;i = fr[e[i ^ 1].to]) e[i].v -= fl, e[i ^ 1].v +=
        ↪fl;
      f += fl, c += fl * d[t];
    }
    return std::make_pair(f, c);
  }
}
// in flow problems with lower bounds (or with negative cycles), flow the
  ↪negative edges first
// after the first round, revert the auxiliary edges
```

1.5 二分图最大权匹配 | KM

```
namespace KM {
  int nl,nr;
  ll e[sz][sz];
  ll lw[sz],rw[sz];
  int lpr[sz],rpr[sz];
  int vis[sz],fa[sz];
  ll mnw[sz];
  void work(int x) {
    int xx=x;
    rep(i,1,nr) vis[i]=0,mnw[i]=1e18;
    while (233) {
      rep(i,1,nr) if (!vis[i]&&chkmin(mnw[i],lw[x]+rw[i]-e[x][i]))
        ↪fa[i]=x;
      ll mn=1e18; int y=-1;
      rep(i,1,nr) if (!vis[i]&&chkmin(mn,mnw[i])) y=i;
      lw[xx]-=mn; rep(i,1,nr) if (vis[i]) rw[i]+=mn,lw[rpr[i]]-=mn; else
        ↪mnw[i]-=mn;
      if (rpr[y]) x=rpr[y],vis[y]=1; else { while (y)
        ↪rpr[y]=fa[y],swap(y,lpr[fa[y]]); return; }
    }
  }
  void init(int nl,int nr) {
    assert(nl<=nr);
    KM::nl=nl,KM::nr=nr;
    rep(i,1,nl) lw[i]=-1e18;
    rep(i,1,nl) rep(j,1,nr) e[i][j]=0; // or -1e18
  }
  void clr() {
    rep(i,1,nl) lpr[i]=lw[i]=0;
    rep(i,1,nr) rpr[i]=rw[i]=vis[i]=fa[i]=mnw[i]=0;
    rep(i,1,nl) rep(j,1,nr) e[i][j]=0;
  }
  void addedge(int x,int y,ll w){chkmax(e[x][y],w),chkmax(lw[x],w);}
  ll work() {
    rep(i,1,nl) work(i);
    ll tot=0;
    rep(i,1,nl) tot+=e[i][lpr[i]];
    return tot;
  }
}
```

1.6 一般图最大匹配 | 带花树

```
namespace blossom {
  vector<int>V[sz];
  int f[sz];
  int n,match[sz];
  int getfa(int x){return f[x]==x?x:f[x]=getfa(f[x]);}
  void link(int x,int y){V[x].push_back(y),V[y].push_back(x);}
  int pre[sz],mk[sz];
  int vis[sz],T;
  queue<int>q;
  int LCA(int x,int y) {
    T++;
    for (;;x=pre[match[x]],swap(x,y))
      if (vis[x=getfa(x)]==T) return x;
      else vis[x]=x?T:0;
  }
  void flower(int x,int y,int z) {
    while (getfa(x)!=z) {
      pre[x]=y; y=match[x]; f[x]=f[y]=z; x=pre[y];
      if (mk[y]==2) q.push(y),mk[y]=1;
    }
```

```
  }
  void aug(int s){
    for (int i=1;i<=n;i++) pre[i]=mk[i]=vis[i]=0,f[i]=i;
    q={};
    mk[s]=1; q.push(s);
    while (q.size()) {
      int x=q.front(); q.pop();
      for (auto v:V[x]) {
        int y=v,z;
        if (mk[y]==2) continue;
        if (mk[y]==1) z=LCA(x,y),flower(x,y,z),flower(y,x,z);
        else if (!match[y]) {
          for (pre[y]=x;y;) x=pre[y],match[y]=x,swap(y,match[x]);
          return;
        }
        else pre[y]=x,mk[y]=2,q.push(match[y]),mk[match[y]]=1;
      }
    }
  }
  int work() {
    rep(i,1,n) if (!match[i]) aug(i);
    int res=0;
    rep(i,1,n) res+=match[i]>i;
    return res;
  }
}
```

1.7 最小树形图

抄罗大的，返回值是边的集合，如果没有最小树形图会返回空集（注意 $n = 1$），可以修改建图。

```
namespace DMST {
  struct edge {
    int u, v, id; ll w;
    bool operator < (const edge & y) const {
      return w < y.w;
    }
  } ent[N], val[M];
  int ls[M], rs[M], size[M], cc; ll tag[M];
  int fs[N], fw[N], rt[N];
  void put(int x, ll v) {
    if(x) val[x].w += v, tag[x] += v;
  }
  void pushdown(int x) {
    put(ls[x], tag[x]);
    put(rs[x], tag[x]);
    tag[x] = 0;
  }
  int merge(int x, int y) {
    if(!x || !y) return x | y;
    if(val[y] < val[x]) std::swap(x, y);
    pushdown(x), rs[x] = merge(rs[x], y);
    if(size[rs[x]] > size[ls[x]]) {
      std::swap(ls[x], rs[x]);
    }
    size[x] += size[y];
    return x;
  }
  void ins(int & x, const edge & z) {
    val[++cc] = z, size[cc] = 1;
    x = merge(x, cc);
  }
  void pop(int & x) { x = merge(ls[x], rs[x]); }
  edge top(int x) { return val[x]; }
  int find(int x, int * anc) {
    return anc[x] == x ? x : anc[x] = find(anc[x], anc);
  }
  void link(int u, int v, int w, int id) {
    ins(rt[v], {u, v, id, w});
  }
  int pa[N * 2], tval[N * 2], up[N * 2], end_edge[M], cmt, baned[M];
  std::vector<int> solve(int r) {
    std::queue<int> roots;
    for(int i = 1;i <= n;++i) {
      fs[i] = fw[i] = i, tval[i] = ++ cmt;
      if(i != r) roots.push(i);
    }
    std::vector<edge> H;
    std::vector<int> ret;
    for(;!roots.empty();) {
      int k = roots.front(); roots.pop();
      if(!rt[k]) return ret;
      edge e = top(rt[k]); pop(rt[k]);
      int i = e.u, j = e.v;
      if(find(i, fs) == k) roots.push(k);
      else {
        H.push_back(e); end_edge[e.id] = tval[k];
        if(find(i, fw) != find(j, fw)) {
          fw[find(j, fw)] = i;
          ent[k] = e;
        } else {
          pa[tval[k]] = ++ cmt, up[tval[k]] = e.id;
          put(rt[k], -e.w);
          for(;(e = ent[find(e.u, fs)]).u;) {
            int p = find(e.v, fs);
            pa[tval[p]] = cmt;
            up[tval[p]] = e.id;
            put(rt[p], -e.w);
            rt[k] = merge(rt[k], rt[p]);
            fs[p] = k;
          }
          tval[k] = cmt;
          roots.push(k);
        }
      }
    }
    reverse(H.begin(), H.end());
    for(edge i : H) if(!baned[i.id]) {
```

```
    |   |   | ret.push_back(i.id);
    |   |   | for(int j = i.v;j != end_edge[i.id];j = pa[j]) ++ baned[up[j]];
    |   | }
    |   | sort(ret.begin(), ret.end());
    |   | return ret;
    | }
}
```

1.8 缩点 | kasaraju

时间复杂度 $O(\frac{n^2}{w})$, 可以对于边修改不多的图快速计算。

```
using set = std::bitset<N>;
// re 是反向边, 需要连好
set e[N], re[N], vis;
std::vector<int> sta;
void dfs0(int x, set * e) {
  | vis.reset(x);
  | for(;;) {
  |   | int go = (e[x] & vis)._Find_first();
  |   | if(go == N) break;
  |   | dfs0(go, e);
  | }
  | sta.push_back(x);
}
std::vector<std::vector<int>> solve() {
  | vis.set();
  | for(int i = 1;i <= n;++i) if(vis.test(i)) dfs0(i, e);
  | vis.set();
  | auto s = sta;
  | std::vector<std::vector<int>> ret;
  | for(int i = n - 1;i >= 0;--i) if(vis.test(s[i])) {
  |   | sta.clear(), dfs0(s[i], re), ret.push_back(sta);
  | }
  | return ret;
}
```

1.9 缩点 | Tarjan

```
int dfn[sz],low[sz],cc;
stack<int>S; int in[sz];
int bel[sz],T;
void dfs(int x) {
  | dfn[x]=low[x]=++cc; S.push(x),in[x]=1;
  | for (auto v:V[x]) {
  |   | if (!dfn[v]) dfs(v,x),chkmin(low[x],low[v]);
  |   | else if (in[v]) chkmin(low[x],dfn[v]);
  | }
  | if (dfn[x]==low[x]) {
  |   | int y; ++T;
  |   | do y=S.top(),S.pop(),in[y]=0,bel[y]=T; while (y!=x);
  | }
}
```

1.10 缩点 | 点双

```
int dfn[sz],low[sz],cc;
stack<int>S;
int T;
void dfs(int x,int fa) {
  | dfn[x]=low[x]=++cc; S.push(x);
  | for (auto v:V[x]) if (v!=fa) {
  |   | if (!dfn[v]) dfs(v,x),chkmin(low[x],low[v]);
  |   | else chkmin(low[x],dfn[v]);
  | }
  | if (fa&&low[x]>=dfn[fa]) {
  |   | int y; ++T;
  |   | do y=S.top(),S.pop(),V2[T].push_back(y),V2[y].push_back(T); while (y!
  |   |   ↪ =x);
  |   | V2[T].push_back(fa),V2[fa].push_back(T);
  | }
}
```

1.11 缩点 | 边双

```
int dfn[sz],low[sz],cc;
stack<int>S;
int bel[sz],T;
void dfs(int x,int fa) { // cannot handle multiple edges
  | dfn[x]=low[x]=++cc; S.push(x);
  | for (auto v:V[x]) if (v!=fa) {
  |   | if (!dfn[v]) dfs(v,x),chkmin(low[x],low[v]);
  |   | else chkmin(low[x],dfn[v]);
  | }
  | if (dfn[x]==low[x]) {
  |   | int y; ++T;
  |   | do y=S.top(),S.pop(),bel[y]=T; while (y!=x);
  | }
}
```

1.12 仙人掌

```
vector<int>V2[sz],V[sz]; // V2: cactus edges; V: reconstructed tree edges
int m; // set to n before dfs
void dfs(int x,int f) {
  | static int mark[sz],fa[sz],vis[sz],dep[sz];
  | fa[x]=f; vis[x]=1;dep[x]=dep[f]+1;
  | for (auto v:V2[x]) if (v!=f) {
  |   | if (!vis[v]) dfs(v,x);
  |   | else if (dep[v]<dep[x]) {
  |   |   | ++m;
  |   |   | V[v].push_back(m);
  |   |   | for (int y=x;y!=v;y=fa[y]) V[m].push_back(y),mark[y]=1;
  |   | }
  | }
  | if (!mark[x]) {
  |   | ++m;
  |   | V[fa[x]].push_back(m),V[m].push_back(x);
  | }
}
```

1.13 2-Sat

```
rep(i,1,n) if (bel[i<<1]==bel[i<<1|1]) return puts("IMPOSSIBLE"),0;
puts("POSSIBLE");
rep(i,1,n) printf("%d ",bel[i<<1]>bel[i<<1|1]);
```

1.14 支配树

```
namespace BuildTree {
  int idom[sz];
  vector<int>V[sz],ANS[sz]; // ANS: final tree
  int deg[sz];
  int fa[sz][25],dep[sz];
  int lca(int x,int y) {
    if (dep[x]<dep[y]) swap(x,y);
    drep(i,20,0)
      if (fa[x][i]&&dep[fa[x][i]]>=dep[y])
        x=fa[x][i];
    if (x==y) return x;
    drep(i,20,0)
      if (fa[x][i]!=fa[y][i])
        x=fa[x][i],y=fa[y][i];
    return fa[x][0];
  }
  void work() {
    queue<int>q;q.push(1);
    while (!q.empty()) {
      int x=q.front();q.pop();
      ANS[idom[x]].push_back(x);fa[x][0]=idom[x];dep[x]=dep[idom[x]]+1;
      rep(i,1,20) fa[x][i]=fa[fa[x][i-1]][i-1];
      for (int v:V[x]) {
        --deg[v];if (!deg[v]) q.push(v);
        if (!idom[v]) idom[v]=x;
        else idom[v]=lca(idom[v],x);
      }
    }
  }
}
namespace BuildDAG {
  vector<int>V[sz],rV[sz];
  int dfn[sz],id[sz],anc[sz],cnt;
  void dfs(int x) {
    id[dfn[x]=++cnt]=x;
    for (int v:V[x]) if (!dfn[v])
      BuildTree::V[x].push_back(v), BuildTree::deg[v]++, anc[v] = x,
        ↪ dfs(v);
  }
  int fa[sz],mn[sz];
  int find(int x) {
    if (x==fa[x]) return x;
    int tmp=fa[x];fa[x]=find(fa[x]);
    chkmin(mn[x],mn[tmp]);
    return fa[x];
  }
  int semi[sz];
  void work() {
    dfs(1);
    rep(i,1,n) fa[i]=i,mn[i]=1e9,semi[i]=i;
    drep(w,n,2) {
      int x=id[w];int cur=1e9;
      if (w>cnt) continue;
      for (int v:rV[x]) {
        if (!dfn[v]) continue;
        if (dfn[v]<dfn[x]) chkmin(cur,dfn[v]);
        else find(v),chkmin(cur,mn[v]);
      }
      semi[x]=id[cur];mn[x]=cur;fa[x]=anc[x];
      BuildTree::V[semi[x]].push_back(x);BuildTree::deg[x]++;
    }
  }
  void link(int x,int y){V[x].push_back(y),rV[y].push_back(x);}
}
```

1.15 三/四元环

```
static int id[sz],rnk[sz];
rep(i,1,n) id[i]=i;
sort(id+1,id+n+1,[](int x,int y){return pii{deg[x],x}<pii{deg[y],y};});
rep(i,1,n) rnk[id[i]]=i;
rep(i,1,n) for (auto v:V[i]) if (rnk[v]>rnk[i]) V2[i].push_back(v);
int ans3=0; // 3-cycle
rep(i,1,n) {
  static int vis[sz];
  for (auto v:V2[i]) vis[v]=1;
  for (auto v1:V2[i]) for (auto v2:V2[v1]) if (vis[v2]) ++ans3; // (i,v1,v2)
  for (auto v:V2[i]) vis[v]=0;
}
ll ans4=0; // 4-cycle
rep(i,1,n) {
  static int vis[sz];
  for (auto v1:V[i]) for (auto v2:V2[v1]) if (rnk[v2]>rnk[i])
    ↪ ans4+=vis[v2],vis[v2]++;
  for (auto v1:V[i]) for (auto v2:V2[v1]) vis[v2]=0;
}
```

1.16 双极定向

```
vector<int>G[sz];
namespace bipolar_orientation {
  int dfn[sz],low[sz],cc,p[sz],inv[sz],topo[sz];
  bool flg,sgn[sz];
  void dfs(int x,int fa,int s,int t) {
    dfn[x]=low[x]=++cc; inv[cc]=x,p[x]=fa;
    if (x==s) dfs(t,x,s,t);
    for (int y:G[x]) {
      if (x==s&&y==t) continue;
      if (!dfn[y]) {
        dfs(y,x,s,t);
        chkmin(low[x],low[y]);
        if (x==s||low[y]>=dfn[x]) flg=1;
```

```
    |   |   | }
    |   |   | else if (dfn[y]<dfn[x]&&y!=fa) chkmin(low[x],dfn[y]);
    |   | }
    | }
    | int check(int s,int t,int n) { // return topo
    |   | if (n==1) return topo[1]=1,1;
    |   | if (s==t) return 0;
    |   | cc=flg=0; dfs(s,s,s,t);
    |   | if (flg) return 0;
    |   | sgn[s]=0;
    |   | static int pre[sz],suf[sz];
    |   | suf[0]=s,pre[s]=0,suf[s]=t;
    |   | pre[t]=s,suf[t]=n+1,pre[n+1]=t;
    |   | rep(i,3,n) {
    |   |   | int v=inv[i];
    |   |   | if (!sgn[inv[low[v]]]) {
    |   |   |   | int P=pre[p[v]];
    |   |   |   | pre[v]=P,suf[v]=p[v];
    |   |   |   | suf[P]=pre[p[v]]=v;
    |   |   | }
    |   |   | else {
    |   |   |   | int S=suf[p[v]];
    |   |   |   | pre[v]=p[v],suf[v]=S;
    |   |   |   | suf[p[v]]=pre[S]=v;
    |   |   | }
    |   |   | sgn[p[v]]=!sgn[inv[low[v]]];
    |   | }
    |   | for (int x=s,cnt=0;x!=n+1;x=suf[x]) topo[++cnt]=x;
    |   | return 1;
    | }
    | void clr(int n) {
    |   | rep(i,1,n) dfn[i]=low[i]=p[i]=inv[i]=topo[i]=sgn[i]=0,G[i].clear();
    | }
}
```

1.17 Tree And Graph

1.17.1 树的计数 Prufer序列

树和其prufer编码一一对应, 一颗 n 个点的树, 其prufer编码长度为 $n-2$, 且度数为 d_i 的点在prufer 编码中出现 d_i-1 次.

由树得到序列: 总共需要 $n-2$ 步, 第 i 步在当前的树中寻找具有最小标号的叶子节点, 将与其相连的点的标号设为Prufer序列的第 i 个元素 p_i, 并将此叶子节点从树中删除, 直到最后得到一个长度为 $n-2$ 的Prufer 序列和一个只有两个节点的树.

由序列得到树: 先将所有点的度赋初值为 1, 然后加上它的编号在Prufer序列中出现的次数, 得到每个点的度; 执行 $n-2$ 步, 第 i 步选取具有最小标号的度为 1 的点 u 与 $v=p_i$ 相连, 得到树中的一条边, 并将 u 和 v 的度减一. 最后再把剩下的两个度为 1 的点连边, 加入到树中.

相关结论: n 个点完全图, 每个点度数依次为 $d_1,d_2,\ldots,d_n$, 这样生成树的棵树为: $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\ldots(d_n-1)!}$.

左边有 n_1 个点, 右边有 n_2 个点的完全二分图的生成树棵树为 $n_1^{n_2-1}\times n_2^{n_1-1}$.

m 个连通块, 每个连通块有 c_i 个点, 把他们全部连通的生成树方案数: $(\sum c_i)^{m-2}\prod c_i$

1.17.2 有根树的计数

首先, 令 $S_{n,j}=\sum_{1\le j\le n/j}$; 于是 $n+1$ 个结点的有根树的总数为 $a_{n+1}=\frac{\sum_{j=1}^{n} ja_jS_{n-j}}{n}$. 注: $a_1=1,a_2=1,a_3=2,a_4=4,a_5=9,a_6=20,a_9=286,a_{11}=1842$.

1.17.3 无根树的计数

n 是奇数时, 有 $a_n-\sum_i^{n/2}a_ia_{n-i}$ 种不同的无根树.

n 时偶数时, 有 $a_n-\sum_i^{n/2}a_ia_{n-i}+\frac{1}{2}a_{n/2}(a_{n/2}+1)$ 种不同的无根树.

1.17.4 生成树计数 Kirchhoff's Matrix-Tree Thoerem

Kirchhoff Matrix $T=Deg-A$, Deg 是度数对角阵, A 是邻接矩阵. 无向图度数矩阵是每个点度数; 有向图度数矩阵是每个点入度.

邻接矩阵 $A[u][v]$ 表示 $u\to v$ 边个数, 重边按照边数计算, 自环不计入度数.

无向图生成树计数: $c=|K$ 的任意1个 $n1$ 阶主子式 $|$

有向图外向树计数: $c=|$ 去掉根所在的那阶得到的主子式 $|$

1.17.5 有向图欧拉回路计数 BEST Thoerem

$$ec(G)=t_w(G)\prod_{v\in V}(\deg(v)-1)!$$

其中 deg 为入度 (欧拉图中等于出度), $t_w(G)$ 为以 w 为根的外向树的个数. 相关计算参考生成树计数.

欧拉连通图中任意两点外向树个数相同: $t_v(G)=t_w(G)$.

以 1 结尾的欧拉路径计数就是把 deg 视为出度，把 $\deg(1)$ 的贡献改为 $\deg(1)!$.

1.17.6 Tutte Matrix

Tutte matrix A of a graph $G=(V,E)$:

$$A_{ij}=\begin{cases}x_{ij} & \text{if } (i,j)\in E \text{ and } i<j\\ -x_{ij} & \text{if } (i,j)\in E \text{ and } i>j\\ 0 & \text{otherwise}\end{cases}$$

where x_{ij} are indeterminates. The determinant of this skew-symmetric matrix is then a polynomial (in the variables x_{ij}, $i<j$): this coincides with the square of the pfaffian of the matrix A and is non-zero (as a polynomial) if and only if a perfect matching exists.

1.17.7 Edmonds Matrix

Edmonds matrix A of a balanced ($|U|=|V|$) bipartite graph $G=(U,V,E)$:

$$A_{ij}=\begin{cases}x_{ij} & (u_i,v_j)\in E\\ 0 & (u_i,v_j)\notin E\end{cases}$$

where the x_{ij} are indeterminates. G 有完美匹配当且仅当关于 x_{ij} 的多项式 $det(A_{ij})$ 不恒为 0. 完美匹配的个数等于多项式中单项式的个数.

1.18 拟阵交

```
// max size, minimum weight
namespace MatroidIntersection {
    int K;
    ll W[sz]; // weight
    int in[sz]; // ans
    namespace Check { // implementation needed
```

```
        // recommend writing checker here
        void init() {}
        // return {-1} if no cycle; return cycle otherwise.
        vector<int> cycleA(int x) {}
        vector<int> cycleB(int x) {}
        // not necessary
        void check() {init();}
    }
    bool work() { // try augment
        using pli=pair<ll,int>;
        static vector<int> V[sz];
        static pli dis[sz];
        static int fr[sz];
        Check::init();
        rep(i,1,K) V[i].clear();
        vector<int>A,B;
        rep(i,1,K) if (!in[i]) {
            auto cyca=Check::cycleA(i);
            if (cyca.size()==1u&&cyca[0]==-1) A.push_back(i);
            else for (auto y:cyca) V[y].push_back(i);
            auto cycb=Check::cycleB(i);
            if (cycb.size()==1u&&cycb[0]==-1) B.push_back(i);
            else for (auto y:cycb) V[i].push_back(y);
        }
        rep(i,1,K) dis[i]={ll(1e18),K+1},fr[i]=0;
        priority_queue<pair<pli,int>, vector<pair<pli,int>>,
          ↪greater<pair<pli,int>>>q;
        for (auto x:A) dis[x]={W[x],0},q.push({dis[x],x});
        while (!q.empty()) {
            auto [ww,x]=q.top(); q.pop();
            if (dis[x]!=ww) continue;
            for (auto v:V[x])
                if (chkmin(dis[v],{dis[x].fir+W[v],dis[x].sec+1}))
                    q.push({dis[v],v}),fr[v]=x;
        }
        pli mn={ll(1e18),K+1}; int mnp=-1;
        for (auto x:B) if (chkmin(mn,dis[x])) mnp=x;
        if (mnp==-1) return 0;
        for (int x=mnp;x;x=fr[x]) in[x]^=1;
        Check::check();
        return 1;
    }
    void clr() {
        rep(i,1,K) in[i]=0;
    }
}
```

2 数论

2.1 取模还原分数

```
std::pair<int, int> approx(int p, int q, int A) {
  | int x = q, y = p, a = 1, b = 0;
  | for(;x > A;) {
  |   | std::swap(x, y), std::swap(a, b);
  |   | a -= x / y * b, x %= y;
  | }
  | return {x, a};
} // q ≡ x/a (mod p), x ≤ A, |a| 取到最小值
```

2.2 扩展欧几里得

```
// result : -b < x < b AND -a < y <= a when a,b != 0
void exgcd(ll a, ll b, ll & x, ll & y) {
  | if(!b) return x = 1, y = 0, void();
  | exgcd(b, a % b, y, x), y -= a / b * x;
}
```

2.3 万能欧几里得

```
// 万欧
// 前提 : r < q, r >= q 先提几个 U 出来再用
// 使用：Y * q <= X * p + r, 斜率 p/q, U表示向上，R表示到达一个顶点，先一些 U 再一个 R
template<class T>
T power(T a, ll k) {
  | // 有效率需求可以改为半群乘法
  | if(!k) return T();
  | T res = a;
  | for(--k;k;) {
  |   | if(k & 1) res = res + a;
  |   | if(k >>= 1) a = a + a;
  | }
  | return res;
}
template<class T>
T solve(ll p, ll q, ll r, ll l, T U, T R) {
  | if (p >= q)
  |   | return solve(p % q, q, r, l, U, power(U, p / q) + R);
  | ll m = ((__int128)p * l + r) / q;
  | if (!m) return power(R, l);
  | ll cnt = l - ((__int128)q * m - r - 1) / p;
  | return power(R, (q - r - 1) / p) + U + solve(q, p, (q - r - 1) % p, m - 1,
      ↪R, U) + power(R, cnt);
}
```

2.4 直线下点数|欧几里得

$n < 2^{32}, 1 \le m < 2^{32}$

$$result = \sum_{i=0}^{n-1} \lfloor \frac{ai+b}{m} \rfloor \pmod{2^{63}}$$

```
u64 floor_sum(u64 n, u64 m, u64 a, u64 b) {
  | u64 ans = 0;
  | for(;;) {
  |   | if(a >= m) ans += n * (n - 1) / 2 * (a / m), a %= m;
  |   | if(b >= m) ans += n * (b / m), b %= m;
  |   | u64 ymax = a * n + b; // use u128 if it's big
```

```
    if(ymax < m) break;
    n = ymax / m;
    b = ymax % m;
    std::swap(m, a);
  }
  return ans;
}
```

2.5 Stern-Brocot Tree 二分

```
using cp = std::complex<ll>;
cp fracBS(ll n, ll m, auto f) {
  bool dir = 1, A = 1, B = 1;
  cp lo(0, 1), hi(1, 1); // hi can be (1, 0), f(hi) must be true
  if (f(lo)) return lo;
  while(A || B) {
    ll adv = 0, s = 1;
    for (int x = 0;s;(s *= 2) >>= x) {
      adv += s;
      cp mid = lo * adv + hi;
      if (mid.real() > n || mid.imag() > m || dir == !f(mid)) {
        adv -= s, x = 2;
      }
    }
    hi += lo * adv, dir = !dir;
    swap(lo, hi);
    A = B, B = adv;
  }
  return dir ? hi : lo;
} // 返回值是最小的使得 f 为真的
  // 另外一个是最大的使得 f 为假的
```

2.6 扩展中国剩余定理

```
ll exCRT(ll a1, ll p1, ll a2, ll p2) {
  ll a, b, gcd = std::gcd(p1, p2);
  if((a1 - a2) % gcd)
    return -1;
  exgcd(p1, p2, a, b);
  ll k = i128((a2 - a1) % p2 + p2) * (a + p2) % p2;
  return p1 / gcd * k + a1;
}
```

2.7 Miller-Rabin

```
using f64 = long double;
ll p;
f64 invp;
void setmod(ll x) {
  p = x, invp = (f64) 1 / x;
}
ll mul(ll a, ll b) {
  ll z = a * invp * b + 0.5;
  ll res = a * b - z * p;
  return res + (res >> 63 & p);
}
ll pow(ll a, ll x, ll res = 1) {
  for(;x;x >>= 1, a = mul(a, a))
    if(x & 1) res = mul(res, a);
  return res;
}
bool checkprime(ll p) {
  if(p == 1) return 0;
  setmod(p);
  ll d = __builtin_ctzll(p - 1), s = (p - 1) >> d;
  for(ll a : {2, 3, 5, 7, 11, 13, 82, 373}) {
    if(a % p == 0)
      continue;
    ll x = pow(a, s), y;
    for(int i = 0;i < d;++i, x = y) {
      y = mul(x, x);
      if(y == 1 && x != 1 && x != p - 1)
        return 0;
    }
    if(x != 1) return 0;
  }
  return 1;
}
```

2.8 Pollard-rho

```
ll rho(ll n) {
  if(!(n & 1)) return 2;
  static std::mt19937_64 gen((size_t)"hehezhou");
  ll x = 0, y = 0, prod = 1;
  auto f = [&](ll o) { return mul(o, o) + 1; };
  setmod(n);
  for(int t = 30, z = 0;t % 64 || std::gcd(prod, n) == 1;++t) {
    if (x == y) x = ++ z, y = f(x);
    if(ll q = mul(prod, x + n - y)) prod = q;
    x = f(x), y = f(f(y));
  }
  return std::gcd(prod, n);
}
std::vector<ll> factor(ll x) {
  std::vector<ll> res;
  auto f = [&](auto f, ll x) {
    if(x == 1) return ;
    if(checkprime(x)) return res.push_back(x);
    ll y = rho(x);
    f(f, y), f(f, x / y);
  };
  f(f, x), sort(res.begin(), res.end());
  return res;
}
```

3 Math

3.1 拉格朗日反演

$$G(F(x)) = H(x) \Rightarrow [x^n]G(x) = \frac{1}{n}[u^{n-1}]H'(u)(\frac{u}{F(u)})^n$$

$$G(F(x)) = x \Rightarrow [x^n]H(G(x)) = \frac{1}{n}[u^{n-1}]H'(u)(\frac{u}{F(u)})^n$$

$$G(F(x)) = x \Rightarrow [x^n]G^k(x) = \frac{k}{n}[u^{n-k}](\frac{u}{F(u)})^n$$

3.2 分拆数|五边形数

$$\prod_{i\geq 1}(1-x^i) = \sum_{k=-\infty}^{\infty}(-1)^k x^{\frac{k(3k-1)}{2}}$$

3.3 Fast Fourier Transform

```
using db = double;
using cp = std::complex<db>;
// cp::real, cp::imag, std::conj, std::arg
const db pi = std::acos(-1);
int rev[N], lim;
cp wn[N];
void init(int len) {
  lim = 2 << std::__lg(len - 1);
  for(static int i = 1;i < lim;i += i) {
    for(int j = 0;j < i;++j) {
      wn[i + j] = std::polar(1., db(j) / i * pi);
    }
  }
  for(int i = 1;i < lim;++i) {
    rev[i] = rev[i >> 1] >> 1 | (i % 2u * lim / 2);
  }
}
void DFT(cp * a) {
  for(int i = 0;i < lim;++i) {
    if(rev[i] < i) std::swap(a[rev[i]], a[i]);
  }
  for(int i = 1;i < lim;i += i) {
    for(int j = 0;j < lim;j += i + i) {
      for(int k = 0;k < i;++k) {
        cp x = a[i + j + k] * wn[i + k];
        a[i + j + k] = a[k + j] - x;
        a[k + j] += x;
      }
    }
  }
}
void IDFT(cp * a) {
  DFT(a), std::reverse(a + 1, a + lim);
  for(int i = 0;i < lim;++i)
    a[i] /= lim;
}
```

3.4 Number Theoretic Transform

```
int rev[N], wn[N], lim, invlim;
int pow(int a, int b, int ans = 1) {
  for(;b;b >>= 1, a = (u64) a * a % mod) if(b & 1)
    ans = (u64) ans * a % mod;
  return ans;
}
void init(int len) {
  lim = 2 << std::__lg(len - 1);
  invlim = mod - (mod - 1) / lim;
  for(static int i = 1;i < lim;i += i) {
    wn[i] = 1;
    const int w = pow(3, mod / i / 2);
    for(int j = 1;j < i;++j) {
      wn[i + j] = (u64) wn[i + j - 1] * w % mod;
    }
  }
  for(int i = 1;i < lim;++i) {
    rev[i] = rev[i >> 1] >> 1 | (i % 2u * lim / 2);
  }
}
void DFT(int * a) {
  static u64 t[N];
  for(int i = 0;i < lim;++i) t[i] = a[rev[i]];
  for(int i = 1;i < lim;i += i) {
    for(int k = i & (1 << 19);k--;)
      if(t[k] >= mod * 9ull) t[k] -= mod * 9ull;
    for(int j = 0;j < lim;j += i + i) {
      for(int k = 0;k < i;++k) {
        const u64 x = t[i + j + k] * wn[i + k] % mod;
        t[i + j + k] = t[k + j] + (mod - x), t[k + j] += x;
      }
    }
  }
  for(int i = 0;i < lim;++i) a[i] = t[i] % mod;
}
void IDFT(int * a) {
  DFT(a), std::reverse(a + 1, a + lim);
  for(int i = 0;i < lim;++i)
    a[i] = (u64) a[i] * invlim % mod;
}
```

3.5 Generating function

```
void cpy(int * a, int * b, int n) {
  if(a != b) memcpy(a, b, n << 2);
  memset(a + n, 0, (lim - n) << 2);
}
void inv(int * a, int * b, int n) { // b = inv(a) mod x^n
  if(n == 1) return void(*b = pow(*a, mod - 2));
  static int c[N], d[N];
  int m = (n + 1) / 2;
  inv(a, b, m);
  init(n + m), cpy(c, b, m), cpy(d, a, n);
```

```
  | DFT(c), DFT(d);
  | for(int i = 0;i < lim;++i) c[i] = (u64) c[i] * c[i] % mod * d[i] % mod;
  | IDFT(c);
  | for(int i = m;i < n;++i) b[i] = norm(mod - c[i]);
}
void log(int * a, int * b, int n) { // b = log(a) (mod x^n)
  | static int c[N], d[N];
  | inv(a, c, n), init(n + n);
  | for(int i = 1;i < n;++i) d[i - 1] = (u64) a[i] * i % mod;
  | cpy(d, d, n - 1), cpy(c, c, n);
  | DFT(c), DFT(d);
  | for(int i = 0;i < lim;++i) c[i] = (u64) c[i] * d[i] % mod;
  | IDFT(c), *b = 0;
  | for(int i = 1;i < n;++i) b[i] = pow(i, mod - 2, c[i - 1]);
}
```

3.6 全在线卷积

```
struct oc {
  | std::vector<int> f, g, res;
  | std::vector<std::vector<int>> fa, fb;
  | int n, p;
  | oc(int n) : f(n), g(n), res(n), n(n), p(0) { }
  | void push(int v0, int v1) {
  |   | f[p] = v0;
  |   | res[p] = (res[p] + (u64) f[0] * v1 + (u64) g[0] * v0) % mod;
  |   | g[p++] = v1;
  |   | static int A[N], B[N];
  |   | int lb = p & -p;
  |   | init(lb * 2);
  |   | memset(A, 0, lim << 2);
  |   | memset(B, 0, lim << 2);
  |   | for(int i = 0;i < lb;++i) A[i] = g[p - lb + i], B[i] = f[p - lb + i];
  |   | DFT(A), DFT(B);
  |   | if(lb == p) {
  |   |   | fa.emplace_back(A, A + lim);
  |   |   | fb.emplace_back(B, B + lim);
  |   |   | for(int i = 0;i < lim;++i) A[i] = (u64) A[i] * B[i] % mod;
  |   | } else {
  |   |   | auto & C = fb[std::__lg(lim)], & D = fa[std::__lg(lim)];
  |   |   | for(int i = 0;i < lim;++i) A[i] = ((u64) A[i] * C[i * 2] + (u64)
  |   |   |   ↪ B[i] * D[i * 2]) % mod;
  |   | }
  |   | IDFT(A);
  |   | for(int j = p;j < p + lb && j < n;++j) res[j] = (res[j] + A[j - p +
  |   |   ↪ lb]) % mod;
  | }
};
struct Exp : oc {
  | std::vector<int> res;
  | Exp(int n) : oc(n), res(n) { }
  | void push(int v) {
  |   | if(!res[0]) return void(res[0] = 1);
  |   | oc::push(res[p], v * u64(p + 1) % mod);
  |   | res[p] = (u64) oc::res[p - 1] * inv[p] % mod;
  | }
};
struct Ln : oc {
  | std::vector<int> res; int fi;
  | Ln(int n) : oc(n), res(n), fi(0) {}
  | void push(int v) {
  |   | if(!fi) return void(fi = 1);
  |   | oc::push(res[p] * (u64) p % mod, v);
  |   | res[p] = ((u64) v * p + mod - oc::res[p - 1]) % mod * inv[p] % mod;
  | }
};
struct Inv : oc {
  | std::vector<int> res; int fi;
  | Inv(int n) : oc(n), res(n), fi(0) {}
  | void push(int v) {
  |   | res[p] = fi ? (oc::res[p] + (u64) v * res[0]) % mod * (mod - res[0]) %
  |   |   ↪ mod : pow(fi = v, mod - 2);
  |   | oc::push(res[p], v);
  | }
};
```

3.7 Berlekamp Massey

```
vector<int> berlekamp_massey(const vector<int> &a) {
  | vector<int> v, last; // v is the answer, 0-based
  | int k = -1, delta = 0;
  | for (int i = 0; i < (int)a.size(); i++) {
  |   | int tmp = 0;
  |   | for (int j = 0; j < (int)v.size(); j++)
  |   |   | tmp = (tmp + (long long)a[i - j - 1] * v[j]) % p;
  |   | if (a[i] == tmp) continue;
  |   | if (k < 0) {
  |   |   | k = i; delta = (a[i] - tmp + p) % p;
  |   |   | v = vector<int>(i + 1); continue; }
  |   | vector<int> u = v;
  |   | int val = (long long)(a[i] - tmp + p) *
  |   |   | qpow(delta, p - 2) % p;
  |   | if (v.size() < last.size() + i - k)
  |   |   | v.resize(last.size() + i - k);
  |   | (v[i - k - 1] += val) %= p;
  |   | for (int j = 0; j < (int)last.size(); j++) {
  |   |   | v[i - k + j] = (v[i - k + j] -
  |   |   |   | (long long)val * last[j]) % p;
  |   |   | if (v[i - k + j] < 0) v[i - k + j] += p; }
  |   | if ((int)u.size() - i < (int)last.size() - k) {
  |   |   | last = u; k = i; delta = a[i] - tmp;
  |   |   | if (delta < 0) delta += p; } }
  | for (auto &x : v) x = (p - x) % p;
  | v.insert(v.begin(), 1); //一般是需要最小递推式的，处理一下
  | return v; }
// ∀i, ∑_{j=0}^{m} a_{i-j} v_j = 0
```

3.8 线性规划 | 单纯形法

```
using db = long double;
const db eps = 1e-16;
int sgn(db x) { return x < -eps ? -1 : x > eps; }
namespace LP {
  const int N = 21, M = 21;
  int n, m; // n : 变量个数, m : 约束个数
  db a[M + N][N], x[N + M];
  // 约束: 对于 1 <= i <= m : a[i][0] + \sum_j x[j] * a[i][j] >= 0
  // x[j] >= 0
  // 最大化 \sum_j x[j] * a[0][j]
  int id[N + M];
  void pivot(int p, int o) {
    std::swap(id[p], id[o + n]);
    db w = -a[o][p];
    for(int i = 0;i <= n;++i) a[o][i] /= w;
    a[o][p] = -1 / w;
    for(int i = 0;i <= m;++i) if(sgn(a[i][p]) && i != o) {
      db w = a[i][p]; a[i][p] = 0;
      for(int j = 0;j <= n;++j) a[i][j] += w * a[o][j];
    }
  }
  db solve() { // nan : 无解, inf : 无界, 否则返回最大值
    for(int i = 1;i <= n + m;++i) id[i] = i;
    for(;;) {
      int p = 0, min = 1;
      for(int i = 1;i <= m;++i) {
        if(a[i][0] < a[min][0]) min = i;
      }
      if(a[min][0] >= -eps) break;
      for(int i = 1;i <= n;++i) if(a[min][i] > eps && id[i] > id[p]) {
        p = i;
      }
      if(!p) return nan("");
      pivot(p, min);
    }
    for(;;) {
      int p = 1;
      for(int i = 1;i <= n;++i) if(a[0][i] > a[0][p]) p = i;
      if(a[0][p] < eps) break;
      db min = INFINITY; int o = 0;
      for(int i = 1;i <= m;++i) if(a[i][p] < -eps) {
        db w = -a[i][0] / a[i][p]; int d = sgn(w - min);
        if(d < 0 || !d && id[i] > id[o]) o = i, min = w;
      }
      if(!o) return INFINITY;
      pivot(p, o);
    }
    for(int i = 1;i <= m;++i) x[id[i + n]] = a[i][0];
    return a[0][0];
  }
}
```

3.9 Simpson 积分

$$\int_a^b f(x)dx \approx \frac{b-a}{3n}(f(x_0)+4\sum_{i=1}^{n/2} f(x_{2i-1})+2\sum_{i=1}^{n/2-1} f(x_{2i})+f(x_n))$$

$$\approx \frac{3(b-a)}{8n}(f(x_0)+3\sum_{i=1}^{n/3}(f(x_{3i-1})+f(x_{3i-2}))+2\sum_{i=1}^{n/3-1} f(x_{3i})+f(x_n))$$

3.10 黄金三分

```
db findmax(db a, db c, auto f) {
  auto g = [&](db l, db r) {
    return l + (r - l) * (std::numbers::phi_v<db> - 1);
  };
  db b = g(a, c), bv = f(b);
  for(int i = 0;i < 45;++i) {
    db x = g(a, b), xv = f(x);
    if(xv > bv) { // change here if findmin
      c = b, b = x, bv = xv;
    } else {
      a = c, c = x;
    }
  }
  return bv;
} // log_{1.618}(2) ≈ 1.44
```

4 字符串

4.1 后缀自动机 | SAM

需要两倍点数量。

```
int ch[N][26], lk[N], len[N], nd = 1, las = 1;
void extend(int c, int k) {
  int x = ++ nd, p = las; las = x;
  len[x] = len[p] + 1;
  for(; p && !ch[p][c]; p = lk[p]) ch[p][c] = x;
  if(!p) return lk[x] = 1, void();
  int q = ch[p][c];
  if(len[q] == len[p] + 1)
    return lk[x] = q, void();
  int cl = ++ nd;
  len[cl] = len[p] + 1;
  memcpy(ch[cl], ch[q], 104);
  lk[cl]= lk[q], lk[q] = lk[x] = cl;
  for(; p && ch[p][c] == q; p = lk[p]) ch[p][c] = cl;
} void init() {
  static int bin[N];
  memset(bin, 0, sizeof (int) * (n + 1));
  for(int i = 1; i <= nd; i++) ++ bin[len[i]];
  for(int i = 1; i <= n; i++) bin[i] += bin[i - 1];
  for(int i = nd; i; i--) A[bin[len[i]]--] = i;
}
```

4.2 基本子串字典

```
for(int i = 2; i <= T[0].nd; i++) {
  | int x = T[0].A[i];
  | int R = T[0].r[x], L = R - T[0].len[x] + 1;
  | int y = T[1].fnd(T[1].ed[L], R - L + 1);
  | if(T[1].len[y] == R - L + 1) {
  |   | ++ cnt; T[0].tag[x] = cnt; T[1].tag[y] = cnt;
  |   | rt[0][cnt] = x, rt[1][cnt] = y;
  | }
}
for(int o = 0; o < 2; o++)
for(int i = T[o].nd; i > 1; i--) {
  | int x = T[o].A[i];
  | if(T[o].tag[x]) continue;
  | for(int k = 0; k < 26; k++)
  |   | if(T[o].ch[x][k]) T[o].tag[x] = T[o].tag[T[o].ch[x][k]];
}
for(int o = 0; o < 2; o++)
for(int i = 2; i <= T[o].nd; i++) {
  | int x = T[o].A[i];
  | vec[o][T[o].tag[x]].pb(x);
} // vec[0] : from left to right, node id of the colunm , vec[1] : from down
  ↪ to up
// U : T[0].r[rt] - T[0].len[rt] + 1, D = U + vec[1][t].size() - 1, L = R -
  ↪ vec[0][t].size() + 1, R = T[0].r[rt]
int x = T[0].fnd(T[0].ed[r], r - l + 1);
int blk = T[0].tag[x];
// distance to the right , 0 - base
int rp = T[0].r[rt[0][blk]] - T[0].r[x];
// distance to the up // the upper - right point is (T[0].r[rt] -
  ↪ T[0].len[rt] + 1, T[0].r[rt])
int lp = T[0].r[x] - (r - l) - (T[0].r[rt[0][blk]] - T[0].len[rt[0][blk]] +
  ↪ 1);
```

4.3 DAG 剖分

```
void build() {
  | for(int i = 2; i <= nd; i++)
  |   | e[lk[i]].pb(i);
  | static int q[N], d[N];
  | for(int i = 1; i <= nd; i++)
  |   | for(int j = 0; j < 26; j++)
  |   |   | if(ch[i][j]) ++ d[ch[i][j]];
  | int hd = 1, tl = 0;
  | q[++ tl] = 1;
  | while(hd <= tl) {
  |   | int x = q[hd ++];
  |   | for(int i = 0; i < 26; i++)
  |   |   | if(ch[x][i]) {
  |   |   |   | int v = ch[x][i];
  |   |   |   | if((-- d[v]) == 0) q[++ tl] = v;
  |   |   | }
  | }
  | static ll f[N], h[N];
  | for(int i = tl, x; i; i--) {
  |   | f[x = q[i]] ++;
  |   | for(int j = 0; j < 26; j++)
  |   |   | if(ch[x][j]) f[x] += f[ch[x][j]];
  | }
  | for(int i = 1, x; i <= tl; i++) {
  |   | h[x = q[i]] ++;
  |   | for(int j = 0; j < 26; j++)
  |   |   | if(ch[x][j]) h[ch[x][j]] += h[x];
  | }
  | static int nx[N], fr[N];
  | for(int i = 1; i <= nd; i++) {
  |   | for(int j = 0; j < 26; j++)
  |   |   | if(ch[i][j] && f[ch[i][j]] > f[nx[i]]) nx[i] = ch[i][j];
  |   | for(int j = 0; j < 26; j++)
  |   |   | if(ch[i][j] && h[i] > h[fr[ch[i][j]]]) fr[ch[i][j]] = i;
  | }
  | fr[0] = nx[0] = 0;
  | static bool vis[N];
  | for(int i = 1; i <= nd; i++) {
  |   | if(fr[nx[i]] == i) son[i] = nx[i], vis[son[i]] = 1;
  | }
}
```

4.4 exKMP

```
static int lcp[N];
int mx=1, pt=1; lcp[1]=n;
for(int i=2; i<=n; i++){
  | if(i<=mx) lcp[i]=min(lcp[i-pt+1],mx-i+1);
  | while(i+lcp[i]<=n && S[i+lcp[i]]==S[1+lcp[i]]) ++lcp[i];
  | if(i+lcp[i]-1>mx) pt=i, mx=i+lcp[i]-1;
}
```

4.5 log 个最小后缀

```
for(int i = 1; i <= n; i ++) {
  | St.pb(i); vector<int> nw;
  | for(auto t : St) {
  |   | bool ok = true;
  |   | while(!nw.empty()){
  |   |   | int x = nw.back();
  |   |   | if(S[i] > S[i-t+x]) ok = false;
  |   |   | if(S[i] >= S[i-t+x]) break; nw.pop_back();
  |   | }
  |   | if(ok && (nw.empty() || (i - t + 1 <= t - nw.back()))) nw.pb(t);
  | } St = nw;
}
for(int x : St){
  | bool FLAG = true;
  | while(nx.size()){
  |   | int y = nx.back(); int lcp = LCP(x, y); if(x + lcp - 1 >= r) break;
  |   | if(S[x + lcp] > S[y + lcp]){ FLAG = false; break; } nx.pop_back();
  | } if(FLAG && (nx.empty() || r - x + 1 <= x - nx.back())) nx.pb(x);
} // in segmentree, work(L, ans, rpos), work(R, ans, rpos), then return ans
```

4.6 SA

```
char s[N]; int m, rk[N * 2], sa[N], tmp[N * 2], h[N], y[N];
void Sort(){
  for(int i=1;i<=m;i++) c[i] = 0;
  for(int i=1;i<=n;i++) c[rk[i]]++;
  for(int i=1;i<=m;i++) c[i] += c[i-1];
  for(int i=n;i>=1;i--) sa[c[rk[y[i]]]--] = y[i];
}
void get_sa(){
  for(int i=1;i<=n;i++) rk[i] = s[i], y[i] = i; Sort();
  for(int k=1;k<=n;k<<=1){
    int ret = 0;
    for(int i=n-k+1;i<=n;i++) y[++ret] = i;
    for(int i=1;i<=n;i++) if(sa[i] > k) y[++ret] = sa[i] - k;
    Sort();
    for(int i = 1; i <= n; i++) swap(rk[i], tmp[i]);
    rk[sa[1]] = 1; int num = 1;
    for(int i=2;i<=n;i++){
      if(tmp[sa[i]] == tmp[sa[i-1]] && tmp[sa[i]+k] == tmp[sa[i-1]+k])
        rk[sa[i]] = num;
      else rk[sa[i]] = ++num;
    } m = num;
  }
}
void get_h(){
  int k = 0;
  for(int i=1;i<=n;i++){
    if(rk[i]==1) continue;
    int j = sa[rk[i]-1]; if(k) k--;
    while(i+k<=n && j+k<=n && s[i+k]==s[j+k]) k++;
    h[rk[i]] = k;
  }
}
```

4.7 PAM

```
namespace pam {
  int ch[N][26], len[N], lk[N], rp, las, nd, top[N], d[N];
  void init() { rp = 0, las = nd = 1, len[1] = -1, lk[0] = 1; }
  // remember to set S[0] = *
  int jmp(int x) { while(S[rp - len[x] - 1] != S[rp]) x = lk[x]; return x; }
  void ins(int c) {
    ++ rp; int p = jmp(las);
    if(!ch[p][c]) {
      int x = ++ nd;
      len[x] = len[p] + 2;
      lk[x] = ch[jmp(lk[p])][c];
      ch[p][c] = x;
      if(len[x] - len[lk[x]] == d[lk[x]])
        top[x] = top[lk[x]], d[x] = d[lk[x]];
      else {
        top[x] = x;
        d[x] = len[x] - len[lk[x]];
      }
    } las = ch[p][c];
  }
}
while(x) {
  if(pam :: d[x] == pam :: d[pam :: lk[x]]) {
    // when doing dp, the position i - len[x] ~
    // i - (len[top[x]] + d[x]) have been updated (in i - d[x])
    g[x] = f[i - pam :: len[pam :: top[x]]];
      Add(g[x], g[pam :: lk[x]]);
  }
  else {
    // update from i - len[x]
    g[x] = f[i - pam :: len[x]];
  }
  Add(f[i], g[x]);
  x = pam :: lk[pam :: top[x]];
}
```

4.8 AC 自动机

```
void init() {
  queue <int> q;
  for(int i = 0; i < 26; i++)
    if(ch[0][i]) q.push(ch[0][i]);
  while(!q.empty()) {
    int x = q.front(); q.pop();
    e[lk[x]].pb(x);
    for(int i = 0; i < 26; i++) {
      if(ch[x][i]) {
        lk[ch[x][i]] = ch[lk[x]][i];
        q.push(ch[x][i]);
      }
      else ch[x][i] = ch[lk[x]][i];
    }
  }
}
```

4.9 Manacher

```
S[1] = '%';
for(int i = 1; i <= len; i++){
  S[i << 1] = '&';
  S[i << 1|1] = s[i];
}
len = len << 1 | 1;
S[++len] = '&';
S[++len] = '$';
int mx = 0, id = 0, ans = 0;
for(int i = 1; i <= len; i++){
  if(mx > i) p[i] = min(p[id * 2 - i], mx - i);
  else p[i] = 1;
  while(S[i - p[i]] == S[i + p[i]]) ++p[i];
  if(i + p[i] > mx) id = i, mx = i + p[i];
  ans = max(ans, p[i] - 1);
}
```

4.10 Lyndon/最小表示法

```
vector <int> duval(vector <int> S) {
  | int i = 0, j, k, s = S.size(); vector <int> ans;
  | while(i < s) {
  |   | j = i, k = i + 1;
  |   | while(j < s && k < s && S[j] <= S[k]) {
  |   |   | if(S[j] == S[k]) ++ j;
  |   |   | else j = i; ++ k;
  |   | } while (i <= j) { ans.pb(i + k - j - 1); i += k - j; }
  | } return ans;  // [ans[i] + 1, ans[i + 1]] is a lyndon word
}
vector <int> min_rep(vector <int> S) {
  | int k = 0, i = 0, j = 1, n = S.size();
  | while (k < n && i < n && j < n) {
  |   | if (S[(i + k) % n] == S[(j + k) % n]) k ++;
  |   | else {
  |   |   | S[(i + k) % n] > S[(j + k) % n] ? i = i + k + 1 : j = j + k + 1;
  |   |   | if (i == j) i ++; k = 0;
  |   | }
  | } i = min(i, j);
  | rotate(S.begin(), S.begin() + i, S.end()); return S;
}
```

4.11 Runs

```
// need lcp and lcs
bool cmp(int x, int y){
  | int l = lcp(x, y);
  | if(x + l > n) return true;
  | if(y + l > n) return false;
  | return S[x + l] < S[y + l];
}
set <pi> ex;
void ins(int l, int r) {
  | int p = r - l;
  | int l1 = lcp(l, r);
  | int l2 = lcs(l - 1, r - 1);
  | int L = l - l2, R = r + l1 - 1;
  | if(R - L + 1 >= 2 * p) {
  |   | auto iter = ex.lower_bound(pi(L, R));
  |   | if(iter != ex.end() && *iter == pi(L, R)) return ;
  |   | ex.emplace_hint(iter, pi(L, R));
  |   | runs.pb((run){L, R, p});
  | }
}
void Run(int o){
  | static int s[N];
  | int top = 0; s[++top] = n + 1;
  | for(int i = n; i; i--){
  |   | while(top > 1 && cmp(i, s[top]) == o) --top;
  |   | ins(i, s[top]), s[++top] = i;
  | }
}
```

5 数据结构

5.1 区间加区间求和树状数组

```
// 后缀加，前缀求和
struct BIT {
  | ll a[N], b[N];
  | void add(ll p, int v) {
  |   | for(int i = p;i < N;i += i & -i)
  |   |   | a[i] += v, b[i] += p * v;
  | }
  | ll qry(ll p) {
  |   | ll res = 0;
  |   | for(int i = p;i;i &= i - 1) res += (p + 1) * a[i] - b[i];
  |   | return res;
  | }
  | void add(int l, int r, int v) { add(l, v), add(r + 1, -v); }
  | ll qry(int l, int r) { return qry(r) - qry(l - 1); }
} bit;
```

5.2 zkw 线段树

```
struct seg {
  | ll o[1 << 20]; int L;
  | void upt(int x) {
  |   | o[x] = o[x << 1] + o[x << 1 | 1];
  | }
  | void init(int n, int * w) {
  |   | L = 2 << std::__lg(n + 1);
  |   | for(int i = 1;i <= n;++i) o[i + L] = w[i];
  |   | for(int i = L;i >= 1;--i) upt(i);
  | }
  | void upt(int p, int v) {
  |   | for(o[p += L] += v;p >>= 1;upt(p));
  | }
  | ll qry(int l, int r) {
  |   | l += L - 1, r += L + 1;
  |   | ll ans = 0;
  |   | for(;l ^ r ^ 1;l >>= 1, r >>= 1) {
  |   |   | if((l & 1) == 0) ans += o[l ^ 1];
  |   |   | if((r & 1) == 1) ans += o[r ^ 1];
  |   | }
  |   | return ans;
  | }
  | // if there is no I
  | ll qry2(int l, int r) {
  |   | if(l == r) return o[l + L];
  |   | ll le = o[l + L], ri = o[r + L];
  |   | l += L, r += L;
  |   | for(;l ^ r ^ 1;l >>= 1, r >>= 1) {
  |   |   | if((l & 1) == 0) le = le + o[l ^ 1];
  |   |   | if((r & 1) == 1) ri = o[r ^ 1] + ri;
  |   | }
  |   | return le + ri;
  | }
```

```
} sgt;
```

5.3 Link Cut Tree

```
int son[N][2], fa[N], rev[N];
int get(int x, int p = 1) { return son[fa[x]][p] == x; }
void update(int x) { }
int is_root(int x) { return !(get(x) || get(x, 0)); }
void rotate(int x) {
 | int y = fa[x], z = fa[y], b = get(x);
 | if(!is_root(y)) son[z][get(y)] = x;
 | son[y][b] = son[x][!b], son[x][!b] = y;
 | fa[son[y][b]] = y, fa[y] = x, fa[x] = z;
 | update(y);
}
void put(int x) {
 | if(x) rev[x] ^= 1, std::swap(son[x][0], son[x][1]);
}
void down(int x) {
 | if(rev[x]) {
 |  | put(son[x][0]);
 |  | put(son[x][1]);
 |  | rev[x] = 0;
 | }
}
void pushdown(int x) {
 | if(!is_root(x)) pushdown(fa[x]);
 | down(x);
}
void splay(int x) {
 | for(pushdown(x);!is_root(x);rotate(x)) if(!is_root(fa[x]))
 |  | rotate(get(x) ^ get(fa[x]) ? x : fa[x]);
 | update(x);
}
void access(int x) {
 | for(int t = 0;x;son[x][1] = t, t = x, x= fa[x])
 |  | splay(x);
}
void makeroot(int x) {
 | access(x), splay(x), put(x);
}
```

5.4 FHQ Treap

```
int root,cc;
struct hh{int w,pri,ch[2],size;}tr[sz];
#define ls(x) tr[x].ch[0]
#define rs(x) tr[x].ch[1]
void pushup(int x){tr[x].size=1+tr[ls(x)].size+tr[rs(x)].size;}
int newnode(int w) {
 | ++cc;
 | tr[cc].w=w,tr[cc].pri=rnd(1,int(1e9)),tr[cc].size=1;
 | return cc;
}
int merge(int x,int y) {
 | if (!x||!y) return x+y;
 | if (tr[x].pri<tr[y].pri) return rs(x)=merge(rs(x),y),pushup(x),x;
 | return ls(y)=merge(x,ls(y)),pushup(y),y;
}
void split(int x,int w,int &a,int &b) {
 | if (!x) return a=b=0,void();
 | if (tr[x].w<=w) a=x,split(rs(x),w,rs(x),b);
 | else b=x,split(ls(x),w,a,ls(x));
 | pushup(x);
}
```

5.5 pbds tree

```
#include <bits/extc++.h>
using namespace __gnu_pbds;
template<class T> // insert, erase, join, order_of_key, find_by_order(return
  ↪ iterator), order is 0-index
using Tree = tree<T, null_type, std::less<T>, rb_tree_tag,
  ↪ tree_order_statistics_node_update>;
```

6 geometry

6.1 向量

```
using db = long double;
const db eps = 1e-10;
db sgn(db x) { return x < -eps ? -1 : x > eps; }
db eq(db x, db y) { return !sgn(x - y); }
struct p2 {
 | db x, y;
 | db norm() const { return x * x + y * y; }
 | db abs() const { return std::sqrt(x * x + y * y); }
 | db arg() const { return atan2(y, x); }
};
db arg(p2 x, p2 y) {
 | db a = y.arg() - x.arg();
 | if(a > pi) a -= pi * 2;
 | if(a < -pi) a += pi * 2;
 | return a;
}
p2 r90(p2 x) { return {-x.y, x.x}; }
p2 operator + (p2 x, p2 y) { return {x.x + y.x, x.y + y.y}; }
p2 operator - (p2 x, p2 y) { return {x.x - y.x, x.y - y.y}; }
p2 operator / (p2 x, db y) { return {x.x / y, x.y / y}; }
p2 operator * (p2 x, db y) { return {x.x * y, x.y * y}; }
p2 operator * (db y, p2 x) { return {x.x * y, x.y * y}; }
db operator * (p2 x, p2 y) { return x.x * y.y - x.y * y.x; }
db operator % (p2 x, p2 y) { return x.x * y.x + x.y * y.y; }
int half(p2 x){return x.y < 0 || (x.y == 0 && x.x <= 0); }
int half(p2 x){return x.y < -eps || (std::fabs(x.y) < eps && x.x < eps);}
bool cmp(p2 a, p2 b) { return half(a) == half(b) ? a * b > 0 : half(b); }
bool cmp_eq(p2 A, p2 B) { return half(A) == half(B) && eq(A * B, 0) ; }
// 判断 A, B, C 三个向量是否是逆时针顺序
// 如果是，返回 1
// 如果 (A, B), (C, B) 同方向共线，返回 -1
```

```
// 如果是顺时针，返回 0
bool cmp_ct(p2 A, p2 B, p2 C) {
  if(cmp_eq(A, B)) return -1;
  if(cmp_eq(C, B)) return -1;
  if(cmp(A, B)) {
    return cmp(B, C) || cmp(C, A);
  } else {
    return cmp(B, C) && cmp(C, A);
  }
}
// 凸包 DP
struct pr { int i, j; p2 get() const { return a[j] - a[i]; } };
bool cmpseg(pr x, pr y) {
  p2 A = x.get(), B = y.get();
  if(!cmp(A, B) && !cmp(B, A)) return a[x.i] % A < a[y.i] % A;
  return cmp(A, B);
}
```

6.2 直线半平面

```
struct line : p2 {
  db z;
  // a * x + b * y + c (= or >) 0
  line() = default;
  line(db a, db b, db c) : p2{a, b}, z(c) {}
  line(p2 a, p2 b) : p2(r90(b - a)), z(a * b) { } //左侧 > 0
  db operator ()(p2 a) const { return a % p2(*this) + z; }
  line perp() const { return {y, -x, 0}; } // 垂直
  line para(p2 o) { return {x, y, z - (*this)(o)}; } // 平行
};
p2 operator & (line x, line y) {
  return p2{p2{x.z, x.y} * p2{y.z, y.y}, p2{x.x, x.z} * p2{y.x, y.z}} /
    ↪-(p2(x) * p2(y));
  // 注意此处精度误差较大，以及 res.y 需要较高精度
}
p2 proj(p2 x, line l){return x - p2(l) * (l(x) / l.norm());}//投影
p2 refl(p2 x, line l){return x - p2(l) * (l(x) / l.norm()) * 2;}//对称
db dist(line l, p2 x={0, 0}){return l(x) / l.abs();}//有向点到线距离
bool is_para(line x, line y){return eq(p2(x) * p2(y), 0);}//判断线平行
bool is_perp(line x, line y){return eq(p2(x) % p2(y), 0);}//判断线垂直
bool online(p2 x, line l) { return eq(l(x), 0); } // 判断点在线上
int ccw(p2 a, p2 b, p2 c) {
  int sign = sgn((b - a) * (c - a));
  if(sign == 0) {
    if(sgn((b - a) % (c - a)) == -1) return 2;
    if((c - a).norm() > (b - a).norm() + eps) return -2;
  }
  return sign;
}
db det(line a, line b, line c) {
  p2 A = a, B = b, C = c;
  return c.z * (A * B) + a.z * (B * C) + b.z * (C * A);
}
db check(line a, line b, line c) { // sgn same as c(a & b), 0 if error
  return sgn(det(a, b, c)) * sgn(p2(a) * p2(b));
}
bool paraS(line a, line b) { // 射线同向
  return is_para(a, b) && p2(a) % p2(b) > 0;
}
```

6.3 半平面交

```
std::vector<p2> HPI(std::vector<line> vs) {
  auto cmp = [](line a, line b) {
    if(paraS(a, b)) return dist(a) < dist(b);
    return ::cmp(p2(a), p2(b));
  };
  sort(vs.begin(), vs.end(), cmp);
  int ah = 0, at = 0, n = size(vs);
  std::vector<line> deq(n + 1);
  std::vector<p2> ans(n);
  deq[0] = vs[0];
  for(int i = 1;i <= n;++i) {
    line o = i < n ? vs[i] : deq[ah];
    if(paraS(vs[i - 1], o)) continue;
    for(;ah < at && check(deq[at - 1], deq[at], o) < 0;) -- at;//maybe <=
    if(i != n) for(;ah < at && check(deq[ah], deq[ah + 1], o) < 0;) ++ ah;
    if(!is_para(o, deq[at])) {
      ans[at] = o & deq[at];
      deq[++at] = o;
    }
  }
  if(at - ah <= 2) return {};
  return {ans.begin() + ah, ans.begin() + at};
}
```

6.4 线段

```
struct seg {
  p2 x, y;
  seg() {}
  seg(const p2 & A, const p2 & B) : x(A), y(B) {}
  bool onseg(const p2 & o) const {
    return (o - x) % (o - y) < eps && std::fabs((o - x) * (o - y)) < eps;
  }
};
db dist(const seg & o, const p2 & x) {
  if((o.x - o.y) % (x - o.y) <= eps) return (x - o.y).abs();
  if((o.y - o.x) % (x - o.x) <= eps) return (x - o.x).abs();
  return fabs((o.x - x) * (o.y - x) / (o.x - o.y).abs());
}
bool is_isc(const seg & x, const seg & y) {
  return
    ccw(x.x, x.y, y.x) * ccw(x.x, x.y, y.y) <= 0 &&
    ccw(y.x, y.y, x.x) * ccw(y.x, y.y, x.y) <= 0;
}
db dist(const seg & x, const seg & y) {
  if(is_isc(x, y)) return 0;
  return std::min({dist(y, x.x), dist(y, x.y), dist(x, y.x), dist(x, y.y)});
```

```
}
```

6.5 多边形

```
using polygon = std::vector<p2>;
// counter-clockwise
db area(const polygon & x) {
  db res = 0;
  for(int i = 2;i < (int) x.size();++i) {
    res += (x[i - 1] - x[0]) * (x[i] - x[0]);
  }
  return res / 2;
}
bool is_convex(const polygon & x, bool strict = 1) {
  // warning, maybe wrong
  const db z = strict ? eps : -eps;
  for(int i = 2;i < (int) x.size() + 2;++i) {
    if((x[(i - 1) % x.size()] - x[i - 2]) * (x[i % x.size()] - x[i - 2]) < z) return 0;
  }
  return 1;
}
int contain(const std::vector<p2> & a, p2 o) { // 简单多边形包含判定
  bool in = 0;
  for(int i = 0;i < (int) a.size();++i) {
    p2 x = a[i] - o, y = a[(i + 1) % a.size()] - o;
    if(x.y > y.y) std::swap(x, y);
    if(x.y <= eps && y.y > eps && x * y < -eps) in ^= 1;
    if(std::fabs(x * y) < eps && x % y < eps) return 2; // 在线段上，看情况改
  }
  return in;
}
```

6.6 线段 in 多边形

```
bool contains(p2 x, p2 y, const std::vector<p2> & a) {
  using pr = std::pair<double, int>;
  std::vector<pr> e = {pr(-inf, 0), pr(inf, 0)};
  p2 t = y - x;
  auto f = [&](p2 a, p2 b, p2 c, p2 d) {
    return (b - a).abs() * ((c - a) * (d - c)) / ((b - a) * (d - c));
  };
  for(int i = 0;i < (int) a.size();++i) {
    p2 u = a[i], v = a[(i + 1) % a.size()];
    int a = sgn(t * (u - x));
    int b = sgn(t * (v - x));
    if(a != b) e.emplace_back(f(x, y, u, v), b - a);
  }
  sort(e.begin(), e.end());
  int sum = 0; db R = t.abs();
  for(int i = 0;i + 1 < (int) e.size();++i) {
    sum += e[i].second;
    if(sum == 0 && std::max(e[i].first, 0.) + eps < std::min(e[i + 1].first, R)) {
      return 0;
    }
  }
  return 1;
}
```

6.7 图形求交

```
struct circle : p2 { db r; };
std::vector<p2> operator & (circle o, line l) {
  p2 v = l, Rv = r90(v); db L = l.abs();
  db d = l(p2(o)) / L, x = o.r * o.r - d * d;
  if(x < -eps) return {};
  x = std::sqrt(x * sgn(x));
  p2 z = p2(o) - v * (d / L), p = Rv * (x / L);
  return {z + p, z - p};
} // l 如果是构造函数给出，那么返回交点按射线顺序
std::vector<p2> operator & (circle o, seg s) {
  std::vector<p2> b;
  for(p2 x : (o & s.to_l()))
    if(s.onseg(x)) b.push_back(x);
  return b;
}
std::vector<p2> operator & (circle o0, circle o1) {
  p2 tmp = (p2(o1) - p2(o0)) * 2.;
  return o0 & line(tmp.x, tmp.y, o1.r * o1.r - o0.r * o0.r + o0.norm() - o1.norm());
}
std::vector<p2> tang(circle o, p2 x) {
  db d = (x - p2(o)).abs();
  if(d <= o.r + eps) return {};
  return o & circle{x, sqrt(d * d - o.r * o.r)};
}
// 三角形 (0, a, b) 和圆 o 的交的有向面积 * 2
db intersect(circle o, p2 a, p2 b) {
  a = a - p2(o), b = b - p2(o); o.x = o.y = 0;
  int va = a.abs() <= o.r + eps;
  int vb = b.abs() <= o.r + eps;
  if(va && vb) return a * b;
  auto v = o & seg{a, b}; // 注意这里，有必要改一下 onseg，去掉平行判定
  if(v.empty()) return arg(a, b) * o.r * o.r;
  db sum = 0;
  sum += va ? a * v[0] : arg(a, v[0]) * o.r * o.r;
  sum += vb ? v.back() * b : arg(v.back(), b) * o.r * o.r;
  if(v.size() > 1) sum += v[0] * v[1];
  return sum;
}
// 有向弓形面积 * 2, arg 不能改
db csegS(circle o, p2 a, p2 b) {
  a = a - p2(o);
  b = b - p2(o);
  db d = b.arg() - a.arg();
  if(d < 0) d += pi * 2;
  return d * o.r * o.r - a * b;
}
```

```cpp
// 两圆交的面积 * 2
db intersect(circle o0, circle o1) {
  if(o0.r > o1.r) std::swap(o0, o1);
  db d = (p2(o0) - p2(o1)).abs();
  if(d <= (o1.r - o0.r) + eps) return 2 * pi * o0.r * o0.r;
  if(d >= o1.r + o0.r - eps) return 0;
  auto v = o0 & o1;
  return csegS(o0, v[1], v[0]) + csegS(o1, v[0], v[1]);
}
```

6.8 凸包

结果为逆时针。

```cpp
db cross(p2 x, p2 y, p2 z) { return (y.x - x.x) * (z.y - x.y) - (y.y - x.y) * (z.x - x.x); }
std::vector<p2> gethull(std::vector<p2> o) {
  rgs::sort(o, [](p2 x, p2 y) { return eq(x.x, y.x) ? x.y < y.y : x.x < y.x; });
  o.erase(unique(o.begin(), o.end(), [](p2 x, p2 y) {
    return eq(x.x, y.x) && eq(x.y, y.y);
  }), o.end());
  std::vector<p2> s;
  for(int i = 0;i < (int) o.size();++i) {
    for(;s.size() >= 2 && cross(s.rbegin()[1], s.back(), o[i]) <= eps;)
      s.pop_back();
    s.push_back(o[i]);
  }
  for(int i = o.size() - 2, t = s.size();i >= 0;--i) {
    for(;s.size() > t && cross(s.rbegin()[1], s.back(), o[i]) <= eps;)
      s.pop_back();
    s.push_back(o[i]);
  }
  if(s.size() > 1) s.pop_back();
  return s;
} // 把两个 eps 改成 -eps 可求出所有在凸包上的点
int findmin(std::vector<p2> & a, auto cmp) {
  int l = 0, r = a.size() - 1, d = 1;
  if(cmp(a.back(), a[0])) std::swap(l, r), d = -1;
  for(;(r - l) * d > 1;) {
    int mid = (l + r) >> 1;
    if(cmp(a[mid], a[mid - d]) && cmp(a[mid], a[l])) {
      l = mid;
    } else {
      r = mid;
    }
  }
  return l;
} // cmp is less, and a.size()>0 plz
int contains(std::vector<p2> & a, p2 x) {
  auto it = lower_bound(a.begin() + 2, a.end(), x, [&](p2 x, p2 y) {
    return cross(a[0], x, y) > 0;
  });
  ll c0 = cross(it[-1], *it, x), c1 = cross(a[0], a[1], x);
  if(it != a.end() && c0 >= 0 && c1 >= 0) {
    return c0 > 0 && c1 > 0 && cross(a.back(), a[0], x) > 0 ? IN : ON;
  } else {
    return 0;
  }
} // a.size()>2 plz
```

6.9 上凸壳

结果显然为顺时针。

```cpp
std::vector<p2> gethull(std::vector<p2> o) {
  sort(o.begin(), o.end(), [](p2 x, p2 y) {
    if(x.x == y.x) {
      return x.y > y.y; // gt ⇒ lt
    } else {
      return x.x < y.x;
    }
  });
  std::vector<p2> stack;
  for(p2 x : o) {
    if(stack.size() && stack.back().x == x.x) {
      continue;
    }
    for(;stack.size() >= 2 && cross(stack.rbegin()[1], stack.back(), x) >= 0;) { // gt ⇒ lt
      stack.pop_back();
    }
    stack.push_back(x);
  }
  return stack;
}
```

6.10 最小圆覆盖

```cpp
struct circle : p2 { db r; };
circle incircle(p2 a, p2 b, p2 c) {
  db A = (b - c).abs(), B = (c - a).abs(), C = (a - b).abs();
  return {(a * A + b * B + c * C) / (A + B + C), fabs((b - a) * (c - a)) / (A + B + C)};
} // 三点确定内心，不是最小圆覆盖内容
circle circumcenter(p2 a, p2 b, p2 c) {
  p2 bc = c - b, ca = a - c, ab = b - a;
  p2 o = (b + c - r90(bc) * (ca % ab) / (ca * ab)) / 2;
  return {o, (a - o).abs()};
}// 三点确定外心
circle cir(p2 a, p2 b) { // 根据直径生成圆
  return {(a + b) / 2, (a - b).abs() / 2};
}
bool in(circle x, p2 y) { return (p2(x) - y).abs() <= x.r + eps; }
circle mincircle(std::vector<p2> a) { // 最小圆覆盖，需要 shuffle
  circle o = {a[0], 0};
  int n = a.size();
  for(int i = 1;i < n;++i) {
    if(in(o, a[i])) continue;
    o = cir(a[0], a[i]);
```

```
    | | for(int j = 1;j < i;++j) {
    | | | if(in(o, a[j])) continue;
    | | | o = cir(a[j], a[i]);
    | | | for(int k = 0;k < j;++k) {
    | | | | if(in(o, a[k])) continue;
    | | | | o = circumcenter(a[i], a[j], a[k]);
    | | | }
    | | }
    | }
    | return o;
}
```

6.11 最近点对

```
db mindist(std::vector<p2> a) {
  | db ans = 1e18;
  | sort(a.begin(), a.end(), [](p2 x, p2 y) { return x.x < y.x; });
  | ans = (a[0] - a[1]).abs();
  | auto solve = [&](auto s, int l, int r) {
  | | if(l + 1 == r) return ;
  | | int mid = (l + r) >> 1;
  | | db mx = a[mid].x;
  | | s(s, l, mid), s(s, mid, r);
  | | static std::vector<p2> b; b.clear();
  | | inplace_merge(a.begin() + l, a.begin() + mid, a.begin() + r, [](p2 x,
  |   ↪ p2 y) { return x.y < y.y; });
  | | for(int i = l;i < r;++i)
  | | | if(fabs(a[i].x - mx) <= ans) b.push_back(a[i]);
  | | for(int i = 1;i < (int) b.size();++i)
  | | | for(int j = i - 1;j >= 0 && b[i].y <= b[j].y + ans;--j) ans =
  |   ↪ std::min(ans, (b[i] - b[j]).abs());
  | };
  | solve(solve, 0, a.size());
  | return ans;
}
```

6.12 凸包直径

```
db convex_diameter(std::vector<p2> & o) {
  | int n = size(o);
  | db max = 0;
  | for(int i = 0, j = 0;i < n;++i) {
  | | for(;j + 1 < n && (o[j] - o[i]).abs() < (o[j + 1] - o[i]).abs();) ++ j;
  | | max = std::max(max, (o[j] - o[i]).abs());
  | }
  | return max;
} // 凸包直径
```

6.13 切凸包

```
std::vector<p2> cut(const std::vector<p2> & o, line l) {
  | std::vector<p2> res;
  | int n = size(o);
  | for(int i = 0;i < n;++i) {
  | | p2 a = o[i], b = o[(i + 1) % n];
  | | if(sgn(l(a)) >= 0) res.push_back(a); // 注意 sgn 精度
  | | if(sgn(l(a)) * sgn(l(b)) < 0) res.push_back(line(a, b) & l);
  | }
  | if(res.size() <= 2) return {};
  | return res;
} // 切凸包
```

6.14 V 图

```
std::vector<line> cut(const std::vector<line> & o, line l) {
  | std::vector<line> res;
  | int n = size(o);
  | for(int i = 0;i < n;++i) {
  | | line a = o[i], b = o[(i + 1) % n], c = o[(i + 2) % n];
  | | int va = check(a, b, l), vb = check(b, c, l);
  | | if(va > 0 || vb > 0 || (va == 0 && vb == 0)) {
  | | | res.push_back(b);
  | | }
  | | if(va >= 0 && vb < 0) {
  | | | res.push_back(l);
  | | }
  | }
  | if(res.size() <= 2) return {};
  | return res;
} // 切凸包
line bisector(p2 a, p2 b) { return line(a.x - b.x, a.y - b.y, (b.norm() -
  ↪ a.norm()) / 2); }
std::vector<std::vector<line>> voronoi(std::vector<p2> p) {
  | int n = p.size();
  | auto b = p; shuffle(b.begin(), b.end(), gen);
  | const db V = 1e5; // 边框大小，重要
  | std::vector<std::vector<line>> a(n, {
  | | {V, 0, V * V}, {0, V, V * V},
  | | {-V, 0, V * V}, {0, -V, V * V},
  | });
  | for(int i = 0;i < n;++i) {
  | | for(p2 x : b) if((x - p[i]).abs() > eps) {
  | | | a[i] = cut(a[i], bisector(p[i], x));
  | | }
  | }
  | return a;
}
```

6.15 Delaunay 三角剖分

```
using i128 = __int128;
using Q = struct Quad*;
p2 arb(LLONG_MAX, LLONG_MAX);
struct Quad {
  | Q rot, o; p2 p = arb; bool mark;
  | p2& F() { return r() -> p; }
  | Q& r() { return rot->rot; }
  | Q prev() { return rot->o->rot; }
  | Q next() { return r()->prev(); }
} *H;
ll cross(p2 a, p2 b, p2 c) {
```

```
  | return (b - a) * (c - a);
}
bool circ(p2 p, p2 a, p2 b, p2 c) { // p 是否在 a, b, c 外接圆中
  | i128 p2 = p.norm(), A = a.norm() - p2, B = b.norm() - p2, C = c.norm() -
    ↪ p2;
  | a = a - p, b = b - p, c = c - p;
  | return (a * b) * C + (b * c) * A + (c * a) * B > 0;
}
Q link(p2 orig, p2 dest) {
  | Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
  | H = r -> o; r -> r() -> r() = r;
  | for(int i = 0;i < 4;++i)
  |   | r = r -> rot, r -> p = arb, r -> o = i & 1 ? r : r -> r();
  | r -> p = orig, r -> F() = dest;
  | return r;
}
void splice(Q a, Q b) {
  | std::swap(a -> o -> rot -> o, b -> o -> rot -> o);
  | std::swap(a -> o, b -> o);
}
Q conn(Q a, Q b) {
  | Q q = link(a -> F(), b -> p);
  | splice(q, a -> next());
  | splice(q -> r(), b);
  | return q;
}
std::pair<Q, Q> rec(const std::vector<p2> & s) {
  | int N = size(s);
  | if(N <= 3) {
  |   | Q a = link(s[0], s[1]), b = link(s[1], s.back());
  |   | if(N == 2) return {a, a -> r()};
  |   | splice(a -> r(), b);
  |   | ll side = cross(s[0], s[1], s[2]);
  |   | Q c = side ? conn(b, a) : 0;
  |   | return {side < 0 ? c->r() : a, side < 0 ? c : b -> r() };
  | }
#define H(e) e -> F(), e -> p
#define valid(e) (cross(e->F(), H(base)) > 0)
  | int half = N / 2;
  | auto [ra, A] = rec({s.begin(), s.end() - half});
  | auto [B, rb] = rec({s.end() - half, s.end()});
  | while((cross(B -> p, H(A)) < 0 && (A = A -> next())) ||
  |   |   | (cross(A -> p, H(B)) > 0 && (B = B -> r() -> o)));
  | Q base = conn(B -> r(), A);
  | if(A -> p == ra -> p) ra = base -> r();
  | if(B -> p == rb -> p) rb = base;
#define DEL(e, init, dir) Q e = init -> dir; if(valid(e)) \
  | for(;circ(e -> dir -> F(), H(base), e -> F());) { \
  |   | Q t = e -> dir; \
  |   | splice(e, e -> prev()); \
  |   | splice(e -> r(), e -> r() -> prev()); \
  |   | e -> o = H, H = e, e = t; \
  | }
  | for(;;) {
  |   | DEL(LC, base -> r(), o);
  |   | DEL(RC, base, prev());
  |   | if(!valid(LC) && !valid(RC)) break;
  |   | if(!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
  |   |   | base = conn(RC, base -> r());
  |   | else
  |   |   | base = conn(base -> r(), LC -> r());
  | }
  | return {ra, rb};
}
std::vector<p2> triangulate(std::vector<p2> a) {
  | sort(a.begin(), a.end()); // unique
  | if((int)size(a) < 2) return {};
  | Q e = rec(a).first;
  | std::vector<Q> q = {e};
  | while(cross(e -> o -> F(), e -> F(), e -> p) < 0) e = e -> o;
#define ADD { Q c = e; do { c -> mark = 1; a.push_back(c -> p); \
  | q.push_back(c -> r()), c = c -> next(); } while(c != e); }
  | ADD; a.clear();
  | for(int qi = 0;qi < (int) size(q);) if(!(e = q[qi++]) -> mark) ADD;
  | return a;
} // 返回若干逆时针三角形 \{t[0][0], t[0][1], t[0][2], t[1][0], \dots\}
```

7 geometry3d

7.1 向量

```
struct p3 {
  | db x, y, z;
  | db norm() const { return x * x + y * y + z * z; }
  | db abs() const { return std::sqrt(norm()); }
};
p3 operator + (p3 x, p3 y){ return {x.x + y.x, x.y + y.y, x.z + y.z}; }
p3 operator - (p3 x, p3 y){ return {x.x - y.x, x.y - y.y, x.z - y.z}; }
p3 operator * (p3 x, db y) { return {x.x * y, x.y * y, x.z * y}; }
p3 operator / (p3 x, db y) { return {x.x / y, x.y / y, x.z / y}; }
p3 operator * (p3 x, p3 y) { // 三维叉积需要更高的精度
  | return {
  |   | x.y * y.z - x.z * y.y,
  |   | x.z * y.x - x.x * y.z,
  |   | x.x * y.y - x.y * y.x
  | };
}
db operator % (p3 x, p3 y) { return x.x * y.x + x.y * y.y + x.z * y.z; }
p3 perpvec(p3 x) {
  | return fabs(x.x) > fabs(x.z) ? p3{ x.y, -x.x, 0 } : p3{0, -x.z, x.y};
} // 找到一个与给定向量垂直的向量
db area(p3 a, p3 b, p3 c) { return ((b - a) * (c - a)).abs(); } // 三角形面积两
  ↪ 倍
db volume(p3 d, p3 a, p3 b, p3 c) { // 四面体有向体积六倍
  | return (d - a) % ((b - a) * (c - a));
}
```

7.2 平面

```
struct plane {
  p3 n; db d; // n dot x = d
  plane() {}
  plane(p3 a, p3 b, p3 c) : n((c - a) * (b - a)) { d = n % a; }
  db side(p3 x) const { return n % x - d; }
  db dist(p3 w) const { return side(w) / n.abs(); }
  p3 proj(p3 w) const { return w - n * (side(w) / n.abs()); }
};
```

7.3 直线

```
struct line3 {
  p3 d, o; // kd + o
  line3() {}
  line3(p3 p, p3 q) : d(q - p), o(p) {}
  line3(plane p1, plane p2) : d(p1.n * p2.n) { // 平面交出直线
    o = (p2.n * p1.d - p1.n * p2.d) * d / d.norm();
  }
  db dist(p3 p) const { return (d * (p - o)).abs() / d.abs(); }
  p3 proj(p3 p) const { return o + d * (d % (p - o)) / d.norm(); }//投影
  p3 relf(p3 p) const { return proj(p) * 2 - p; } // 对称
  p3 operator & (const plane & p) const { // 线与平面交
    return o - d * p.side(o) / (p.n % d);
  }
};
db dist(line3 l1, line3 l2) {
  p3 n = l1.d * l2.d;
  if(n.abs() < eps) return l1.dist(l2.o);
  return abs((l2.o - l1.o) % n) / n.abs();
}
p3 closestOnL1(line3 l1, line3 l2) {
  p3 n2 = l2.d * (l1.d * l2.d);
  return l1.o + l1.d * ((l2.o - l1.o) % n2) / (l1.d % n2);
}
bool ispara(plane p1, plane p2){return(p1.n * p2.n).abs() < eps;}//判断是否平行
bool ispara(line3 p1, line3 p2){return(p1.d * p2.d).abs() < eps;}//判断是否平行
bool isperp(plane p1, plane p2){return fabs(p1.n % p2.n) < eps;}//判断是否垂直
bool isperp(line3 p1, line3 p2){return fabs(p1.d % p2.d) < eps;}//判断是否垂直
line3 perpthrough(plane p, p3 o){return line3(o, o + p.n);}//过平面一点做垂线
```

7.4 凸包

```
const int N = 2005;
struct face { int a[3]; plane p; };
int vis[N][N];
std::vector<face> f;
void convex3d(const std::vector<p3> & a) { // need to deal coplane
  if(a.size() < 3) return ;
  auto getface = [&](int i, int j, int k) -> face { return {{i, j, k},
    ↪plane(a[i], a[j], a[k])}; };
  f = {getface(0, 1, 2), getface(0, 2, 1)};
  std::vector<face> tmp[2];
  for(int i = 3;i < (int) a.size();++i) {
    for(auto x : f) {
      if(x.p.dist(a[i]) < -eps) {
        tmp -> push_back(x);
      } else {
        tmp[1].push_back(x);
        for(int t : {0, 1, 2}) vis[x.a[t]][x.a[(t + 1) % 3]] = i;
      }
    }
    for(auto x : tmp[1]) {
      for(int t : {0, 1, 2}) {
        if(vis[x.a[t]][x.a[(t + 1) % 3]] == i && vis[x.a[(t + 1) % 3]]
          ↪[x.a[t]] != i)
          tmp[0].push_back(getface(x.a[t], x.a[(t + 1) % 3], i));
      }
    }
    f = tmp[0]; tmp[0].clear(); tmp[1].clear();
  }
  for(int i = 0;i < (int)a.size();++i) memset(vis[i], 0, a.size() << 2);
}
```

8 Misc

8.1 Pragma

```
#pragma GCC optimize("Ofast")
#pragma GCC optimize("unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,avx2")
#pragma pack(1) // default=8
```

8.2 Barrett

```
struct DIV {
  u64 x;
  void init(u64 v) { x = -1ull / v + 1; }
}; // 带误差版本 x = -1ull/v;
// ret=ans while x*(y-1)<2^64, ans-1<=ret<=ans while x<2^64
u64 operator / (const u64 & x, const DIV & y) {
  return (u128) x * y.x >> 64;
}
```

8.3 LCS

```
int lim;
struct bitset {
  static const int B = 63;
  u64 a[N / B + 1];
  void set(int p) { a[p / B] |= 1ull << (p % B); }
  bool test(int p) { return a[p / B] >> (p % B) & 1; }
  void run(const bitset & o) {
    u64 c = 1;
    for(int i = 0;i < lim;++i) {
      u64 x = a[i], y = x | o.a[i];
      x += x + c + (~y & (1ull << 63) - 1);
      a[i] = x & y, c = x >> 63;
    }
  }
} dp;
```

8.4 日期公式

```
// Mon = 0, ... % 7
// days since 1/1/1
int getday(int y, int m, int d) {
  if(m < 3) -- y, m += 12;
  return (365 * y + y / 4 - y / 100 + y / 400 + (153 * (m - 3) + 2) / 5 + d
    ↪ - 307);
}
void date(int n, int & y, int & m, int & d) {
  n += 429 + ((4 * n + 1227) / 146097 + 1) * 3 / 4;
  y = (4 * n - 489) / 1461;
  n -= y * 1461 / 4;
  m = (5 * n - 1) / 153;
  d = n - m * 153 / 5;
  if (--m > 12) m -= 12, ++y;
}
```

8.5 Xorshift

```
u64 xorshift(u64 x) { x ^= x << 13; x ^= x >> 7; x ^= x << 17; return x; }
u32 xorshift(u32 x) { x ^= x << 13; x ^= x >> 17; x ^= x << 5; return x; }
```

9 配置

9.1 vimrc

```
set si ci ts=4 sw=4 nu cino=j1 backup undofile
syntax on
map<F9> <ESC>:!make %<<CR>
map<F10> <ESC>:!./%<<CR>
map<F4> <ESC>:!gdb %<<CR>
```

9.2 bashrc

```
export CXXFLAGS='-g -Wall -fsanitize=address,undefined -Dzqj -std=gnu++20'
mk() { g++ -O2 -Dzqj -std=gnu++20 $1.cpp -o $1; }
ulimit -s 1048576
ulimit -v 1048576
```

9.3 对拍

需要 chmod +x

```
while true; do
  ./gen > 1.in
  ./naive < 1.in > std.out
  ./a < 1.in > 1.out
  if diff 1.out std.out; then
    echo ac
  else
    echo wa
    break
  fi
done
```

9.4 编译参数

```
-D_GLIBCXX_DEBUG : STL debug mode
-fsanitize=address : 内存错误检查
-fsanitize=undefined : UB 检查
```

9.5 随机素数

```
979345007 986854057502126921
935359631 949054338673679153
931936021 989518940305146613
984974633 972090414870546877
984858209 956380060632801307
```

9.6 常数表

n	$\log_{10} n$	$n!$	$C(n, n/2)$	$\mathrm{LCM}(1 \dots n)$	P_n
2	0.30102999	2	2	2	2
3	0.47712125	6	3	6	3
4	0.60205999	24	6	12	5
5	0.69897000	120	10	60	7
6	0.77815125	720	20	60	11
7	0.84509804	5040	35	420	15
8	0.90308998	40320	70	840	22
9	0.95424251	362880	126	2520	30
10	1	3628800	252	2520	42
11	1.04139269	39916800	462	27720	56
12	1.07918125	479001600	924	27720	77
15	1.17609126	1.31e12	6435	360360	176
20	1.30103000	2.43e18	184756	232792560	627
25	1.39794001	1.55e25	5200300	26771144400	1958
30	1.47712125	2.65e32	155117520	1.444e14	5604
P_n	37338_{40}	204226_{50}	966467_{60}	190569292_{100}	$1e9_{114}$

$n \le$	10	100	1e3	1e4	1e5	1e6
$\max \omega(n)$	2	3	4	5	6	7
$\max d(n)$	4	12	32	64	128	240
$\pi(n)$	4	25	168	1229	9592	78498
$n \le$	1e7	1e8	1e9	1e10	1e11	1e12
$\max \omega(n)$	8	8	9	10	10	11
$\max d(n)$	448	768	1344	2304	4032	6720
$\pi(n)$	664579	5761455	5.08e7	4.55e8	4.12e9	3.7e10
$n \le$	1e13	1e14	1e15	1e16	1e17	1e18
$\max \omega(n)$	12	12	13	13	14	15
$\max d(n)$	10752	17280	26880	41472	64512	103680
$\pi(n)$	Prime number theorem: $\pi(x) \sim x/\log(x)$					

10 注意事项

10.1 测试项目

pbds tree, float128, int128, long double submit 命令, printfile, MLE ?= RE, pragma, axv2, python,

10.2 bugs

看数据范围（多测总和）, 变量 shadow, 清空, long long, 数组大小, 模数, MLE?, 对拍记得看输出在不在变, 输出格式，inf 开小，答案初值，STL 重构导致引用失效，极端情况 (n=1)

11 tables

11.1 导数积分

$$(\frac{u}{v})' = \frac{u'v-uv'}{v^2}$$
$$(a^x)' = (\ln a)a^x$$
$$(\tan x)' = \sec^2 x$$
$$(\cot x)' = \csc^2 x$$
$$(\sec x)' = \tan x \sec x$$
$$(\csc x)' = -\cot x \csc x$$
$$(\arcsin x)' = \frac{1}{\sqrt{1-x^2}}$$
$$(\arccos x)' = -\frac{1}{\sqrt{1-x^2}}$$
$$(\arctan x)' = \frac{1}{1+x^2}$$
$$(\operatorname{arccot} x)' = -\frac{1}{1+x^2}$$
$$(\operatorname{arccsc} x)' = -\frac{1}{x\sqrt{1-x^2}}$$
$$(\operatorname{arcsec} x)' = \frac{1}{x\sqrt{1-x^2}}$$
$$(\tanh x)' = \operatorname{sech}^2 x$$
$$(\coth x)' = -\operatorname{csch}^2 x$$
$$(\operatorname{sech} x)' = -\operatorname{sech} x \tanh x$$
$$(\operatorname{csch} x)' = -\operatorname{csch} x \coth x$$
$$(\operatorname{arcsinh} x)' = \frac{1}{\sqrt{1+x^2}}$$
$$(\operatorname{arccosh} x)' = \frac{1}{\sqrt{x^2-1}}$$
$$(\operatorname{arctanh} x)' = \frac{1}{1-x^2}$$
$$(\operatorname{arccoth} x)' = \frac{1}{x^2-1}$$
$$(\operatorname{arccsch} x)' = -\frac{1}{|x|\sqrt{1+x^2}}$$
$$(\operatorname{arcsech} x)' = -\frac{1}{x\sqrt{1-x^2}}$$

$ax^2+bx+c(a>0)$

1. $\int \frac{\mathrm{d}x}{ax^2+bx+c} = \begin{cases} \frac{2}{\sqrt{4ac-b^2}} \arctan \frac{2ax+b}{\sqrt{4ac-b^2}} + C & (b^2 < 4ac) \\ \frac{1}{\sqrt{b^2-4ac}} \ln \left| \frac{2ax+b-\sqrt{b^2-4ac}}{2ax+b+\sqrt{b^2-4ac}} \right| + C & (b^2 > 4ac) \end{cases}$
2. $\int \frac{x}{ax^2+bx+c}\mathrm{d}x = \frac{1}{2a}\ln|ax^2+bx+c| - \frac{b}{2a}\int \frac{\mathrm{d}x}{ax^2+bx+c}$

$\sqrt{\pm ax^2+bx+c}(a>0)$

1. $\int \frac{\mathrm{d}x}{\sqrt{ax^2+bx+c}} = \frac{1}{\sqrt{a}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C$
2. $\int \sqrt{ax^2+bx+c}\mathrm{d}x = \frac{2ax+b}{4a}\sqrt{ax^2+bx+c} + \frac{4ac-b^2}{8\sqrt{a^3}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C$
3. $\int \frac{x}{\sqrt{ax^2+bx+c}}\mathrm{d}x = \frac{1}{a}\sqrt{ax^2+bx+c} - \frac{b}{2\sqrt{a^3}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C$
4. $\int \frac{\mathrm{d}x}{\sqrt{c+bx-ax^2}} = -\frac{1}{\sqrt{a}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}} + C$
5. $\int \sqrt{c+bx-ax^2}\mathrm{d}x = \frac{2ax-b}{4a}\sqrt{c+bx-ax^2} + \frac{b^2+4ac}{8\sqrt{a^3}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}} + C$
6. $\int \frac{x}{\sqrt{c+bx-ax^2}}\mathrm{d}x = -\frac{1}{a}\sqrt{c+bx-ax^2} + \frac{b}{2\sqrt{a^3}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}} + C$

$\sqrt{\pm\frac{x-a}{x-b}}$ 或 $\sqrt{(x-a)(x-b)}$

1. $\int \frac{\mathrm{d}x}{\sqrt{(x-a)(b-x)}} = 2\arcsin\sqrt{\frac{x-a}{b-x}} + C\,(a<b)$
2. $\int \sqrt{(x-a)(b-x)}\mathrm{d}x = \frac{2x-a-b}{4}\sqrt{(x-a)(b-x)} + \frac{(b-a)^2}{4}\arcsin\sqrt{\frac{x-a}{b-x}} + C, (a<b)$

三角函数的积分

1. $\int \tan x\mathrm{d}x = -\ln|\cos x| + C$
2. $\int \cot x\mathrm{d}x = \ln|\sin x| + C$
3. $\int \sec x\mathrm{d}x = \ln\left|\tan\left(\frac{\pi}{4}+\frac{x}{2}\right)\right| + C = \ln|\sec x+\tan x| + C$
4. $\int \csc x\mathrm{d}x = \ln\left|\tan\frac{x}{2}\right| + C = \ln|\csc x-\cot x| + C$
5. $\int \sec^2 x\mathrm{d}x = \tan x + C$
6. $\int \csc^2 x\mathrm{d}x = -\cot x + C$
7. $\int \sec x\tan x\mathrm{d}x = \sec x + C$
8. $\int \csc x\cot x\mathrm{d}x = -\csc x + C$
9. $\int \sin^2 x\mathrm{d}x = \frac{x}{2} - \frac{1}{4}\sin 2x + C$
10. $\int \cos^2 x\mathrm{d}x = \frac{x}{2} + \frac{1}{4}\sin 2x + C$
11. $\int \sin^n x\mathrm{d}x = -\frac{1}{n}\sin^{n-1}x\cos x + \frac{n-1}{n}\int \sin^{n-2}x\mathrm{d}x$
12. $\int \cos^n x\mathrm{d}x = \frac{1}{n}\cos^{n-1}x\sin x + \frac{n-1}{n}\int \cos^{n-2}x\mathrm{d}x$
13. $\int \frac{\mathrm{d}x}{\sin^n x} = -\frac{1}{n-1}\frac{\cos x}{\sin^{n-1}x} + \frac{n-2}{n-1}\int \frac{\mathrm{d}x}{\sin^{n-2}x}$
14. $\int \frac{\mathrm{d}x}{\cos^n x} = \frac{1}{n-1}\frac{\sin x}{\cos^{n-1}x} + \frac{n-2}{n-1}\int \frac{\mathrm{d}x}{\cos^{n-2}x}$
15. $$\begin{aligned} &\int \cos^m x\sin^n x\mathrm{d}x \\ =&\frac{1}{m+n}\cos^{m-1}x\sin^{n+1}x + \frac{m-1}{m+n}\int \cos^{m-2}x\sin^n x\mathrm{d}x \\ =&-\frac{1}{m+n}\cos^{m+1}x\sin^{n-1}x + \frac{n-1}{m+1}\int \cos^m x\sin^{n-2}x\mathrm{d}x \end{aligned}$$
16. $\int \frac{\mathrm{d}x}{a+b\sin x} = \begin{cases} \frac{2}{\sqrt{a^2-b^2}}\arctan\frac{a\tan\frac{x}{2}+b}{\sqrt{a^2-b^2}} + C & (a^2>b^2) \\ \frac{1}{\sqrt{b^2-a^2}}\ln\left|\frac{a\tan\frac{x}{2}+b-\sqrt{b^2-a^2}}{a\tan\frac{x}{2}+b+\sqrt{b^2-a^2}}\right| + C & (a^2<b^2) \end{cases}$
17. $\int \frac{\mathrm{d}x}{a+b\cos x} = \begin{cases} \frac{2}{a+b}\sqrt{\frac{a+b}{a-b}}\arctan\left(\sqrt{\frac{a-b}{a+b}}\tan\frac{x}{2}\right) + C & (a^2>b^2) \\ \frac{1}{a+b}\sqrt{\frac{a+b}{a-b}}\ln\left|\frac{\tan\frac{x}{2}+\sqrt{\frac{a+b}{b-a}}}{\tan\frac{x}{2}-\sqrt{\frac{a+b}{b-a}}}\right| + C & (a^2<b^2) \end{cases}$
18. $\int \frac{\mathrm{d}x}{a^2\cos^2 x+b^2\sin^2 x} = \frac{1}{ab}\arctan\left(\frac{b}{a}\tan x\right) + C$
19. $\int \frac{\mathrm{d}x}{a^2\cos^2 x-b^2\sin^2 x} = \frac{1}{2ab}\ln\left|\frac{b\tan x+a}{b\tan x-a}\right| + C$
20. $\int x\sin ax\mathrm{d}x = \frac{1}{a^2}\sin ax - \frac{1}{a}x\cos ax + C$
21. $\int x^2\sin ax\mathrm{d}x = -\frac{1}{a}x^2\cos ax + \frac{2}{a^2}x\sin ax + \frac{2}{a^3}\cos ax + C$
22. $\int x\cos ax\mathrm{d}x = \frac{1}{a^2}\cos ax + \frac{1}{a}x\sin ax + C$
23. $\int x^2\cos ax\mathrm{d}x = \frac{1}{a}x^2\sin ax + \frac{2}{a^2}x\cos ax - \frac{2}{a^3}\sin ax + C$

反三角函数的积分 (其中 $a>0$)

1. $\int \arcsin\frac{x}{a}\mathrm{d}x = x\arcsin\frac{x}{a} + \sqrt{a^2-x^2} + C$
2. $\int x\arcsin\frac{x}{a}\mathrm{d}x = (\frac{x^2}{2}-\frac{a^2}{4})\arcsin\frac{x}{a} + \frac{x}{4}\sqrt{x^2-x^2} + C$
3. $\int x^2\arcsin\frac{x}{a}\mathrm{d}x = \frac{x^3}{3}\arcsin\frac{x}{a} + \frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2} + C$
4. $\int \arccos\frac{x}{a}\mathrm{d}x = x\,arccos\frac{x}{a} - \sqrt{a^2-x^2} + C$
5. $\int x\arccos\frac{x}{a}\mathrm{d}x = (\frac{x^2}{2}-\frac{a^2}{4})\arccos\frac{x}{a} - \frac{x}{4}\sqrt{a^2-x^2} + C$
6. $\int x^2\arccos\frac{x}{a}\mathrm{d}x = \frac{x^3}{3}\arccos\frac{x}{a} - \frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2} + C$
7. $\int \arctan\frac{x}{a}\mathrm{d}x = x\arctan\frac{x}{a} - \frac{a}{2}\ln(a^2+x^2) + C$
8. $\int x\arctan\frac{x}{a}\mathrm{d}x = \frac{1}{2}(a^2+x^2)\arctan\frac{x}{a} - \frac{a}{2}x + C$
9. $\int x^2\arctan\frac{x}{a}\mathrm{d}x = \frac{x^3}{3}\arctan\frac{x}{a} - \frac{a}{6}x^2 + \frac{a^3}{6}\ln(a^2+x^2) + C$

指数函数的积分

1. $\int a^x\mathrm{d}x = \frac{1}{\ln a}a^x + C$
2. $\int \mathrm{e}^{ax}\mathrm{d}x = \frac{1}{a}a^{ax} + C$
3. $\int x\mathrm{e}^{ax}\mathrm{d}x = \frac{1}{a^2}(ax-1)a^{ax} + C$
4. $\int x^n\mathrm{e}^{ax}\mathrm{d}x = \frac{1}{a}x^n\mathrm{e}^{ax} - \frac{n}{a}\int x^{n-1}\mathrm{e}^{ax}\mathrm{d}x$
5. $\int xa^x\mathrm{d}x = \frac{x}{\ln a}a^x - \frac{1}{(\ln a)^2}a^x + C$
6. $\int x^n a^x\mathrm{d}x = \frac{1}{\ln a}x^n a^x - \frac{n}{\ln a}\int x^{n-1}a^x\mathrm{d}x$
7. $\int \mathrm{e}^{ax}\sin bx\mathrm{d}x = \frac{1}{a^2+b^2}\mathrm{e}^{ax}(a\sin bx - b\cos bx) + C$
8. $\int \mathrm{e}^{ax}\cos bx\mathrm{d}x = \frac{1}{a^2+b^2}\mathrm{e}^{ax}(b\sin bx + a\cos bx) + C$
9. $\int \mathrm{e}^{ax}\sin^n bx\mathrm{d}x = \frac{1}{a^2+b^2n^2}\mathrm{e}^{ax}\sin^{n-1}bx(a\sin bx - nb\cos bx) + \frac{n(n-1)b^2}{a^2+b^2n^2}\int \mathrm{e}^{ax}\sin^{n-2}bx\mathrm{d}x$
10. $\int \mathrm{e}^{ax}\cos^n bx\mathrm{d}x = \frac{1}{a^2+b^2n^2}\mathrm{e}^{ax}\cos^{n-1}bx(a\cos bx + nb\sin bx) + \frac{n(n-1)b^2}{a^2+b^2n^2}\int \mathrm{e}^{ax}\cos^{n-2}bx\mathrm{d}x$

对数函数的积分

1. $\int \ln x\mathrm{d}x = x\ln x - x + C$
2. $\int \frac{\mathrm{d}x}{x\ln x} = \ln|\ln x| + C$
3. $\int x^n\ln x\mathrm{d}x = \frac{1}{n+1}x^{n+1}(\ln x - \frac{1}{n+1}) + C$
4. $\int (\ln x)^n\mathrm{d}x = x(\ln x)^n - n\int (\ln x)^{n-1}\mathrm{d}x$
5. $\int x^m(\ln x)^n\mathrm{d}x = \frac{1}{m+1}x^{m+1}(\ln x)^n - \frac{n}{m+1}\int x^m(\ln x)^{n-1}\mathrm{d}x$

STL 积分/求和 (need std::)

1. $\int_0^1 t^{x-1}(1-t)^{y-1}dt = beta(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$
2. $\int_0^{+\infty} t^{num-1}e^{-t}dt = tgamma(num) = e^{lgamma(num)} = \Gamma(num)$
3. $\int_0^{phi} \frac{d\theta}{\sqrt{1-k^2\sin^2\theta}} = ellint\_1(k,phi)$
4. $\int_0^{phi} \sqrt{1-k^2\sin^2\theta}d\theta = ellint\_2(k,phi)$
5. $\int_{num}^{+\infty} \frac{e^{-t}}{t}dt = -expint(-num)$
6. $\sum_{n=1}^{+\infty} n^{-num} = riemann\_zeta(num)$
7. $\frac{2}{\sqrt{\pi}}\int_0^{arg} e^{-t^2}dt = erf(arg)$