Data Structure

SegmentTree

```
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info>info;
    SegmentTree(const int & n) {
        this->n = n;
        info.assign(4 << std::__lg(n), Info());
    }
    SegmentTree(const std::vector<Info>& a) {
        int n = a.size() - 1;
        this->n = n;
        info.assign(4 << std::__lg(n), Info());
        auto work = [&](auto && self, int p, int l, int r) {
            if (l == r) {
                info[p] = Info(a[l]);
                return;
            }
            int mid = (l + r) >> 1;
            self(self, p << 1, l, mid), self(self, p << 1 | 1, mid + 1, r);
            info[p] = info[p << 1] + info[p << 1 | 1];
        };
        work(work, 1, 1, n);
    }
    void modify(int p, int l, int r, int L, int R, const Info& v) {
        if (l > R or r < L) {
            return;
        }
        if (L <= l and r <= R) {
            info[p] = v;
            return;
        }
        int mid = (l + r) >> 1;
        modify(p << 1, l, mid, L, R, v), modify(p << 1 | 1, mid + 1, r, L, R,
v);
        info[p] = info[p << 1] + info[p << 1 | 1];
    }
    void modify(int p, const Info& v) {
        modify(1, 1, n, p, p, v);
    }
    Info rangeQuery(int p, int l, int r, int L, int R) {
        if (l > R or r < L) {
            return Info();
        }
        if (L <= l and r <= R) {
            return info[p];
        }
        int mid = (l + r) >> 1;
        return rangeQuery(p << 1, l, mid, L, R) + rangeQuery(p << 1 | 1, mid +
1, r, L, R);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 1, n, l, r);
    }
    template<class F>
```

```
    int findFirst(int p, int l, int r, int L, int R, F pred) {
        if (l > R or r < L or not pred(info[p])) {
            return -1;
        }
        if (l == r) {
            return l;
        }
        int mid = (l + r) >> 1;
        int res = findFirst(p << 1, l, mid, L, R, pred);
        return res == -1 ? findFirst(p << 1 | 1, mid + 1, r, L, R, pred) : res;
    }
    template<class F>
    int findFirst(int l, int r, F pred) {
        return findFirst(1, 1, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int L, int R, F pred) {
        if (l > R or r < L or not pred(info[p])) {
            return -1;
        }
        if (l == r) {
            return r;
        }
        int mid = (l + r) >> 1;
        int res = findLast(p << 1 | 1, mid + 1, r, L, R, pred);
        return res == -1 ? findLast(p << 1, l, mid, L, R, pred) : res;
    }
    template<class F>
    int findLast(int l, int r, F pred) {
        return findLast(1, 1, n, l, r, pred);
    }
    template<class F>
    int findPrefixFirst(int p, int l, int r, int L, int R, const F& pred, Info&
pref) {
        if (l > R or r < L) {
            return r + 1;
        }
        if (L <= l and r <= R) {
            if (not pred(pref + info[p])) {
                pref = pref + info[p];
                return r + 1;
            }
            if (l == r) {
                return l;
            }
            int mid = (l + r) >> 1;
            int res;
            if (pred(pref + info[p << 1])) {
                res = findPrefixFirst(p << 1, l, mid, L, R, pred, pref);
            } else {
                pref = pref + info[p << 1];
                res = findPrefixFirst(p << 1 | 1, mid + 1, r, L, R, pred,
pref);
            }
            return res;
        }
        int mid = (l + r) >> 1;
        int res = mid + 1;
        if (L <= mid) {
            res = findPrefixFirst(p << 1, l, mid, L, R, pred, pref);
```

```
        }
        if (res == mid + 1 and mid + 1 <= R) {
            res = findPrefixFirst(p << 1 | 1, mid + 1, r, L, R, pred, pref);
        }
        return res;
    }
    template<class F>
    int findPrefixFirst(int l, int r, const F& pred) {
        Info pref = Info();
        int res = findPrefixFirst(1, 1, n, l, r, pred, pref);
        return res == r + 1 ? -1 : res;
    }
    template<class F>
    int findSurfixLast(int p, int l, int r, int L, int R, const F& pred, Info&
surf) {
        if (l > R or r < L) {
            return l - 1;
        }
        if (L <= l and r <= R) {
            if (not pred(surf + info[p])) {
                surf = surf + info[p];
                return l - 1;
            }
            if (l == r) {
                return r;
            }
            int mid = (l + r) >> 1;
            int res;
            if (pred(surf + info[p << 1 | 1])) {
                res = findSurfixLast(p << 1 | 1, mid + 1, r, L, R, pred, surf);
            } else {
                surf = surf + info[p << 1 | 1];
                res = findSurfixLast(p << 1, l, mid, L, R, pred, surf);
            }
            return res;
        }
        int mid = (l + r) >> 1;
        int res = mid;
        if (mid + 1 <= R) {
            res = findSurfixLast(p << 1 | 1, mid + 1, r, L, R, pred, surf);
        }
        if (L <= mid and res == mid) {
            res = findSurfixLast(p << 1, l, mid, L, R, pred, surf);
        }
        return res;
    }
    template<class F>
    int findSurfixLast(int l, int r, const F& pred) {
        Info surf = Info();
        int res = findSurfixLast(1, 1, n, l, r, pred, surf);
        return res == l - 1 ? -1 : res;
    }
};
```

动态维护树的直径，保证边权为正，否则贪心不成立（依赖欧拉序实现贪心）

$dist(u,v) = dep_u + dep_v - 2 * dep_{lca}$

$maxl$ 表示在考虑同一棵子树中 $dep_{max} - 2 * dep_{min}$，然后可以跨子树合并，
$maxl = dep_{u,max} - 2 * dep_{u,min} + dep_{v,max}$

```
struct Tag {
    i64 add = 0;
    void apply(const Tag & t) {
        add += t.add;
    }
};

constexpr i64 inf = 1E18;
struct Info {
    i64 diameter = 0;
    i64 max = 0, min = 0, maxl = 0, maxr = 0;
    void apply(const Tag & t) {
        max += t.add;
        min -= t.add * 2;
        maxl -= t.add;
        maxr -= t.add;
    }
};

Info operator+(const Info & a, const Info & b) {
    Info c {};
    c.diameter = std::max({a.diameter, b.diameter, a.maxl + b.max, a.max +
b.maxr});
    c.max = std::max(a.max, b.max);
    c.min = std::max(a.min, b.min);
    c.maxl = std::max({a.maxl, b.maxl, a.max + b.min});
    c.maxr = std::max({a.maxr, b.maxr, a.min + b.max});
    return c;
}

auto main() ->int32_t {
    int n, q;
    i64 w;
    std::cin >> n >> q >> w;
    std::vector<std::array<i64, 3>>edges(n);
    std::vector<std::vector<std::pair<int,i64>>>adj(n + 1);
    for (int i = 1; i <= n - 1; i += 1) {
        auto& [u, v, w] = edges[i];
        std::cin >> u >> v >> w;
        adj[u].push_back({v, w});
        adj[v].push_back({u, w});
    }
    int cur = 0;
    const int m = 2 * n - 1;
    std::vector<i64>top(n + 1);
    std::vector<int>ord(m + 1), l(n + 1), r(n + 1);
    auto dfs = [&] (this auto && dfs, int u, int par) ->void {
        ord[++cur] = u;
        l[u] = cur;
        for (const auto & [v, w] : adj[u]) {
            if (v == par) {
                continue;
            }
            top[v] = w;
            dfs(v, u);
            ord[++cur] = u;
        }
        r[u] = cur;
    };
    dfs(1, 0);
```

```
    LazySegmentTree<Info, Tag>seg(m);
    for (int u = 2; u <= n; u += 1) {
        seg.rangeApply(l[u], r[u], {top[u]});
    }

    i64 last = 0;
    while (q--) {
        i64 d, e;
        std::cin >> d >> e;
        d = (d + last) % (n - 1);
        d += 1;
        e = (e + last) % w;
        auto [u, v, x] = edges[d];
        if (l[u] < l[v]) {
            std::swap(u, v);
        }
        seg.rangeApply(l[u], r[u], {e - top[u]});
        top[u] = e;
        std::cout << (last = seg.info[1].diameter) << '\n';
    }
    return 0;
}
```

```
// 静态子树直径
std::vector<std::array<int, 2>>t(n + 1);
auto dfs = [&](this auto && dfs, int u, int par) ->void {
    t[u] = {u, u};
    for (const auto & v : adj[u]) {
        if (v == par) {
            continue;
        }
        dfs(v, u);
        int cur = 0;
        std::array<int, 4> p{t[u][0], t[u][1], t[v][0], t[v][1]};
        for (const auto & a : p) {
            for (const auto & b : p) {
                if (cur < hld.dist(a, b)) {
                    cur = hld.dist(a, b);
                    t[u] = {a, b};
                }
            }
        }
    }
};
dfs(1, 0);
```

求区间众数：

solution 1: 莫队

solution 2: 主席树，每一次往 cnt 大的子树走进去即可。

solution 3: 随机化取值，二分求区间某个数的出现次数, 因为区间众数的出现次数足够多，正确性得以保证

```
int x = a[rand(l, r)];
int c = std::ranges::upper_bound(vec[x], r) - std::ranges::lower_bound(vec[x],
l);
```

solution 4: 线段树维护摩尔投票法

由于主元素的出现的次数超过 ，那么在不断消掉两个不同的元素之后，最后一定剩下主元素。

由于我们只需要关心元素的值而不关心其位置，故可用 `val` 和 `cnt` 两个变量代替完整存储元素。

(摩尔投票法的正确性建立在区间存在绝对众数，如果没有保证区间存在绝对众数需要验证)

```
struct Info {
    int val = 0;
    int cnt = 0;
    constexpr friend Info operator+(const Info& a, const Info& b) {
        if (a.val == b.val) {
            return {a.val, a.cnt + b.cnt};
        } else if (a.cnt >= b.cnt) {
            return {a.val, a.cnt - b.cnt};
        }
        return {b.val, b.cnt - a.cnt};
    }
};
```

LazySegmentTree

```
template<class Info, class Tag>
requires requires(Info info, Tag tag) {info.apply(tag); tag.apply(tag);}
struct LazySegmentTree {
    int n;
    std::vector<Info>info;
    std::vector<Tag>tag;
    LazySegmentTree (const int& n) {
        this->n = n;
        info.assign(4 << std::__lg(n), Info());
        tag.assign(4 << std::__lg(n), Tag());
    }
    LazySegmentTree (const std::vector<Info> & a) {
        int n = a.size() - 1;
        this->n = n;
        info.assign(4 << std::__lg(n), Info());
        tag.assign(4 << std::__lg(n), Tag());
        auto work = [&](auto && self, int p, int l, int r) {
            if (l == r) {
                info[p] = Info(a[l]);
                return;
            }
            int mid = (l + r) >> 1;
            self(self, p << 1, l, mid), self(self, p << 1 | 1, mid + 1, r);
            info[p] = info[p << 1] + info[p << 1 | 1];
        };
        work(work, 1, 1, n);
    }
    void apply(int p, const Tag& v) {
        info[p].apply(v), tag[p].apply(v);
    }
    void pull(int p) {
        apply(p << 1, tag[p]), apply(p << 1 | 1, tag[p]);
        tag[p] = Tag();
    }
    void modify(int p, int l, int r, int L, int R, const Info& v) {
        if (l > R or r < L) {
            return;
        }
        if (L <= l and r <= R) {
```

```
            info[p] = v;
            return;
        }
        pull(p);
        int mid = (l + r) >> 1;
        modify(p << 1, l, mid, L, R, v), modify(p << 1 | 1, mid + 1, r, L, R,
v);
        info[p] = info[p << 1] + info[p << 1 | 1];
    }
    void modify(int p, const Info& v) {
        modify(1, 1, n, p, p, v);
    }
    Info rangeQuery(int p, int l, int r, int L, int R) {
        if (l > R or r < L) {
            return Info();
        }
        if (L <= l and r <= R) {
            return info[p];
        }
        pull(p);
        int mid = (l + r) >> 1;
        return rangeQuery(p << 1, l, mid, L, R) + rangeQuery(p << 1 | 1, mid +
1, r, L, R);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 1, n, l, r);
    }
    void rangeApply(int p, int l, int r, int L, int R, const Tag& v) {
        if (l > R or r < L) {
            return;
        }
        if (L <= l and r <= R) {
            apply(p, v);
            return;
        }
        pull(p);
        int mid = (l + r) >> 1;
        rangeApply(p << 1, l, mid, L, R, v), rangeApply(p << 1 | 1, mid + 1, r,
L, R, v);
        info[p] = info[p << 1] + info[p << 1 | 1];
    }
    void rangeApply(int l, int r, const Tag& v) {
        rangeApply(1, 1, n, l, r, v);
    }
    template<class F>
    int findFirst(int p, int l, int r, int L, int R, F pred) {
        if (l > R or r < L or not pred(info[p])) {
            return -1;
        }
        if (l == r) {
            return l;
        }
        pull(p);
        int mid = (l + r) >> 1;
        int res = findFirst(p << 1, l, mid, L, R, pred);
        return res == -1 ? findFirst(p << 1 | 1, mid + 1, r, L, R, pred) : res;
    }
    template<class F>
    int findFirst(int l, int r, F pred) {
        return findFirst(1, 1, n, l, r, pred);
```

```
    }
    template<class F>
    int findLast(int p, int l, int r, int L, int R, F pred) {
        if (l > R or r < L or not pred(info[p])) {
            return -1;
        }
        if (l == r) {
            return l;
        }
        pull(p);
        int mid = (l + r) >> 1;
        int res = findLast(p << 1 | 1, mid + 1, r, L, R, pred);
        return res == -1 ? findLast(p << 1, l, mid, L, R, pred) : res;
    }
    template<class F>
    int findLast(int l, int r, F pred) {
        return findLast(1, 1, n, l, r, pred);
    }
    template<class F>
    int findPrefixFirst(int p, int l, int r, int L, int R, const F& pred, Info&
pref) {
        if (l > R or r < L) {
            return r + 1;
        }
        if (L <= l and r <= R) {
            if (l != r) {
                pull(p);
            }
            if (not pred(pref + info[p])) {
                pref = pref + info[p];
                return r + 1;
            }
            if (l == r) {
                return l;
            }
            int mid = (l + r) >> 1;
            int res;
            if (pred(pref + info[p << 1])) {
                res = findPrefixFirst(p << 1, l, mid, L, R, pred, pref);
            } else {
                pref = pref + info[p << 1];
                res = findPrefixFirst(p << 1 | 1, mid + 1, r, L, R, pred,
pref);
            }
            return res;
        }
        int mid = (l + r) >> 1;
        pull(p);
        int res = mid + 1;
        if (L <= mid) {
            res = findPrefixFirst(p << 1, l, mid, L, R, pred, pref);
        }
        if (res == mid + 1 and mid + 1 <= R) {
            res = findPrefixFirst(p << 1 | 1, mid + 1, r, L, R, pred, pref);
        }
        return res;
    }
    template<class F>
    int findPrefixFirst(int l, int r, const F& pred) {
        Info pref = Info();
```

```
        int res = findPrefixFirst(1, 1, n, l, r, pred, pref);
        return res == r + 1 ? -1 : res;
    }

    template<class F>
    int findSurfixLast(int p, int l, int r, int L, int R, const F& pred, Info&
surf) {
        if (l > R or r < L) {
            return l - 1;
        }
        if (L <= l and r <= R) {
            if (l != r) {
                pull(p);
            }
            if (not pred(surf + info[p])) {
                surf = surf + info[p];
                return l - 1;
            }
            if (l == r) {
                return r;
            }
            int mid = (l + r) >> 1;
            int res;
            if (pred(surf + info[p << 1 | 1])) {
                res = findSurfixLast(p << 1 | 1, mid + 1, r, L, R, pred, surf);
            } else {
                surf = surf + info[p << 1 | 1];
                res = findSurfixLast(p << 1, l, mid, L, R, pred, surf);
            }
            return res;
        }
        int mid = (l + r) >> 1;
        int res = mid;
        pull(p);
        if (mid + 1 <= R) {
            res = findSurfixLast(p << 1 | 1, mid + 1, r, L, R, pred, surf);
        }
        if (L <= mid and res == mid) {
            res = findSurfixLast(p << 1, l, mid, L, R, pred, surf);
        }
        return res;
    }
    template<class F>
    int findSurfixLast(int l, int r, const F& pred) {
        Info surf = Info();
        int res = findSurfixLast(1, 1, n, l, r, pred, surf);
        return res == l - 1 ? -1 : res;
    }
};
struct Tag {
    void apply(const Tag & t) {
    }
};
constexpr int inf = 1E9;
struct Info {
    void apply(const Tag & t) {
    }
};
Info operator+(const Info& a, const Info& b) {
}
```

```cpp
// 维护式子 \sum_{i = l}^{r - k + 1}\text{min}(a_{l},a_{i})
// 类似于笛卡尔树
std::vector<std::vector<std::pair<int, int>>>qry(n + 1);
for (int i = 1; i <= q; i += 1) {
    int l, r;
    std::cin >> l >> r;
    qry[l].push_back({r - k + 1, i});
}

std::vector<int>stk;
std::vector<i64>ans(q + 1);
LazySegmentTree<Info, Tag>seg(std::vector<Info>(n - k + 2, {1, 0}));
for (int i = n - k + 1; i >= 1; i -= 1) {
    while (!stk.empty() && c[i] <= c[stk.back()]) {
        int l = stk.back();
        stk.pop_back();
        int r = stk.empty() ? n - k + 2 : stk.back();
        seg.rangeApply(l, r - 1, { -c[l]});
    }
    int l = i, r = stk.empty() ? n - k + 2 : stk.back();
    seg.rangeApply(l, r - 1, {c[i]});
    stk.push_back(i);
    for (const auto & [r, j] : qry[i]) {
        ans[j] = seg.rangeQuery(i, r).val;
    }
}

// inter
i64 ans = 0;
LazySegmentTree<Info, Tag>seg(std::vector<Info>(n + 1, {0, 1}));
for (int i = 1; i <= n; i += 1) {
    adj[a[i]].push_back(i);
    if (adj[a[i]].size() == k) {
        seg.rangeApply(1, adj[a[i]][0], {1});
    } else if (adj[a[i]].size() > k) {
        int j = adj[a[i]].size();
        if (j == k + 1) {
            seg.rangeApply(1, adj[a[i]][0], { -1});
            seg.rangeApply(adj[a[i]][0] + 1, adj[a[i]][1], {1});
        } else {
            seg.rangeApply(adj[a[i]][j - k - 2] + 1, adj[a[i]][j - k - 1], {
-1});
            seg.rangeApply(adj[a[i]][j - k - 1] + 1, adj[a[i]][j - k], {1});
        }
    }
    auto v = seg.rangeQuery(1, i);
    ans += v.cnt * (v.val == 0);
}
std::cout << ans << '\n';
```

如何维护区间只有一个数在某个 bit 上为 0

$\text{info:}(\text{s} = \inf, \text{h} = 0)$, $c_h = (a_s \& b_h) | (a_h \& b_s)$.

PresidentTree

强制在线区间求 LCM ： 考虑根号分治

$\text{LCM} = \prod p^{\max_i^r cnt}$

根据唯一分解定理，对 x 进行分解，大于 $\sqrt{x}$ 的质因数的次幂最多为 1 次。

那么对小于根号的质因数每个维护一个ST表区间查询即可。空间复杂度为 $O(n\sqrt{n}\log_2 n)$,比较难以接受，但是注意到每个数的次幂不会太大，改为 $\mathrm{int16\_t}$ 即可。

那么对大于根号的质因数，每次区间查询区间里面不同的数的乘积即可。处理这个问题考虑主席树，记录每个数的pre，主席树维护区间乘积

维护 $\prod[pre_x \le l-1]x$ 即可。

```
struct PresidentTree {
    struct Info {
        int lsh = 0, rsh = 0;
        int cnt = 0;
        i64 sum = 0;
        friend Info operator+(const Info& a, const Info& b) {
            return { -1, -1, a.cnt + b.cnt, a.sum + b.sum};
        }
    };

    int tot;
    std::vector<Info>t;
    PresidentTree(int KN) {
        tot = 0;
        t.resize((KN << 5) + 1);
    }

    void add(int& now, int pre, int L, int R, int x, int v = 1) {
        t[now = ++tot] = t[pre];
        t[now].cnt += v;
        t[now].sum += x;
        if (L == R) {
            return;
        }
        int mid = (L + R) >> 1;
        if (x <= mid) {
            add(t[now].lsh, t[pre].lsh, L, mid, x, v);
        } else {
            add(t[now].rsh, t[pre].rsh, mid + 1, R, x, v);
        }
    }

    int getKthMin(int l, int r, int L, int R, int k) {
        if (t[r].cnt - t[l].cnt < k) {
            return -1;
        }
        if (L == R) {
            return L;
        }
        int mid = (L + R) >> 1;
        int all = t[t[r].lsh].cnt - t[t[l].lsh].cnt;
        if (k <= all) {
            return getKthMin(t[l].lsh, t[r].lsh, L, mid, k);
        } else {
            return getKthMin(t[l].rsh, t[r].rsh, mid + 1, R, k - all);
        }
    }

    int getKthMax(int l, int r, int L, int R, int k) {
        if (t[r].cnt - t[l].cnt < k) {
            return -1;
```

```
        }
        if (L == R) {
            return R;
        }
        int mid = (L + R) >> 1;
        int all = t[t[r].rsh].cnt - t[t[l].rsh].cnt;
        if (k <= all) {
            return getKthMax(t[l].rsh, t[r].rsh, mid + 1, R, k);
        } else {
            return getKthMax(t[l].lsh, t[r].lsh, L, mid, k - all);
        }
    }

    Info getRange(int l, int r, int L, int R, int x, int y) {
        if (L > y or R < x) {
            return Info();
        }
        if (x <= L and R <= y) {
            return Info(-1, -1, t[r].cnt - t[l].cnt, t[r].sum - t[l].sum);
        }
        int mid = (L + R) >> 1;
        return getRange(t[l].lsh, t[r].lsh, L, mid, x, y) + getRange(t[l].rsh,
t[r].rsh, mid + 1, R, x, y);
    }
};
// pst on tree && kth min
auto query = [&](int u, int v, int k) {
    int j = hld.lca(u, v);
    auto find = [&](auto && find, int u, int v, int j, int w, i64 L, i64 R, int
k) {
        if (L == R) {
            return L;
        }
        i64 mid = (L + R) >> 1;
        int all = pst.t[pst.t[u].lsh].cnt + pst.t[pst.t[v].lsh].cnt -
pst.t[pst.t[j].lsh].cnt - pst.t[pst.t[w].lsh].cnt;
        if (k <= all) {
            return find(find, pst.t[u].lsh, pst.t[v].lsh, pst.t[j].lsh,
pst.t[w].lsh, L, mid, k);
        } else {
            return find(find, pst.t[u].rsh, pst.t[v].rsh, pst.t[j].rsh,
pst.t[w].rsh, mid + 1, R, k - all);
        }
    };
    return find(find, t[u], t[v], t[j], t[par[j]], 1, inf, k);
};

//如何理解标记永久化实现主席树区间加法
constexpr int Kn = 2E5;
struct Node {
    int ch[2] {0, 0};
    i64 val = 0;
    i64 tag = 0;
} t[Kn << 5];

int tot = 0;
void merge(int p, int l, int r) {
    t[p].val = t[t[p].ch[0]].val + t[t[p].ch[1]].val + t[p].tag * (r - l + 1);
}
```

```
void build(int& p, int l, int r) {
    t[p = ++tot] = Node();
    if (l == r) {
        std::cin >> t[p].val;
        return ;
    }
    int mid = (l + r) >> 1;
    build(t[p].ch[0], l, mid);
    build(t[p].ch[1], mid + 1, r);
    merge(p, l, r);
}

void apply(int pre, int& cur, int l, int r, int L, int R, i64 d) {
    t[cur = ++tot] = t[pre];
    if (L <= l && r <= R) {
        t[cur].tag += d;
        t[cur].val += d * (r - l + 1);
        return ;
    }
    int mid = (l + r) >> 1;
    if (L <= mid) {
        apply(t[pre].ch[0], t[cur].ch[0], l, mid, L, R, d);
    }
    if (R >= mid + 1) {
        apply(t[pre].ch[1], t[cur].ch[1], mid + 1, r, L, R, d);
    }
    merge(cur, l, r);
}

i64 query(int p, int l, int r, int L, int R, i64 T) {
    if (L <= l && r <= R) {
        return t[p].val + T * (r - l + 1);
    }
    T += t[p].tag;
    int mid = (l + r) >> 1;
    i64 res = 0;
    if (L <= mid) {
        res += query(t[p].ch[0], l, mid, L, R, T);
    }
    if (R >= mid + 1) {
        res += query(t[p].ch[1], mid + 1, r, L, R, T);
    }
    return res;
}

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

    int n, m;

    while(std::cin >> n >> m) {
        tot = 0;

        int cur = 0;
        std::vector<int>version {0};
        build(version[0], 1, n);

        while (m --) {
            char o;
```

```
            std::cin >> o;

            if (o == 'Q') {
                int l, r;
                std::cin >> l >> r;
                std::cout << query(version[cur], 1, n, l, r, 0) << '\n';
            } else if (o == 'H') {
                int l, r, v;
                std::cin >> l >> r >> v;
                std::cout << query(version[v], 1, n, l, r, 0) << '\n';
            } else if (o == 'C') {
                int l, r, d;
                std::cin >> l >> r >> d;
                cur += 1;
                version.push_back(0);
                apply(version[cur - 1], version[cur], 1, n, l, r, d);
            } else if (o == 'B') {
                int v;
                std::cin >> v;
                cur = v;
            } else {
                assert(false);
            }
        }

        std::cout << '\n';
    }
    return 0;
}
```

在树上挂一棵主席树，动态开点二分

```
#include<bits/stdc++.h>
using u64 = unsigned long long;

constexpr int Kp = 10;
constexpr int Kn = 1E5;
static constexpr u64 Mod = (1ull << 61) - 1;

static constexpr u64 add(u64 a, u64 b) {
    u64 c = a + b;
    if (c >= Mod) {
        c -= Mod;
    }
    return c;
}

static constexpr u64 mul(u64 a, u64 b) {
    __uint128_t c = static_cast<__uint128_t>(a) * b;
    return add(c >> 61, c & Mod);
}

int tot = 0;
struct Node {
    int lsh = 0, rsh = 0;
    u64 val = 0;
} t[Kn * 20];
void add(int& c, int p, int l, int r, int x, u64 v) {
    t[c = ++tot] = t[p];
    t[c].val = add(t[c].val, v);
```

```
    if (l == r) {
        return ;
    }
    int mid = (l + r) >> 1;
    if (x <= mid) {
        add(t[c].lsh, t[p].lsh, l, mid, x, v);
    } else {
        add(t[c].rsh, t[p].rsh, mid + 1, r, x, v);
    }
}

u64 salt(int u, int v, int lca, int par) {
    u64 r = 0;
    r = add(r, t[u].val);
    r = add(r, t[v].val);
    r = add(r, Mod - t[lca].val);
    r = add(r, Mod - t[par].val);
    return r;
}

std::vector<int>query(int u1, int v1, int lca1, int par1, int u2, int v2, int
lca2, int par2, int l, int r) {
    std::vector<int>lo;
    if (salt(u1, v1, lca1, par1) == salt(u2, v2, lca2, par2)) {
        return {};
    }
    if (l == r) {
        lo.push_back(l);
        return lo;
    }
    int mid = (l + r) >> 1;
    lo = query(t[u1].lsh, t[v1].lsh, t[lca1].lsh, t[par1].lsh, t[u2].lsh,
t[v2].lsh, t[lca2].lsh, t[par2].lsh, l, mid);
    if (std::ssize(lo) < Kp) {
        auto hi = query(t[u1].rsh, t[v1].rsh, t[lca1].rsh, t[par1].rsh,
t[u2].rsh, t[v2].rsh, t[lca2].rsh, t[par2].rsh, mid + 1, r);
        for (const auto & c : hi) {
            lo.push_back(c);
        }
    }
    return lo;
}

auto main() ->int32_t {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n;
    std::cin >> n;

    std::vector<int>a(n + 1);
    for (int i = 1; i <= n; i += 1) {
        std::cin >> a[i];
    }

    std::vector<std::vector<int>>adj(n + 1);
    for (int i = 1; i <= n - 1; i += 1) {
        int u, v;
        std::cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
```

```
    }

    const u64 ub = 7894655413;
    std::vector<u64>pw(Kn + 1);
    pw[0] = 1;
    for (int i = 1; i <= Kn; i += 1) {
        pw[i] = mul(pw[i - 1], ub);
    }

    constexpr int Kl = 20;
    std::vector<int>d(n + 1), f(n + 1);
    std::vector<std::array<int, Kl>>par(n + 1);
    auto dfs = [&](this auto && dfs, int u) ->void {
        d[u] = d[par[u][0]] + 1;
        add(f[u], f[par[u][0]], 1, Kn, a[u], pw[a[u]]);
        for (const auto & v : adj[u]) {
            if (v == par[u][0]) {
                continue;
            }
            par[v][0] = u;
            for (int j = 1; j < Kl; j += 1) {
                par[v][j] = par[par[v][j - 1]][j - 1];
            }
            dfs(v);
        }
    };
    dfs(1);

    auto lca = [&](int u, int v) {
        if (d[u] < d[v]) {
            std::swap(u, v);
        }
        for (int y = Kl - 1; y >= 0; y -= 1) {
            if (d[par[u][y]] >= d[v]) {
                u = par[u][y];
            }
        }
        if (u == v) {
            return u;
        }
        for (int y = Kl - 1; y >= 0; y -= 1) {
            if (par[u][y] != par[v][y]) {
                u = par[u][y];
                v = par[v][y];
            }
        }
        return par[u][0];
    };

    int q;
    std::cin >> q;
    while (q--) {
        int u1, v1, u2, v2, k;
        std::cin >> u1 >> v1 >> u2 >> v2 >> k;
        int l1 = lca(u1, v1), l2 = lca(u2, v2);
        auto res =  query(f[u1], f[v1], f[l1], f[par[l1][0]], f[u2], f[v2],
f[l2], f[par[l2][0]], 1, Kn);
        std::cout << std::min(int(res.size()), k) << ' ';
        for (int i = 0; i < std::min(int(res.size()), k); i += 1) {
            std::cout << res[i] << ' ';
```

```
        }
        std::cout << '\n';
    }
    return 0;
}
```

Merge

遇到卡空间的情况，应该在merge完之后在add，否则垃圾节点回收就没有意义了

```
constexpr int KN = 1 << 22;
struct Node {
    Node *lsh = nullptr, *rsh = nullptr;
    int cnt = 0;
    int val = 0;
};

std::vector<Node *>pool;
Node* newNode() {
    if (not pool.empty()) {
        auto res = pool.back();
        pool.pop_back();
        return res;
    }
    return new Node;
}

void push(Node*& t) {
    if (t->lsh == nullptr and t->rsh == nullptr) {
        return ;
    }
    if (t->lsh != nullptr and t->rsh != nullptr) {
        t->cnt = std::max(t->lsh->cnt, t->rsh->cnt);
        if (t->cnt == t->rsh->cnt) {
            t->val = t->rsh->val;
        }
        if (t->cnt == t->lsh->cnt) {
            t->val = t->lsh->val;
        }
    } else if (t->lsh != nullptr) {
        t->cnt = t->lsh->cnt;
        t->val = t->lsh->val;
    } else {
        t->cnt = t->rsh->cnt;
        t->val = t->rsh->val;
    }
}

void add(Node*& t, int l, int r, int x, int v) {
    if (t == nullptr) {
        t = newNode();
    }
    if (l == r) {
        t->cnt += v;
        t->val = x;
        return ;
    }
    int mid = l + r >> 1;
    if (x <= mid) {
```

```
        add(t->lsh, l, mid, x, v);
    } else {
        add(t->rsh, mid + 1, r, x, v);
    }
    push(t);
}

Node* merge(Node* a, Node*& b, int l, int r) {
    if (a == nullptr) {
        return b;
    }
    if (b == nullptr) {
        return a;
    }
    if (l == r) {
        a->cnt += b->cnt;
        b->lsh = b->rsh = nullptr;
        b->cnt = b->val = 0;
        pool.push_back(b);
        return a;
    }
    int mid = l + r >> 1;
    a->lsh = merge(a->lsh, b->lsh, l, mid);
    a->rsh = merge(a->rsh, b->rsh, mid + 1, r);
    push(a);
    b->lsh = b->rsh = nullptr;
    b->cnt = b->val = 0;
    pool.push_back(b);
    return a;
}
```

ZKW

```
template <class T, auto f, auto e>
struct SegmentTree {
  int n;
  vector<T> s;
  SegmentTree(int n) : n(n), s(n * 2, e()) {}
  void set(int i, T v) {
    for (s[i += n] = v; i /= 2;) s[i] = f(s[i * 2], s[i * 2 + 1]);
  }
  /// Returns the product of elements in [l, r).
  T product(int l, int r) {
    T rl = e(), rr = e();
    for (l += n, r += n; l != r; l /= 2, r /= 2) {
      if (l % 2) rl = f(rl, s[l++]);
      if (r % 2) rr = f(s[r -= 1], rr);
    }
    return f(rl, rr);
  }
};
```

SparseTable

```
template <class T, auto f>
struct SparseTable {
    std::vector<std::vector<T>>jump;
    SparseTable() = default;
```

```
    SparseTable(const std::vector<T>& a) {
        int n = a.size() - 1;
        int m = std::__lg(n);
        jump.assign(m + 1, std::vector<T>(n + 1));
        std::copy(a.begin(), a.end(), jump[0].begin());
        for (int j = 1; j <= m; j += 1) {
            for (int i = 1; i + (1 << (j - 1)) - 1 <= n; i += 1) {
                jump[j][i] = f(jump[j - 1][i], jump[j - 1][i + (1 << (j -
1))]);
            }
        }
    }
    constexpr T rangeQuery(int l, int r) const {
        assert(l <= r);
        int k = std::__lg(r - l + 1);
        return f(jump[k][l], jump[k][r - (1 << k) + 1]);
    }
};
// compression
template <class T, class Cmp = std::less<T>>
struct RMQ {
    const Cmp cmp = Cmp();
    static constexpr unsigned B = 64;
    using u64 = unsigned long long;

    const int n;
    std::vector<std::vector<T>> a;
    std::vector<T> pre, suf, ini;
    std::vector<u64> stk;

    RMQ(const std::vector<T> &v)
        : n{v.size()}, pre{v}, suf{v}, ini{v}, stk(n) {
        if (n <= 0) {
            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;
            }
        }
    }
    // [l, r)
    T operator()(int l, int r) const {
        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];
        }
    }
};
```

Fenwick

```
template<typename T>
struct Fenwick {
    int n;
    std::vector<T>a;
    Fenwick() {}
    Fenwick(int n) {
        this->n = n;
        a.assign(n + 1, T());
    }
    void add(int p, const T& x) {
        for (int i = p; i <= n; i += i & -i) {
            a[i] += x;
        }
    }
    T sum(int p) {
        T ans = T();
        for (int i = p; i > 0; i -= i & -i) {
            ans += a[i];
        }
        return ans;
    }
    T getRange(int l, int r) {
        return sum(r) - sum(l - 1);
    }
    int select(int k) {
        int x = 0;
```

```
        T cur = T();
        for (int i = std::__lg(n); ~i; i -= 1) {
            x += 1 << i;
            if (x >= n or cur + a[x] >= k) {
                x -= 1 << i;
            } else {
                cur = cur + a[x];
            }
        }
        return x + 1;
    }
};
```

DSU

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

    DSU() {}
    DSU(int n) {
        par.resize(n + 1);
        std::iota(par.begin() + 1, par.end(), 1);
        siz.assign(n + 1, 1);
    }

    int find(int x) {
        while (x != par[x]) {
            x = par[x] = par[par[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;
        }
        if (siz[x] < siz[y]) {
            std::swap(x, y);
        }
        siz[x] += siz[y];
        par[y] = x;
        return true;
    }

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

CartesianTree

```
int top = 0;
std::vector<int> stk(N + 1), ls(N + 1), rs(N + 1);
for (int i = 1; i <= N; i += 1) {
```

```
    int k = top;
    while (k >= 1 && A[i] < A[stk[k]]) {
        k -= 1;
    }
    if (k > 0) {
        rs[stk[k]] = i;
    }
    if (k < top) {
        ls[i] = stk[k + 1];
    }
    stk[top = (k += 1)] = i;
}

i64 res = 0;
auto dfs = [&](auto &&dfs, int p, int l, int r) {
    if (p == 0) {
        return;
    }
    if (l == r) {
        res += (A[p] + B[p] <= S);
        return;
    }
    dfs(dfs, ls[p], l, p - 1);
    dfs(dfs, rs[p], p + 1, r);
    S -= A[p];
    if (p - l <= r - p) {
        for (int lo = l; lo <= p; lo += 1) {
            int fx = p, fy = r;
            while (fx <= fy) {
                int mid = (fx + fy) >> 1;
                if (sum[mid] - sum[lo - 1] <= S) {
                    fx = mid + 1;
                } else {
                    fy = mid - 1;
                }
            }
            int hi = fx - 1;
            res += hi - p + 1;
        }
    } else {
        for (int hi = p; hi <= r; hi += 1) {
            int fx = l, fy = p;
            while (fx <= fy) {
                int mid = (fx + fy) >> 1;
                if (sum[hi] - sum[mid - 1] <= S) {
                    fy = mid - 1;
                } else {
                    fx = mid + 1;
                }
            }
            int lo = fy + 1;
            res += p - lo + 1;
        }
    }
    S += A[p];
};
dfs(dfs, stk[1], 1, N);
```

RevocableDSU

```cpp
struct RevocableDSU {
    std::vector<int>par, siz;
    std::vector<std::pair<int, int>>stk;

    RevocableDSU() = default;
    RevocableDSU(int n) {
        par.resize(n + 1);
        siz.assign(n + 1, 1);
        std::iota(par.begin(), par.end(), 0);
        stk.clear();
    }

    int find(int x) {
        while (x != par[x]) {
            x = par[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;
        }
        if (siz[x] < siz[y]) {
            std::swap(x, y);
        }
        siz[x] += siz[y];
        par[y] = x;
        stk.emplace_back(x, y);
        return true;
    }

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

    int version() {
        return stk.size();
    }

    void rollback(int v) {
        while (stk.size() > v) {
            auto [x, y] = stk.back();
            stk.pop_back();
            siz[x] -= siz[y];
            par[y] = y;
        }
    }
};
```

可撤销并查集维护线段树分治：常用于维护不同颜色块，可以理解为离线版本的LCT

根据颜色区间的不同，有时候不需要显式的进行线段树分治，根据区间直接fen'zhi

```cpp
auto main() ->int {
    int n;
```

```
    std::cin >> n;
    std::vector<std::vector<int>>t(4 << std::__lg(n));
    auto add = [&](auto && self, int p, int l, int r, int L, int R, int k) -
>void {
        if (l > R || r < L) {
            return ;
        }
        if (L <= l && r <= R) {
            t[p].push_back(k);
            return ;
        }
        int mid = (l + r) >> 1;
        self(self, p << 1, l, mid, L, R, k);
        self(self, p << 1 | 1, mid + 1, r, L, R, k);
    };
    std::vector<std::vector<std::pair<int, int>>>h(n + 1);
    std::vector<std::pair<int, int>>edges(n);
    for (int i = 1; i <= n - 1; i += 1) {
        auto& [u, v] = edges[i];
        int c;
        std::cin >> u >> v >> c;
        h[c].push_back({u, v});
        if (c > 1) {
            add(add, 1, 1, n, 1, c - 1, i);
        }
        if (c < n) {
            add(add, 1, 1, n, c + 1, n, i);
        }
    }
    i64 res = 0;
    RevocableDSU dsu(n);
    auto dfs = [&](auto && self, int p, int l, int r) ->void {
        int cur = dsu.version();
        for (const auto & k : t[p]) {
            const auto& [u, v] = edges[k];
            dsu.merge(u, v);
        }
        if (l == r) {
            for (const auto & [u, v] : h[l]) {
                int x = dsu.size(u), y = dsu.size(v);
                res += 1LL * x * y;
            }
        } else {
            int mid = (l + r) >> 1;
            self(self, p << 1, l, mid);
            self(self, p << 1 | 1, mid + 1, r);
        }
        dsu.rollback(cur);
    };
    dfs(dfs, 1, 1, n);
}
```

HLD

树剖换根考虑三种情况：

当前节点为根；

根在当前节点的子树里面，真正的子树是除根到当前节点路径上的节点外的所有节点；

否则和正常情况无异;

```
if (x == root) {
    std::cout << seg.rangeQuery(1, n).val << "\n";
} else if (hld.lca(x, root) == x) {
    int s = hld.jump(root, hld.dep[root] - hld.dep[x] - 1);
    std::cout << (std::min(seg.rangeQuery(1, hld.dfn[s] - 1).val,
seg.rangeQuery(hld.dfn[s] + hld.siz[s], n).val)) << "\n";
} else {
    std::cout << (seg.rangeQuery(hld.dfn[x], hld.dfn[x] + hld.siz[x] - 1).val)
<< "\n";
}
```

```
struct HLD {
    int n;
    std::vector<std::vector<int>>adj;
    std::vector<int>dfn, siz, par, son, top, dep, seq;
    int cur;
    HLD() {}

    HLD(int n) {
        this->n = n;
        adj.assign(n + 1, std::vector<int>());
        dfn.resize(n + 1), par.resize(n + 1);
        son.resize(n + 1), siz.resize(n + 1);
        dep.resize(n + 1), top.resize(n + 1);
        seq.resize(n + 1);
        cur = 0;
    }

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

    void dfs(int u) {
        siz[u] += 1;
        dep[u] = dep[par[u]] + 1;
        for (const auto & v : adj[u]) {
            if (v == par[u]) {
                continue;
            }
            par[v] = u;
            dfs(v);
            siz[u] += siz[v];
            if (siz[v] > siz[son[u]]) {
                son[u] = v;
            }
        }
    }

    void dfs(int u, int h) {
        dfn[u] = ++cur;
        seq[cur] = u;
        top[u] = h;
        if (son[u]) {
            dfs(son[u], h);
        }
        for (const auto & v : adj[u]) {
            if (v == son[u] or v == par[u]) {
```

```
                continue;
            }
            dfs(v, v);
        }
    }

    void work(int s = 1) {
        dfs(s);
        dfs(s, s);
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] < dep[top[v]]) {
                std::swap(u, v);
            }
            u = par[top[u]];
        }
        return dep[u] < dep[v] ? u : v;
    }
    int lca(int u, int v, int root) {
        return lca(u, v) ^ lca(u, root) ^ lca(v, root);
    }
    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 = par[top[u]];
        }
        return seq[dfn[u] + d - dep[u]];
    }
    int left(int u) {
        return dfn[u];
    }
    int right(int u) {
        return dfn[u] + siz[u] - 1;
    }
    bool isAncestor(int u, int v) {
        return dfn[u] <= dfn[v] and dfn[v] < dfn[u] + siz[u];
    }
};
auto dfs = [&](this auto && dfs, int u) ->void {
    for (const auto & v : hld.adj[u]) {
        if (v == hld.par[u]) {
            continue;
        }
        dfs(v);
    }
    for (auto it = s[a[u]].lower_bound(hld.left(u)); it != s[a[u]].end() and
(*it) <= hld.right(u);) {
        int y = hld.seq[*it];
        it = s[a[u]].erase(it);
        seg.rangeApply(hld.left(y), hld.right(y), { -1});
    }
    seg.rangeApply(hld.left(u), hld.right(u), {1});
    s[a[u]].insert(hld.left(u));
```

```
    int res = 1;
    for (const auto & v : hld.adj[u]) {
        if (v == hld.par[u]) {
            continue;
        }
        int j = seg.rangeQuery(hld.left(v), hld.right(v)).val;
        chmax(ans, i64(res) * j);
        chmax(res, j);
    }
};
LazySegmentTree<Info, Tag>seg(n);
for (int i = 1; i <= n; i += 1) {
    seg.modify(i, {1, w[hld.seq[i]]});
}

int root = 1;

while (q--) {
    int o;
    std::cin >> o;
    if (o == 1) {
        std::cin >> root;
    } else if (o == 2) {
        int x, y, v;
        std::cin >> x >> y >> v;
        int lca = hld.lca(x, y, root);
        if (lca == root) {
            seg.rangeApply(1, n, {v});
        } else if (hld.lca(lca, root) == lca) {
            int s = hld.jump(root, hld.dep[root] - hld.dep[lca] - 1);
            seg.rangeApply(1, n, {v});
            seg.rangeApply(hld.dfn[s], hld.dfn[s] + hld.siz[s] - 1, { -v});
        } else {
            seg.rangeApply(hld.dfn[lca], hld.dfn[lca] + hld.siz[lca] - 1, {v});
        }
    } else {
        int x;
        std::cin >> x;
        if (x == root) {
            std::cout << seg.rangeQuery(1, n).val << "\n";
        } else if (hld.lca(x, root) == x) {
            int s = hld.jump(root, hld.dep[root] - hld.dep[x] - 1);
            std::cout << (seg.rangeQuery(1, n).val - seg.rangeQuery(hld.dfn[s],
hld.dfn[s] + hld.siz[s] - 1).val) << "\n";
        } else {
            std::cout << seg.rangeQuery(hld.dfn[x], hld.dfn[x] + hld.siz[x] -
1).val << "\n";
        }
    }
}
```

DFN

```
auto main() ->int32_t {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    std::cout << std::fixed << std::setprecision(13);

    int n, m;
    std::cin >> n >> m;
```

```cpp
    std::vector<std::array<int, 3>>edges(m);
    for (auto& [u, v, w] : edges) {
        std::cin >> u >> v >> w;
    }

    std::vector<int>ord(m);
    std::ranges::iota(ord, 0);
    std::ranges::sort(ord, {}, [&](const auto & i) {
        return edges[i][2];
    });

    i64 sum = 0;
    HLD hld(n);
    DSU dsu(n);
    std::vector<std::vector<std::array<int, 2>>>adj(n + 1);
    for (int i = 0; i < m; i += 1) {
        const auto& [u, v, w] = edges[ord[i]];
        if (dsu.merge(u, v)) {
            sum += w;
            hld.addEdge(u, v);
            adj[u].push_back({v, w});
            adj[v].push_back({u, w});
        }
    }

    hld.work();
    std::vector<int>link(n + 1);
    auto dfs = [&](this auto && dfs, int u, int par)  -> void{
        for (const auto & [v, w] : adj[u]) {
            if (v == par) {
                continue;
            }
            link[hld.dfn[v]] = w;
            dfs(v, u);
        }
    };
    dfs(1, 0);

    auto Max = SparseTable<int, std::greater<int>>(link, 0);
    const auto& top = hld.top, par = hld.par, dep = hld.dep;
    auto getRange = [&](int u, int v) {
        int res = 0;
        while (top[u] != top[v]) {
            if (dep[top[u]] < dep[top[v]]) {
                std::swap(u, v);
            }
            chmax(res, Max.getRange(hld.dfn[top[u]], hld.dfn[u]));
            u = par[top[u]];
        }
        if (dep[u] > dep[v]) {
            std::swap(u, v);
        }
        if (dep[u] < dep[v]) {
            chmax(res, Max.getRange(hld.dfn[u] + 1, hld.dfn[v]));
        }
        return res;
    };

    for (const auto & [u, v, w] : edges) {
        std::cout << sum + w - getRange(u, v) << '\n';
```

```
    }
    return 0;
}
```

LowestCommonAnc

注意常数

```
struct LowestCommonAncestor {
    int n, low, cnt;
    std::vector<int>seq, dfn, lst, ord;
    std::vector<std::vector<int>>adj, jump;

    LowestCommonAncestor(const int& n) {
        this->n = n;
        low = cnt = 0;
        adj.assign(n + 1, {});
        seq.assign(n + 1, 0);
        dfn.assign(n + 1, 0);
        lst.assign(n + 1, 0);
        ord.assign(2 * n + 1, 0);
    }

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

    void dfs(int u, int par) {
        dfn[u] = ++low;
        seq[low] = u;
        ord[++cnt] = u;
        lst[u] = cnt;
        for (const auto & v : adj[u]) {
            if (v == par) {
                continue;
            }
            dfs(v, u);
            ord[++cnt] = u;
            lst[u] = cnt;
        }
    }

    void work() {
        dfs(1, 0);
        int lgn = std::__lg(cnt);
        jump.assign(lgn + 2, std::vector<int>(cnt + 1));
        for (int i = 1; i <= cnt; i += 1) {
            jump[0][i] = dfn[ord[i]];
        }
        for (int j = 1; j <= lgn + 1; j += 1) {
            for (int i = 1; i + (1 << j) - 1 <= cnt; i += 1) {
                jump[j][i] = std::min(jump[j - 1][i], jump[j - 1][i + (1 << (j
- 1))]);
            }
        }
    }

    constexpr int operator()(int l, int r)const {
        l = lst[l], r = lst[r];
```

```
        if (l > r) {
            std::swap(l, r);
        }
        int k = std::__lg(r - l + 1);
        return seq[std::min(jump[k][l], jump[k][r - (1 << k) + 1])];
    }
};

void dfs(int u) {
    dep[u] = dep[par[u][0]] + 1;
    for (int k = 1; k < Kn; k += 1) {
        par[u][k] = par[par[u][k - 1]][k - 1];
    }
    for (const auto & [v, w] : adj[u]) {
        if (v == par[u][0]) {
            continue;
        }
        sum[v] = sum[u] + w;
        par[v][0] = u;
        dfs(v);
    }
}

int lca(int u, int v) {
    if (dep[u] < dep[v]) {
        std::swap(u, v);
    }
    for (int k = Kn - 1; k >= 0; k -= 1) {
        if (dep[par[u][k]] >= dep[v]) {
            u = par[u][k];
        }
    }

    if (u == v) {
        return u;
    }

    for (int k = Kn - 1; k >= 0; k -= 1) {
        if (par[u][k] != par[v][k]) {
            u = par[u][k];
            v = par[v][k];
        }
    }
    return par[u][0];
}
```

DSU On Tree

```
auto add = [&](int p) {
    rec[cnt[c[p]]] -= c[p];
    cnt[c[p]] += 1;
    rec[cnt[c[p]]] += c[p];
};
auto del = [&](int p) {
    rec[cnt[c[p]]] -= c[p];
    if (not rec[cnt[c[p]]]) {
        rec.erase(cnt[c[p]]);
    }
    cnt[c[p]] -= 1;
    rec[cnt[c[p]]] += c[p];
```

```
};
auto cal = [&]() {
    return rec.rbegin()->second;
};
auto dfs = [&](this auto && self, int u, int top = 0)->void{
    for (const auto & v : hld.adj[u]) {
        if (v == hld.par[u] or v == hld.son[u]) {
            continue;
        }
        self(v);
    }
    if (hld.son[u]) {
        self(hld.son[u], 1);
    }
    for (const auto & v : hld.adj[u]) {
        if (v == hld.par[u] or v == hld.son[u]) {
            continue;
        }
        for (int j = hld.left(v); j <= hld.right(v); j += 1) {
            add(hld.seq[j]);
        }
    }
    add(u);
    ans[u] = cal();
    if (not top) {
        for (int j = hld.left(u); j <= hld.right(u); j += 1) {
            del(hld.seq[j]);
        }
    }
};
```

LinearBasis

```
template<class T, const int M>
struct Basis {
    int zero;
    int cnt;
    std::array<T, M>a;
    std::array<T, M>t;
    Basis() {
        a.fill(0);
        t.fill(-1);
        cnt = 0;
        zero = 0;
    }
    bool add(T x, T y = std::numeric_limits<T>::max()) {
        for (int j = M - 1; j >= 0; j -= 1) {
            if (x >> j & 1) {
                if (not a[j]) {
                    a[j] = x;
                    t[j] = y;
                    cnt += 1;
                    return true;
                } else if (t[j] < y) {
                    std::swap(t[j], y);
                    std::swap(a[j], x);
                }
                x ^= a[j];
            }
        }
```

```
        zero = 1;
        return false;
    }
    T min(T x = std::numeric_limits<T>::max(), T l = {0}) {
        for (int j = M - 1; j >= 0; j -= 1) {
            if (a[j] and t[j] >= l) {
                x = std::min(x, x ^ a[j]);
            }
        }
        return x;
    }
    T max(T x = {0}, T l = {0}) {
        for (int j = M - 1; j >= 0; j -= 1) {
            if (a[j] and t[j] >= l) {
                x = std::max(x, x ^ a[j]);
            }
        }
        return x;
    }
    bool have(T x, T y = {0}) {
        for (int j = M - 1; j >= 0; j -= 1) {
            if (x >> j & 1 and t[j] >= y) {
                x ^= a[j];
            }
        }
        return x == 0;
    }
    T select(T k, T y = {0}) {
        auto b = a;
        k -= zero;
        if (k >= 1ull << cnt) {
            return -1;
        }
        std::vector<T>d;
        for (int i = 0; i <= M - 1; i += 1) {
            for (int j = i - 1; j >= 0; j -= 1) {
                if (b[i] >> j & 1) {
                    b[i] ^= b[j];
                }
            }
            if (b[i])d.push_back(b[i]);
        }
        T res = {0};
        for (int i = 0; i < d.size(); i += 1) {
            if (k >> i & 1) {
                res ^= d[i];
            }
        }
        return res;
    }
};
```

Mo

```
int B;
struct Mo {
    int l = 0, r = 0, e = 0;
    friend bool operator<(const Mo& lsh, const Mo& rsh) {
        if ((lsh.l / B + 1) != (rsh.l / B + 1)) {
            return (lsh.l / B + 1) < (rsh.l / B + 1);
```

```
        } else if ((lsh.l / B + 1) & 1) {
            return lsh.r < rsh.r;
        }
        return lsh.r > rsh.r;
    }
};
auto add = [&](int p) {
    if (not cnt[c[p]]) {
        res += 1;
    }
    cnt[c[p]] += 1;
};
auto del = [&](int p) {
    cnt[c[p]] -= 1;
    if (not cnt[c[p]]) {
        res -= 1;
    }
};
int l = 1, r = 0;
for (int i = 1; i <= q; i += 1) {
    while (l < qry[i].l) {
        del(l++);
    }
    while (l > qry[i].l) {
        add(--l);
    }
    while (r < qry[i].r) {
        add(++r);
    }
    while (r > qry[i].r) {
        del(r--);
    }
    f[qry[i].e] = res;
}
```

LCT

```
access 从当前根节点到这个点打通一条链
splay 将这个点旋转到所在链的根
```

```
struct LinkCutTree {
    struct Node {
        int rev = 0;
        int siz = 1;
        int par = 0;
        int cap = 0;
        std::array<int, 2> ch{};
    };
    std::vector<Node> t;
    LinkCutTree(const int& n) {
        t.assign(n + 1, Node());
        t[0].siz = 0;
    }
    void pull(int p) {
        t[p].siz = t[t[p].ch[0]].siz + t[t[p].ch[1]].siz + 1 + t[p].cap;
    }
    void flip(int p) {
        std::swap(t[p].ch[0], t[p].ch[1]);
        t[p].rev ^= 1;
```

```
    }
    void push(int p) {
        if (t[p].rev) {
            if (t[p].ch[0])flip(t[p].ch[0]);
            if (t[p].ch[1])flip(t[p].ch[1]);
            t[p].rev = 0;
        }
    }
    bool isroot(int p) {
        return (p != t[t[p].par].ch[0] and p != t[t[p].par].ch[1]);
    }
    int pos(int p) {
        return t[t[p].par].ch[1] == p;
    }
    void rotate(int p) {
        int x = t[p].par, y = t[t[p].par].par, k = pos(p), r = t[p].ch[!k];
        if (not isroot(x)) {
            t[y].ch[pos(x)] = p;
        }
        t[p].ch[!k] = x, t[x].ch[k] = r;
        if (r) {
            t[r].par = x;
        }
        t[x].par = p, t[p].par = y;
        pull(x);
        pull(p);
    }
    void pushAll(int p) {
        if (not isroot(p)) {
            pushAll(t[p].par);
        }
        push(p);
    }
    void splay(int p) {
        pushAll(p);
        while (not isroot(p)) {
            if (not isroot(t[p].par)) {
                rotate((pos(t[p].par) == pos(p)) ? t[p].par : p);
            }
            rotate(p);
        }
        pull(p);
    }
    int access(int p) {
        int q = 0;
        for (; p; q = p, p = t[p].par) {
            splay(p);
            t[p].cap += t[t[p].ch[1]].siz - t[q].siz;
            t[p].ch[1] = q;
            pull(p);
        }
        return q;
    }
    void makeroot(int p) {
        access(p);
        splay(p);
        flip(p);
    }
    int findroot(int p) {
        access(p);
```

```
        splay(p);
        while (t[p].ch[0]) {
            push(p);
            p = t[p].ch[0];
        }
        splay(p);
        return p;
    }
    void link(int u, int v) {
        makeroot(u);
        if (findroot(v) != u) {
            t[u].par = v;
            t[v].cap += t[u].siz;
            pull(v);
        }
    }
    void cut(int u, int v) {
        makeroot(u);
        if (findroot(v) == u and t[v].par == u) {
            t[v].par = t[u].ch[1] = 0;
            pull(u);
        }
    }
};
```

Gravity

```
int t = 0, all = n;
max[t] = 1E9;
std::vector<int>max(n + 1), siz(n + 1), del(n + 1);
auto find = [&](this auto && find, int u, int par) ->void {
    max[u] = 0, siz[u] = 1;
    for (const auto & v : adj[u]) {
        if (v == par or del[v])continue;
        find(v, u);
        siz[u] += siz[v];
        chmax(max[u], siz[v]);
    }
    chmax(max[u], all - siz[u]);
    if (max[u] <= max[t]) {
        t = u;
    }
};
auto solve = [&](this auto && solve, int u) ->void{
    del[u] = 1;
    cal(u);
    for (const auto & v : adj[u]) {
        if (del[v]) {
            continue;
        }
        max[t = 0] = 1E9;
        all = siz[v];
        find(v, u);
        solve(t);
    }
};
find(1, 0);
solve(t);
```

VitualTree

```
std::ranges::sort(colr[c], {}, [&](const auto & u) {
    return hld.dfn[u];
});
int n = colr[c].size();
for (int i = 1; i < n; i += 1) {
    colr[c].push_back(hld.lca(colr[c][i - 1], colr[c][i]));
}
colr[c].push_back(1);
std::ranges::sort(colr[c], {}, [&](const auto & u) {
    return hld.dfn[u];
});
colr[c].erase(std::unique(colr[c].begin(), colr[c].end()), colr[c].end());
for (int i = 1; i < colr[c].size(); i += 1) {
    int l = hld.lca(colr[c][i - 1], colr[c][i]);
    adj[l].push_back(colr[c][i]);
}
```

FHQ-Treap

```
template <class T>
struct Treap {
    std::mt19937 rng;
    struct Node {
        int lsh = 0, rsh = 0;
        T val = T();
        int key = 0, siz = 0, rev = 0;
    };
    int root, t1, t2, t3;
    std::vector<Node>info;
    Treap() : info(1), rng(220725) , root(0), t1(0), t2(0), t3(0) {}
    int newNode(const T& val) {
        info.emplace_back(0, 0, val, int(rng()), 1, 0);
        return info.size() - 1;
    }
    void push(int p) {
        info[p].siz = info[info[p].lsh].siz + info[info[p].rsh].siz + 1;
    }
    void pull(int p) {
        if (info[p].rev) {
            std::swap(info[p].lsh, info[p].rsh);
            if (info[p].lsh)info[info[p].lsh].rev ^= 1;
            if (info[p].rsh)info[info[p].rsh].rev ^= 1;
            info[p].rev = 0;
        }
    }
    void split_by_rank(int u, int k, int& x, int & y) {
        if (not u)return x = y = 0, void();
        pull(u);
        int cur = info[info[u].lsh].siz + 1;
        if (cur == k) {
            x = u, y = info[u].rsh;
            info[u].rsh = 0;
        } else if (cur > k) {
            y = u;
            split_by_rank(info[u].lsh, k, x, info[u].lsh);
        } else {
            x = u;
```

```
            split_by_rank(info[u].rsh, k - cur, info[u].rsh, y);
        }
        push(u);
    }
    void split_by_val(int u, const T& val, int& x, int& y) {
        if (not u)return x = y = 0, void();
        if (info[u].val > val) {
            y = u;
            split_by_val(info[u].lsh, val, x, info[u].lsh);
        } else {
            x = u;
            split_by_val(info[u].rsh, val, info[u].rsh, y);
        }
        push(u);
    }
    int merge(int x, int y) {
        if ((not x) or (not y))return x + y;
        pull(x), pull(y);
        if (info[x].key > info[y].key) {
            info[x].rsh = merge(info[x].rsh, y);
            push(x);
            return x;
        } else {
            info[y].lsh = merge(x, info[y].lsh);
            push(y);
            return y;
        }
    }
    void insert_rank(int x, const T& val) {
        split_by_rank(root, x - 1, t1, t2);
        root = merge(merge(t1, newNode(val)), t2);
    }
    void insert_val(const T& val) {
        split_by_val(root, val, t1, t2);
        root = merge(merge(t1, newNode(val)), t2);
    }
    void erase_rank(int x) {
        split_by_rank(root, x - 1, t1, t2);
        split_by_rank(t2, x, t2, t3);
        root = merge(t1, t3);
    }
    void erase_val(const T& val) {
        split_by_val(root, val, t1, t2);
        split_by_val(t1, val - 1, t1, t3);
        t3 = merge(info[t3].lsh, info[t3].rsh);
        root = merge(merge(t1, t3), t2);
    }
    int begin(int u) {
        return info[u].lsh == 0 ? u : begin(info[u].lsh);
    }
    int end(int u) {
        return info[u].rsh == u ? 0 : end(info[u].rsh);
    }
    void flip(int l, int r) {
        split_by_rank(root, l - 1, t1, t2);
        split_by_rank(t2, r - l + 1, t2, t3);
        info[t2].rev ^= 1;
        root = merge(merge(t1, t2), t3);
    }
    int erase_begin(int u) {
```

```
        if (not info[u].lsh)return info[u].rsh;
        info[u].lsh = erase_begin(info[u].lsh);
        push(u);
        return u;
    }
    int erase_end(int u) {
        if (not info[u].rsh)return info[u].lsh;
        info[u].rsh = erase_end(info[u].rsh);
        push(u);
        return u;
    }
    int order_of_key(const T& val) {
        split_by_val(root, val - 1, t1, t2);
        int res = info[t1].siz + 1;
        root = merge(t1, t2);
        return res;
    }
    T find_by_order(int k) {
        int u = root;
        while (u) {
            int cur = info[info[u].lsh].siz + 1;
            if (cur == k)break;
            if (cur > k) {
                u = info[u].lsh;
            } else {
                k -= cur;
                u = info[u].rsh;
            }
        }
        return info[u].val;
    }
    T findPref(const T& x) {
        split_by_val(root, x - 1, t1, t2);
        int u = t1;
        while (info[u].rsh) {
            u = info[u].rsh;
        }
        root = merge(t1, t2);
        return info[u].val;
    }
    T findSurf(const T& x) {
        split_by_val(root, x, t1, t2);
        int u = t2;
        while (info[u].lsh) {
            u = info[u].lsh;
        }
        root = merge(t1, t2);
        return info[u].val;
    }
    void travel(int p) {
        pull(p);
        if (info[p].lsh)travel(info[p].lsh);
        std::cout << info[p].val << ' ';
        if (info[p].rsh)travel(info[p].rsh);
    }
};
```

String

Hash

```
static constexpr u64 Mod = (1ull << 61) - 1;
static constexpr u64 add(u64 a, u64 b) {
    u64 c = a + b;
    if (c >= Mod) {
        c -= Mod;
    }
    return c;
}
static constexpr u64 mul(u64 a, u64 b) {
    __uint128_t c = static_cast<__uint128_t>(a) * b;
    return add(c >> 61, c & Mod);
}
constexpr int Kn = 1E6;
u64 pw[Kn + 1];
u64 htt = rand(Mod / 3, Mod / 2);
struct Hash : public std::vector<u64> {
    Hash() = default;
    Hash(const std::string& s) {
        int n = s.size();
        this->resize(n + 1);
        for (int i = 1; i <= n; i += 1) {
            (*this)[i] = add(mul((*this)[i - 1], htt), int(s[i - 1]));
        }
    }
    constexpr u64 rangeQuery(int l, int r) const {
        return add((*this)[r], Mod - mul((*this)[l - 1], pw[r - l + 1]));
    }
};
```

哈希神技

给定一个文本串，要查询一个字符串集合里面的字符串在该模板串里面的出现次数。

在最坏情况下，集合里面的字符串长度互不相同，即 $1+2+3+\cdots+t\le len$，显然 $t\le\sqrt{2len}$。

受制于给定字符串集合里面的字符串总长度。

按集合里面的长度来枚举长度和起点，使用 unordered _map 统计即可，复杂度为 $O(n\sqrt{len})$。

```
int n;
std::cin >> n;
std::map<int, int> cnt;
std::unordered_map<u64, int> rec;
for (int i = 1; i <= n; i += 1) {
    std::string s;
    std::cin >> s;
    int m = s.size();
    u64 h = 0;
    for (int i = 1; i <= m; i += 1) {
        h = add(mul(h, htt), int(s[i - 1]));
    }
    cnt[m] += 1;
    rec[h] += 1;
}
int res = 0;
```

```
std::string s;
std::cin >> s;
int m = s.size();
auto h = Hash(s);
for (const auto &[len, v]: cnt) {
    if (len > m) {
        break;
    }
    for (int r = len; r <= m; r += 1) {
        res += rec[h.rangeQuery(r - len + 1, r)];
    }
}
std::cout << res << '\n';
```

Extend : segplusHash

```
struct Tag {
    int add = -1;
    void apply(const Tag& t) {
        if (t.add == -1) {
            return;
        }
        add = t.add;
    }
};
struct Info {
    int len = 0;
    int pre = -1, suf = -1;
    void apply(const Tag& t) {
        if (t.add == -1) {
            return;
        }
        int v = 1LL * t.add * sPow[len - 1] % MOD;
        pre = suf = v;
    }
};
Info operator+(const Info& a, const Info& b) {
    Info c {};
    c.len = a.len + b.len;
    if (a.pre == -1) {
        return b;
    }
    if (b.suf == -1) {
        return a;
    }
    c.pre = (1LL * a.pre * Pow[b.len] % MOD + b.pre) % MOD;
    c.suf = (1LL * b.suf * Pow[a.len] % MOD + a.suf) % MOD;
    return c;
}
sPow[0] = Pow[0] = 1;
for (int i = 1; i <= N; i += 1) {
    Pow[i] = 1LL * Pow[i - 1] * BASE % MOD;
    sPow[i] = (1LL * sPow[i - 1] + Pow[i]) % MOD;
}
```

AhoCorasick

fail 指向当前节点的最长真后缀。

文本串在节点上打标记，模式串的出现次数为模式串的子树和。

二进制分组建立Ac自动机

强制在线维护一个集合，可以加入或者删除字符串，查询时给出一个文本串，求集合中每个字符串在文本串中的出现次数的总和。

```
// 在结尾打标记
t[p].cnt += 1;
t[u].cnt += t[t[u].link].cnt;
constexpr int Kw = 20;
std::vector<AhoCorasick>ac(Kw);
std::vector<std::vector<int>>pool(Kw);
auto add = [&](int i) {
    std::vector<int>h;
    for (int k = 0; k < Kw; k += 1) {
        if (!pool[k].empty()) {
            h.insert(h.end(), pool[k].begin(), pool[k].end());
            pool[k].clear();
            ac[k] = AhoCorasick();
        } else {
            pool[k] = std::move(h);
            for (const auto & v : pool[k]) {
                ac[k].add(s[v]);
            }
            ac[k].work();
        }
    }
};
//显然这个查询只需要在每个串的endpos上记录,构建Ac自动机的时候顺便维护前缀和即可。
for (int k = 0; k < Kw; k += 1) {
    int p = 1;
    for (const auto & c : s) {
        p = ac[k].next(p, c - 'a');
        res += ac[k].cnt(p);
    }
}
```

```
struct AhoCorasick {
    static constexpr int ALPHABET = 26;
    struct Node {
        int len;
        int link;
        std::array<int, ALPHABET> next;
        Node() : len{0}, link{0}, next{} {}
    };
    std::vector<Node> t;
    AhoCorasick() {
        t.assign(2, Node());
        t[0].next.fill(1);
        t[0].len = -1;
    }
    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }
    int add(const std::string &a) {
        int p = 1;
        for (auto c : a) {
            int x = c - 'a';
            if (t[p].next[x] == 0) {
                t[p].next[x] = newNode();
```

```
                t[t[p].next[x]].len = t[p].len + 1;
            }
            p = t[p].next[x];
        }
        return p;
    }
    void work() {
        std::queue<int> q;
        q.push(1);
        while (!q.empty()) {
            int x = q.front();
            q.pop();
            for (int i = 0; i < ALPHABET; i++) {
                if (t[x].next[i] == 0) {
                    t[x].next[i] = t[t[x].link].next[i];
                } else {
                    t[t[x].next[i]].link = t[t[x].link].next[i];
                    q.push(t[x].next[i]);
                }
            }
        }
    }
    int next(int p, int x) {
        return t[p].next[x];
    }
    int link(int p) {
        return t[p].link;
    }
    int len(int p) {
        return t[p].len;
    }
    int size() {
        return t.size();
    }
};
```

计算每个模式串在文本串中的出现次数

```
int n;
std::cin >> n;
AhoCorasick ac;
std::vector<std::string>t(n);
std::vector<int>end(n);
for (int i = 0; i < n; i += 1) {
    std::cin >> t[i];
    end[i] = ac.add(t[i]);
}
ac.work();
std::string s;
std::cin >> s;
int p = 1;
std::vector<int>f(ac.size());
for (const auto & c : s) {
    p = ac.next(p, c - 'a');
    f[p] += 1;
}
std::vector adj(ac.size(), std::vector<int>());
for (int i = 2; i < ac.size(); i += 1) {
    adj[ac.link(i)].push_back(i);
}
```

```
auto dfs = [&](auto && self, int x)->void{
    for (const auto & y : adj[x]) {
        self(self, y);
        f[x] += f[y];
    }
};
dfs(dfs, 1);
for (int i = 0; i < n; i += 1) {
    std::cout << f[end[i]] << "\n";
}
```

给定 n 个字符串 S_n，有 m 次询问，每次给定一个字符串 T_i。询问 T_i 中有多少 个子串 $T[l,r]$ 存在正整数 j 满足 $T[l,r]=S_j$。

$\sum_l \sum_r [T_{l,r}存在子串等于S_j] l*(n-r+1) = ((r+1)*sumCnt - sumLen)*(n-r+1)$。

枚举所有 T_i 的所有前缀统计即可。

sum 可以在拓扑排序的同时统计。

```
int p = 1;
for (int r = 1; const auto & c : s) {
    p = ac.next(p, c - 'a');
    res = (1LL * res + (1LL * ac.cnt(p) * (r + 1) - 1LL * ac.sum(p) + Mod) %
Mod * (int(s.length()) - r + 1) % Mod) % Mod;
    r += 1;
}
```

SA

```
struct SA {
    int n;
    std::vector<int>sa, rk, lc;
    SA(const std::string& s) {
        n = s.size();
        int m = 128;
        // pay attention to the size
        rk.assign(2 * n + 1, 0);
        sa.assign(2 * n + 1, 0);
        std::vector<int>cnt(m + 1, 0);
        for (int i = 1; i <= n; i += 1) {
            cnt[rk[i] = s[i - 1]] += 1;
        }
        for (int i = 1; i <= m; i += 1) {
            cnt[i] += cnt[i - 1];
        }
        for (int i = n; i >= 1; i -= 1) {
            sa[cnt[rk[i]]--] = i;
        }
        std::vector<int>ord(n + 1);
        for (int w = 1, p = 0; p != n; w <<= 1, m = p) {
            int cur = 0;
            for (int i = n - w + 1; i <= n; i += 1) {
                ord[++cur] = i;
            }
            for (int i = 1; i <= n; i += 1) {
                if (sa[i] > w) {
                    ord[++cur] = sa[i] - w;
                }
            }
```

```
            cnt.assign(m + 1, 0);
            for (int i = 1; i <= n; i += 1) {
                cnt[rk[i]] += 1;
            }
            for (int i = 1; i <= m; i += 1) {
                cnt[i] += cnt[i - 1];
            }
            for (int i = n; i >= 1; i -= 1) {
                sa[cnt[rk[ord[i]]]--] = ord[i];
            }
            p = 0;
            auto ork = rk;
            for (int i = 1; i <= n; i += 1) {
                if (ork[sa[i]] == ork[sa[i - 1]] and ork[sa[i] + w] == ork[sa[i
- 1] + w]) {
                    rk[sa[i]] = p;
                } else {
                    rk[sa[i]] = ++p;
                }
            }
        }
        lc.assign(n + 1, 0);
        for (int i = 1, k = 0; i <= n; i += 1) {
            if (rk[i] == 1) {
                continue;
            }
            if (k) {
                k -= 1;
            }
            while (s[i + k - 1] == s[sa[rk[i] - 1] + k - 1]) {
                k += 1;
            }
            lc[rk[i]] = k;
        }
    }

    std::vector<std::vector<int>>rmq;
    void work() {
        int logn = std::__lg(n);
        rmq.assign(logn + 1, std::vector<int>(n + 1));
        std::copy(lc.begin(), lc.end(), rmq[0].begin());
        for (int j = 1; j <= logn; j += 1) {
            for (int i = 1; i + (1 << (j - 1)) - 1 <= n; i += 1) {
                rmq[j][i] = std::min(rmq[j - 1][i], rmq[j - 1][i + (1 << (j -
1))]);
            }
        }
    }
    constexpr int range(int l, int r) {
        int k = r - l + 1;
        k = std::__lg(k);
        return std::min(rmq[k][l], rmq[k][r - (1 << k) + 1]);
    }
    constexpr int lcp(int i, int j) {
        if (i < 1 || i > n || j < 1 || j > n) {
            return 0;
        }
        int u = rk[i], v = rk[j];
        if (u == v) {
            return n - sa[u] + 1;
```

```
        }
        if (u > v) {
            std::swap(u, v);
        }
        return range(u + 1, v);
    }
};
```

求解最长公共子串

```
std::string s, t;
std::cin >> s >> t;
int n = s.size(), m = t.size();
SA sa(std::string() + s + '$' + t);
int ans = 0;
for (int i = 2; i <= sa.n; i += 1) {
    if (sa.sa[i - 1] >= n + 1 and sa.sa[i] <= n or sa.sa[i - 1] <= n and
sa.sa[i] >= n + 1) {
        chmax(ans, sa.lc[i]);
    }
}
```

求区间本质不同的子序列个数(其实是一个经典的动态规划模型)

```
Z cal(const std::string& s) {
    int n = s.size();
    std::vector<Z>dp(26);
    for (int i = 0; i < n; i += 1) {
        Z sum = 1;
        std::vector<Z>ndp(26);
        for (int j = 0; j <= 25; j += 1) {
            sum += (ndp[j] = dp[j]);
        }
        ndp[s[i] - 'a'] = sum;
        dp.swap(ndp);
    }
    return std::accumulate(dp.begin(), dp.end(), Z(0));
}
```

SAM

$\text{endpos}(p)$的等价类共用同一个节点，对于同一个等价类里面的字符串分析，假设$|p_1| < |p_2|$，那么p_1是p_2的后缀

因此，在一个等价类中，必然存在且只存在一个最长的串p。

对于p的所有出现，根据位于p前面的那一个字符是什么，可以把原等价类划分为若干个等价类。

根据这样的分析可以得到SAM的空间复杂度是$O(n)$的。

考虑给 SAM 赋予树形结构，树的根为 0，且其余节点 的父亲为 。则这棵树与原 SAM 的关系是：

- 每个节点的终点集合等于其 **子树** 内所有终点节点对应的终点的集合。
- 每个节点的儿子的等价类都是该节点状态等价类的子集，据此分析可知节点父亲对应的字符串为该节点对应字符串的后缀。

在此基础上可以给每个节点赋予一个最长字符串，是其终点集合中 **任意** 一个终点开始 **往前** 取 `len` 个字符得到的字符串。每个这样的字符串都一样，且 `len` 恰好是满足这个条件的最大值。

这些字符串满足的性质是：

- 如果节点 A 是 B 的祖先，则节点 A 对应的字符串是节点 B 对应的字符串的 **后缀**。

这条性质把字符串所有前缀组成了一棵树，且有许多符合直觉的树的性质。例如，$s[1\ldots i]$ 和 $s[1\ldots j]$ 的最长公共后缀对应的字符串就是 v_i 和 v_j 对应的 LCA 的字符串。实际上，这棵树与将字符串 翻转后得到字符串的压缩后缀树结构相同。

每个状态 i 对应的子串数量是 $\operatorname{len}(i)-\operatorname{len}(link(i))$（根节点例外）。注意到 $link(i)$ 对应的字符串是 i 对应的字符串的一个后缀，这些子串就是 i 对应字符串的所有后缀，去掉被父亲「抢掉」的那部分，即 $link(i)$ 对应字符串的所有后缀。

要求两个字符串的最长公共子串，对其中一个构建 SAM，另一个尝试匹配，遇到无法匹配的节点就往 fail 去跳即可，等价于求所有前缀的最长公共后缀。

```
#include<bits/stdc++.h>
using i64 = long long;
struct SAM {
    static constexpr int ALPHABET = 26;
    struct Node {
        int len;
        int link;
        std::array<int, ALPHABET>next;
        Node(): len{}, link{}, next{} {}
    };
    std::vector<Node> t;
    SAM() {
        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 x) {
        if (t[p].next[x]) {
            int q = t[p].next[x];
            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[x] == q) {
                t[p].next[x] = r;
                p = t[p].link;
            }
            return r;
        }
        int cur = newNode();
        t[cur].len = t[p].len + 1;
        while (not t[p].next[x]) {
            t[p].next[x] = cur;
            p = t[p].link;
        }
        t[cur].link = extend(p, x);
        return cur;
    }
    int next(int p, int x) {
        return t[p].next[x];
    }
```

```
    int link(int p) {
        return t[p].link;
    }
    int len(int p) {
        return t[p].len;
    }
    int size() {
        return t.size();
    }
};
auto main() ->int32_t {
    SAM sam;
    std::string s;
    std::cin >> s;
    int p = 1;
    std::vector<int>end;
    for (const auto & c : s) {
        end.push_back(p = sam.extend(p, c - 'a'));
    }
    std::vector<int>f(sam.size());
    for (const auto & c : end) {
        f[c] += 1;
    }
    std::vector<std::vector<int>>adj(sam.size());
    for (int i = 2; i < sam.size(); i += 1) {
        adj[sam.link(i)].push_back(i);
    }
    i64 ans = 0;
    auto dfs = [&](auto && dfs, int u) ->void {
        for (const auto & v : adj[u]) {
            dfs(dfs, v);
            f[u] += f[v];
        }
        if (f[u] > 1) {
            chmax(ans, 1LL * f[u]*sam.len(u));
        }
    };
    dfs(dfs, 1);
    std::cout << ans << '\n';
    return 0;
}
```

解决 $\sum_{i=1}^{k}\sum_{j=1}^{l}\mathrm{LCP}(s[a_i\ldots n],s[b_i\ldots n])$

求解后缀的 LCP 直接对反串构建 SAM，两个后缀的 LCP 即为对应 SAM 上的节点的的 LCA 。

因此传化为在虚树上DP的经典问题。

```
auto main() ->int32_t {
    int n, q;
    std::cin >> n >> q;
    std::string s;
    std::cin >> s;
    SAM sam;
    int p = 1;
    std::vector<int>end(n + 1);
    for (int i = n - 1; i >= 0; i -= 1) {
        end[i + 1] = p = sam.extend(p, s[i] - 'a');
    }
    LowestCommonAncestor lca(sam.size());
    for (int i = 2; i < sam.size(); i += 1) {
```

```
        lca.addEdge(sam.link(i), i);
    }
    lca.work();
    std::vector<int>su(sam.size()), sv(sam.size());
    std::vector<std::vector<int>>adj(sam.size());
    while (q--) {
        int k, l;
        std::cin >> k >> l;
        std::vector<int>t{1};
        for (int i = 1; i <= k; i += 1) {
            int u;
            std::cin >> u;
            t.push_back(end[u]);
            su[end[u]] += 1;
        }
        for (int i = 1; i <= l; i += 1) {
            int v;
            std::cin >> v;
            t.push_back(end[v]);
            sv[end[v]] += 1;
        }
        std::ranges::sort(t, {}, [&](const auto & u) {
            return lca.dfn[u];
        });
        int m = t.size();
        for (int i = 1; i < m; i += 1) {
            t.push_back(lca(t[i - 1], t[i]));
        }
        std::ranges::sort(t, {}, [&](const auto & u) {
            return lca.dfn[u];
        });
        t.erase(std::unique(t.begin(), t.end()), t.end());
        for (int i = 1; i < t.size(); i += 1) {
            adj[lca(t[i - 1], t[i])].push_back(t[i]);
        }

        i64 ans = 0;
        auto dfs = [&](this auto && dfs, int u) ->void {
            int c = su[u];
            for (const auto & v : adj[u]) {
                dfs(v);
                su[u] += su[v];
                sv[u] += sv[v];
            }
            for (const auto & v : adj[u]) {
                ans += 1LL * su[v] * (sv[u] - sv[v]) * sam.len(u);
            }
            ans += 1LL * c * sv[u] * sam.len(u);
        };
        dfs(1);
        std::cout << ans << '\n';

        for (const auto & u : t) {
            adj[u].clear();
            su[u] = sv[u] = 0;
        }
    }
    return 0;
}
```

找最长公共子串

```
// 尽可能匹配
for (const auto & c : t) {
    while (!sam.next(p, c - 'a')) {
        p = sam.link(p);
        l = sam.len(p);
    }
    p = sam.next(p, c - 'a');
    l += 1;
    chmax(res, l);
}
//广义
int res = 0;
auto dfs = [&](auto && dfs, int u) ->void {
    for (const auto & v : adj[u]) {
        dfs(dfs, v);
        f[u] |= f[v];
    }
    if (u > 1 && f[u].count() >= n) {
        chmax(res, sam.len(u));
    }
};
dfs(dfs, 1);
```

本质不同子串个数 $res = \sum_{u=2}^{sam.size()-1} \text{len}(u) - \text{len}(\text{link}(u))$

```
auto solve = [&]() {
    std::string s;
    std::cin >> s;

    int n = s.size();
    s = ' ' + s;
    std::vector<std::vector<i64>>f(n + 1, std::vector<i64>(n + 1));
    for (int l = 1; l <= n; l += 1) {
        SAM sam;
        int p = 1;
        for (int r = l; r <= n; r += 1) {
            p = sam.extend(p, s[r] - 'a');
            f[l][r] = sam.len(p) - sam.len(sam.link(p));
        }
        for (int r = l; r <= n; r += 1) {
            f[l][r] += f[l][r - 1];
        }
    }

    int q;
    std::cin >> q;
    while (q--) {
        int l, r;
        std::cin >> l >> r;
        std::cout << (f[l][r]) << '\n';
    }
};
```

怎么找 $S[l \dots r]$ 在 SAM 上的对应节点呢？

先记录每一个 r 对应位置的节点， 然后在 SAM 上倍增去找即可。

```
int n = s.size();
std::vector<int>fl(n + 1), fp(n + 1);
```

```
for (int i = 1, p = 1, l = 0; i <= n; i += 1) {
    while (!sam.next(p, s[i - 1] - 'a')) {
        p = sam.link(p);
        l = sam.len(p);
    }
    p = sam.next(p, s[i - 1] - 'a');
    l += 1;
    fp[i] = p;
    fl[i] = l;
}
int p = fp[pr];
for (int k = Kw - 1; k >= 0; k -= 1) {
    if (sam.len(par[p][k]) >= pr - pl + 1) {
        p = par[p][k];
    }
}
std::vector<i64> f(sam.size());
for (const auto &i: end) {
    f[i] += 1;
}
constexpr int Kw = 19;
std::vector<std::array<int, Kw>> par(sam.size());
auto dfs = [&](auto &&dfs, int u) -> void {
    for (int k = 1; k < Kw; k += 1) {
        par[u][k] = par[par[u][k - 1]][k - 1];
    }
    for (const auto &v: adj[u]) {
        par[v][0] = u;
        dfs(dfs, v);
        f[u] += f[v];
    }
};
dfs(dfs, 1);
int q;
std::cin >> q;
while (q--) {
    int l, r;
    std::cin >> l >> r;
    int p = end[r - 1];
    for (int k = Kw - 1; k >= 0; k -= 1) {
        if (sam.len(par[p][k]) >= r - l + 1) {
            p = par[p][k];
        }
    }
    std::cout << f[p] << '\n';
}
```

询问 $s[l\dots r]$ 在 $s[pl\dots pr]$ 的出现次数

线段树合并维护每个节点的 $endpos$ 集合即可。

```
Node *merge(Node *a, Node *b, int l, int r) {
    if (a == nullptr) {
        return b;
    }
    if (b == nullptr) {
        return a;
    }
    Node *c = &pool[tot++];
    if (l == r) {
        c->cnt = a->cnt + b->cnt;
```

```
        return c;
    }
    int mid = (l + r) >> 1;
    c->ch[0] = merge(a->ch[0], b->ch[0], l, mid);
    c->ch[1] = merge(a->ch[1], b->ch[1], mid + 1, r);
    c->cnt = (c->ch[0] == nullptr ? 0 : c->ch[0]->cnt) + (c->ch[1] == nullptr ?
0 : c->ch[1]->cnt);
    return c;
}
for (int i = 1; i <= n; i += 1) {
    p = sam.extend(p, s[i - 1] - 'a');
    add(root[p], 1, n, i);
    end[i] = p;
}
std::vector<std::vector<int>> adj(sam.size());
for (int i = 2; i < sam.size(); i += 1) {
    adj[sam.link(i)].push_back(i);
}
auto dfs = [&](auto &&dfs, int u) -> void {
    for (int k = 1; k < Kw; k += 1) {
        par[u][k] = par[par[u][k - 1]][k - 1];
    }
    for (const auto &v: adj[u]) {
        par[v][0] = u;
        dfs(dfs, v);
        root[u] = merge(root[u], root[v], 1, n);
    }
};
dfs(dfs, 1);
while (q--) {
    int l, r, pl, pr;
    std::cin >> l >> r >> pl >> pr;
    if (pr - pl + 1 < r - l + 1) {
        std::cout << 0 << '\n';
    } else {
        int p = end[r];
        for (int k = Kw - 1; k >= 0; k -= 1) {
            if (sam.len(par[p][k]) >= r - l + 1) {
                p = par[p][k];
            }
        }
        std::cout << query(root[p], 1, n, pl + r - l, pr) << '\n';
    }
}
```

PAM

本质不同的回文串数

根据回文自动机的状态定义可知，所求即为回文自动机的状态数

求回文子串的出现次数

利用 $fail$ 指针构建出回文树向父节点转移即可

```
//rooted in zero
struct PAM {
    static constexpr int ALPHABET = 26;
    static constexpr char OFFSET = 'a';
    struct Node {
```

```
        int len;
        int cnt;
        int suffixlink;
        std::array<int, ALPHABET>next;
        Node(): len{}, cnt{}, suffixlink{}, next{} {}
    };
    int suff;
    std::vector<Node>t;
    std::string s;
    PAM() {
        t.assign(2, Node());
        t[0].suffixlink = 1;
        t[1].suffixlink = 1;
        t[1].len = -1;
        suff = 0;
        s = "$";
    }
    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }
    int getfali(int u, int p) {
        while (p - t[u].len - 1 <= 0 or s[p - t[u].len - 1] != s[p]) {
            u = t[u].suffixlink;
        }
        return u;
    }
    void add(const char& c, int p) {
        s += c;
        int par = getfali(suff, p);
        if (not t[par].next[c - OFFSET]) {
            int cur = newNode();
            t[cur].suffixlink = t[getfali(t[par].suffixlink, p)].next[c -
OFFSET];
            t[par].next[c - OFFSET] = cur;
            t[cur].len = t[par].len + 2;
            t[cur].cnt = t[t[cur].suffixlink].cnt + 1;
        }
        suff = t[par].next[c - OFFSET];
    }
    int size() {
        return t.size();
    }
    int next(int p, const char& c) {
        return t[p].next[c - OFFSET];
    }
    int link(int p) {
        return t[p].suffixlink;
    }
    int len(int p) {
        return t[p].len;
    }
    int cnt(int p) {
        return t[p].cnt;
    }
    std::vector<std::vector<int>>work() {
        std::vector adj(size(), std::vector<int>());
        for (int i = 2; i < size(); i += 1) {
            adj[link(i)].push_back(i);
        }
```

```
        return adj;
    }
};
auto main() ->int32_t {
    std::string s;
    std::cin >> s;
    PAM pam;
    int n = s.size();
    s = ' ' + s;
    std::vector<int>end;
    for (int i = 1; i <= n; i += 1) {
        pam.add(s[i], i);
        end.push_back(pam.suff);
    }

    std::vector<int>f(pam.size());
    for (const auto& x : end) {
        f[x] += 1;
    }
    i64 ans = 0;
    for (int i = pam.size() - 1; i >= 2; i -= 1) {
        f[pam.link(i)] += f[i];
        ans = std::max(ans, i64(f[i]) * pam.len(i) * pam.len(i));
    }
    std::cout << ans << '\n';
    auto dfs = [&](auto && dfs, int u) ->void {
        for (const auto & v : adj[u]) {
            dfs(dfs, v);
            f[u] += f[v];
        }
        chmax(ans, i64(f[u]) * pam.len(u) * pam.len(u));
    };
    dfs(dfs, 0);
    // 0 root
    std::cout << ans << '\n';
}
```

Manacher

原来的奇回文子串会变成以普通字符为中心的奇回文子串；原来的偶回文子串会变成以 $ 为中心的奇回文子串。

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

```
    }
    // int n = s.size();
    // std::vector<std::pair<int, int>>f, g;
    // for (int i = 0; i <= 2 * n; i += 1) {
    //     // [l, center)
    //     f.push_back({(i - r[i] + 1) / 2, (i + 1) / 2});
    //     // [center, r)
    //     g.push_back({i / 2, (i + r[i] - 1) / 2});
    // }
    return r;
}
```

KMP

π :前缀的border

```
std::vector<int> kmp(std::string s) {
    int n = s.size();
    std::vector<int> f(n + 1);
    for (int i = 1, j = 0; i < n; i++) {
        while (j && s[i] != s[j]) {
            j = f[j];
        }
        j += (s[i] == s[j]);
        f[i + 1] = j;
    }
    return f;
}
```

Z

z_i 表示 s 和 $s[i, n-1]$ (即 s_i 开头的后缀) 的LCP

```
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;
}
```

Min

```
std::vector<int> minimalString(std::vector<int> &a) {
    int n = a.size();
    int i = 0, j = 1, k = 0;
    while (k < n and i < n and j < n) {
        if (a[(i + k) % n] == a[(j + k) % n])
            k++;
        else {
            (a[(i + k) % n] > a[(j + k) % n] ? i : j) += k + 1;
```

```
            i += (i == j);
            k = 0;
        }
    }
    k = std::min(i, j);
    std::vector<int> ans(n);
    for (int i = 0; i < n; i++)
        ans[i] = a[(i + k) % n];
    return ans;
}
// 直接返回字典序最小循环同构串
```

AtcoderLibrary

```
namespace atcoder {
namespace internal {
std::vector<int> sa_naive(const std::vector<int>& s) {
    int n = int(s.size());
    std::vector<int> sa(n);
    std::iota(sa.begin(), sa.end(), 0);
    std::sort(sa.begin(), sa.end(), [&](int l, int r) {
        if (l == r) return false;
        while (l < n && r < n) {
            if (s[l] != s[r]) return s[l] < s[r];
            l++;
            r++;
        }
        return l == n;
    });
    return sa;
}
std::vector<int> sa_doubling(const std::vector<int>& s) {
    int n = int(s.size());
    std::vector<int> sa(n), rnk = s, tmp(n);
    std::iota(sa.begin(), sa.end(), 0);
    for (int k = 1; k < n; k *= 2) {
        auto cmp = [&](int x, int y) {
            if (rnk[x] != rnk[y]) return rnk[x] < rnk[y];
            int rx = x + k < n ? rnk[x + k] : -1;
            int ry = y + k < n ? rnk[y + k] : -1;
            return rx < ry;
        };
        std::sort(sa.begin(), sa.end(), cmp);
        tmp[sa[0]] = 0;
        for (int i = 1; i < n; i++) {
            tmp[sa[i]] = tmp[sa[i - 1]] + (cmp(sa[i - 1], sa[i]) ? 1 : 0);
        }
        std::swap(tmp, rnk);
    }
    return sa;
}
// SA-IS, linear-time suffix array construction
// Reference:
// G. Nong, S. Zhang, and W. H. Chan,
// Two Efficient Algorithms for Linear Time Suffix Array Construction
template <int THRESHOLD_NAIVE = 10, int THRESHOLD_DOUBLING = 40>
std::vector<int> sa_is(const std::vector<int>& s, int upper) {
    int n = int(s.size());
    if (n == 0) return {};
    if (n == 1) return {0};
```

```
    if (n == 2) {
        if (s[0] < s[1]) {
            return {0, 1};
        } else {
            return {1, 0};
        }
    }
    if (n < THRESHOLD_NAIVE) {
        return sa_naive(s);
    }
    if (n < THRESHOLD_DOUBLING) {
        return sa_doubling(s);
    }
    std::vector<int> sa(n);
    std::vector<bool> ls(n);
    for (int i = n - 2; i >= 0; i--) {
        ls[i] = (s[i] == s[i + 1]) ? ls[i + 1] : (s[i] < s[i + 1]);
    }
    std::vector<int> sum_l(upper + 1), sum_s(upper + 1);
    for (int i = 0; i < n; i++) {
        if (!ls[i]) {
            sum_s[s[i]]++;
        } else {
            sum_l[s[i] + 1]++;
        }
    }
    for (int i = 0; i <= upper; i++) {
        sum_s[i] += sum_l[i];
        if (i < upper) sum_l[i + 1] += sum_s[i];
    }
    auto induce = [&](const std::vector<int>& lms) {
        std::fill(sa.begin(), sa.end(), -1);
        std::vector<int> buf(upper + 1);
        std::copy(sum_s.begin(), sum_s.end(), buf.begin());
        for (auto d : lms) {
            if (d == n) continue;
            sa[buf[s[d]]++] = d;
        }
        std::copy(sum_l.begin(), sum_l.end(), buf.begin());
        sa[buf[s[n - 1]]++] = n - 1;
        for (int i = 0; i < n; i++) {
            int v = sa[i];
            if (v >= 1 && !ls[v - 1]) {
                sa[buf[s[v - 1]]++] = v - 1;
            }
        }
        std::copy(sum_l.begin(), sum_l.end(), buf.begin());
        for (int i = n - 1; i >= 0; i--) {
            int v = sa[i];
            if (v >= 1 && ls[v - 1]) {
                sa[--buf[s[v - 1] + 1]] = v - 1;
            }
        }
    };
    std::vector<int> lms_map(n + 1, -1);
    int m = 0;
    for (int i = 1; i < n; i++) {
        if (!ls[i - 1] && ls[i]) {
            lms_map[i] = m++;
        }
```

```
    }
    std::vector<int> lms;
    lms.reserve(m);
    for (int i = 1; i < n; i++) {
        if (!ls[i - 1] && ls[i]) {
            lms.push_back(i);
        }
    }
    induce(lms);
    if (m) {
        std::vector<int> sorted_lms;
        sorted_lms.reserve(m);
        for (int v : sa) {
            if (lms_map[v] != -1) sorted_lms.push_back(v);
        }
        std::vector<int> rec_s(m);
        int rec_upper = 0;
        rec_s[lms_map[sorted_lms[0]]] = 0;
        for (int i = 1; i < m; i++) {
            int l = sorted_lms[i - 1], r = sorted_lms[i];
            int end_l = (lms_map[l] + 1 < m) ? lms[lms_map[l] + 1] : n;
            int end_r = (lms_map[r] + 1 < m) ? lms[lms_map[r] + 1] : n;
            bool same = true;
            if (end_l - l != end_r - r) {
                same = false;
            } else {
                while (l < end_l) {
                    if (s[l] != s[r]) {
                        break;
                    }
                    l++;
                    r++;
                }
                if (l == n || s[l] != s[r]) same = false;
            }
            if (!same) rec_upper++;
            rec_s[lms_map[sorted_lms[i]]] = rec_upper;
        }
        auto rec_sa =
            sa_is<THRESHOLD_NAIVE, THRESHOLD_DOUBLING>(rec_s, rec_upper);

        for (int i = 0; i < m; i++) {
            sorted_lms[i] = lms[rec_sa[i]];
        }
        induce(sorted_lms);
    }
    return sa;
}
}  // namespace internal
std::vector<int> suffix_array(const std::vector<int>& s, int upper) {
    assert(0 <= upper);
    for (int d : s) {
        assert(0 <= d && d <= upper);
    }
    auto sa = internal::sa_is(s, upper);
    return sa;
}
template <class T> std::vector<int> suffix_array(const std::vector<T>& s) {
    int n = int(s.size());
    std::vector<int> idx(n);
```

```
    iota(idx.begin(), idx.end(), 0);
    sort(idx.begin(), idx.end(), [&](int l, int r) { return s[l] < s[r]; });
    std::vector<int> s2(n);
    int now = 0;
    for (int i = 0; i < n; i++) {
        if (i && s[idx[i - 1]] != s[idx[i]]) now++;
        s2[idx[i]] = now;
    }
    return internal::sa_is(s2, now);
}
std::vector<int> suffix_array(const std::string& s) {
    int n = int(s.size());
    std::vector<int> s2(n);
    for (int i = 0; i < n; i++) {
        s2[i] = s[i];
    }
    return internal::sa_is(s2, 255);
}
// Reference:
// T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park,
// Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its
// Applications
template <class T>
std::vector<int> lcp_array(const std::vector<T>& s,
                           const std::vector<int>& sa) {
    int n = int(s.size());
    assert(n >= 1);
    std::vector<int> rnk(n);
    for (int i = 0; i < n; i++) {
        rnk[sa[i]] = i;
    }
    std::vector<int> lcp(n - 1);
    int h = 0;
    for (int i = 0; i < n; i++) {
        if (h > 0) h--;
        if (rnk[i] == 0) continue;
        int j = sa[rnk[i] - 1];
        for (; j + h < n && i + h < n; h++) {
            if (s[j + h] != s[i + h]) break;
        }
        lcp[rnk[i] - 1] = h;
    }
    return lcp;
}
std::vector<int> lcp_array(const std::string& s, const std::vector<int>& sa) {
    int n = int(s.size());
    std::vector<int> s2(n);
    for (int i = 0; i < n; i++) {
        s2[i] = s[i];
    }
    return lcp_array(s2, sa);
}
// Reference:
// D. Gusfield,
// Algorithms on Strings, Trees, and Sequences: Computer Science and
// Computational Biology
template <class T> std::vector<int> z_algorithm(const std::vector<T>& s) {
    int n = int(s.size());
    if (n == 0) return {};
    std::vector<int> z(n);
```

```
    z[0] = 0;
    for (int i = 1, j = 0; i < n; i++) {
        int& k = z[i];
        k = (j + z[j] <= i) ? 0 : std::min(j + z[j] - i, z[i - j]);
        while (i + k < n && s[k] == s[i + k]) k++;
        if (j + z[j] < i + z[i]) j = i;
    }
    z[0] = n;
    return z;
}
std::vector<int> z_algorithm(const std::string& s) {
    int n = int(s.size());
    std::vector<int> s2(n);
    for (int i = 0; i < n; i++) {
        s2[i] = s[i];
    }
    return z_algorithm(s2);
}
```

Graph

最大匹配等于最小点覆盖(np-hard)，所有的边都至少有一个端点位于点覆盖里面。

SCC

DAG可以求解最长路：拓扑排序之后dp即可

```
struct SCC {
    int n;
    std::vector<std::vector<int>>adj;
    std::vector<int>dfn, low, bel;
    std::vector<int>stk;
    int cnt, cur;
    SCC () {}
    SCC(int n) {
        this->n = n;
        adj.assign(n + 1, std::vector<int>());
        dfn.assign(n + 1, -1), bel.assign(n + 1, -1), low.resize(n + 1);
        cur = cnt = 0;
    }
    void addEdge(int u, int v) {
        adj[u].emplace_back(v);
    }
    void tarjan(int x) {
        dfn[x] = low[x] = ++cur;
        stk.emplace_back(x);
        for (const auto & y : adj[x]) {
            if (dfn[y] == -1) {
                tarjan(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]) {
            cnt += 1;
            int y;
            do {
                y = stk.back(), stk.pop_back();
                bel[y] = cnt;
```

```
            } while (y != x);
        }
    }
    std::vector<int> work() {
        for (int j = 1; j <= n; j += 1) {
            if (dfn[j] == -1) {
                tarjan(j);
            }
        }
        return bel;
    }
};
```

twoSat

定义

2-SAT，简单的说就是给出 n 个集合，每个集合有两个元素，已知若干个 `<a,b>`，表示 a 与 b 矛盾（其中 a 与 b 属于不同的集合）。然后从每个集合选择一个元素，判断能否一共选 n 个两两不矛盾的元素。显然可能有多种选择方案，一般题中只需要求出一种即可。

现实意义

比如邀请人来吃喜酒，夫妻二人必须去一个，然而某些人之间有矛盾（比如 A 先生与 B 女士有矛盾，C 女士不想和 D 先生在一起），那么我们要确定能否避免来人之间有矛盾，有时需要方案。这是一类生活中常见的问题。

使用布尔方程表示上述问题

设 a 表示 A 先生去参加，那么 B 女士就不能参加（`¬a`）；b 表示 C 女士参加，那么 `¬b` 也一定成立（D 先生不参加）。总结一下，即 `(a ∨ b)`（变量 a, b 至少满足一个）。对这些变量关系建有向图，则有：`¬a → b ∧ ¬b → a`（a 不成立则 b 一定成立；同理，b 不成立则 a 一定成立）。建图之后，我们就可以使用缩点算法来求解 2-SAT 问题了。

```
struct TwoSat {
    int n;
    std::vector<std::vector<int>>adj;
    std::vector<bool>res;
    TwoSat() {}
    TwoSat(const int & n): n(n), adj(2 * n + 2), res(n + 1) {}
    void addClause(int u, bool f, int v, bool g) {
        // 本质上，点 u 对应条件 f，点 v 对应条件 g，在Satisfiable True的情况下两个中的
一个必定成立
        adj[2 * u + !f].push_back(2 * v + g);
        adj[2 * v + !g].push_back(2 * u + f);
    }
    bool Satisfiable() {
        int cur = 0, cnt = 0;
        std::vector<int>stk, bel(2 * n + 2, -1), low(2 * n + 2, -1), dfn(2 * n
+ 2, -1);
        auto dfs = [&](auto && dfs, int x) ->void {
            stk.push_back(x);
            low[x] = dfn[x] = ++cur;
            for (const auto & y : adj[x]) {
                if (dfn[y] == -1) {
                    dfs(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]) {
                cnt += 1;
                int y;
                do {
                    y = stk.back();
                    stk.pop_back();
                    bel[y] = cnt;
                } while (y != x);
            }
        };
        for (int u = 1; u <= 2 * n + 1; u += 1) {
            if (dfn[u] == -1) {
                dfs(dfs, u);
            }
        }
        for (int u = 1; u <= n; u += 1) {
            if (bel[2 * u] == bel[2 * u + 1]) {
                return false;
            }
            res[u] = bel[2 * u] > bel[2 * u + 1];
        }
        return true;
    }
    std::vector<bool> answer() {
        return res;
    }
};
```

EBCC

```
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) {
        this->n = n;
        adj.assign(n + 1, std::vector<int>());
        dfn.assign(n + 1, -1), bel.assign(n + 1, -1), low.resize(n + 1);
        stk.clear();
        cnt = cur = 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 (const 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;
            cnt++;
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
        }
    }
    std::vector<int> work() {
        dfs(1, 0);
        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 + 1), g.cnte.resize(cnt + 1);
        for (int i = 1; i <= n; i++) {
            g.siz[bel[i]]++;
            for (const 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;
    }
};
```

Boruvka

每个联通块对外连最短的出边，复杂度为$\mathrm{O}(n \log_2 n)$.

```
struct DSU {
    std::vector<int>par, siz;
    std::vector<std::vector<int>>adj;
    DSU() = default;
    DSU(const int& n) {
        par.resize(n);
        std::iota(par.begin(), par.end(), 0);
        siz.assign(n, 1);
        adj.assign(n, {});
        for (int u = 0; u < n; u += 1) {
            adj[u].push_back(u);
        }
```

```
    }
    int find(int u) {
        while (par[u] != u) {
            u = par[u] = par[par[u]];
        }
        return u;
    }
    bool same(int u, int v) {
        return find(u) == find(v);
    }
    bool merge(int u, int v) {
        u = find(u), v = find(v);
        if (u == v) {
            return false;
        }
        if (siz[u] < siz[v]) {
            std::swap(u, v);
        }
        par[v] = u;
        siz[u] += siz[v];
        for (const auto & x : adj[v]) {
            adj[u].push_back(x);
        }
        adj[v].clear();
        // adj[v].shrink_to_fit();
        return true;
    }
    int size(int u) {
        return siz[find(u)];
    }
};
int cnt = n;
DSU dsu(n);
while (cnt > 1) {
    for (int u = 0; u < n; u += 1) {
        if (!dsu.adj[u].empty()) {
            for (const auto & v : dsu.adj[u]) {
                //通常情况下求解完全图MST，每个联通块的向外引出的边是最小的，需要结合其他算
法求解
            }
            if (dsu.merge) {
                cnt -= 1;
            }
        }
    }
}
```

Kruskal重构树

note：并查集不可以启发式合并。

两个点的连接到一个新的点上，这个点的点权作为原先两个点的边权。

最终原先的 n 个点都是叶子，在重构树上的 LCA 具有重要的路径关系。

维护点权的集合要注意初始化!

```
for (auto [w, u, v] : edges) {
    u = dsu.find(u), v = dsu.find(v);
    if (u != v) {
        n += 1;
```

```
        dsu.par[u] = dsu.par[v] = n;
        adj[n].push_back(u);
        adj[n].push_back(v);
        x[n] = w;
    }
}
auto dfs = [&](auto && dfs, int u) ->void {
    for (const auto & v : adj[u]) {
        par[v][0] = u;
        dfs(dfs, v);
        a[u] += a[v];
    }
    max[u][0] = x[par[u][0]] - a[u];
};
dfs(dfs, n);
for (int k = 1; k < Kw; k += 1) {
    for (int u = n; u >= 1; u -= 1) {
        par[u][k] = par[par[u][k - 1]][k - 1];
        max[u][k] = std::max(max[u][k - 1], max[par[u][k - 1]][k - 1]);
    }
}
```

最短路图求割边

```
auto main() ->int32_t {
    int n, m;
    std::cin >> n >> m;
    std::vector<int>u(m + 1), v(m + 1), w(m + 1);
    std::vector adj(n + 1, std::vector<std::array<int, 2>>());
    for (int i = 1; i <= m; i += 1) {
        std::cin >> u[i] >> v[i] >> w[i];
        adj[u[i]].push_back({v[i], w[i]});
        adj[v[i]].push_back({u[i], w[i]});
    }
    constexpr i64 inf = 1E18;
    auto Dijkstra = [&](int s) {
        std::vector<i64>dis(n + 1, inf);
        std::vector<Z>p(n + 1);
        std::priority_queue<std::pair<i64, int>, std::vector<std::pair<i64,
int>>, std::greater<>>q;
        p[s] = 1;
        q.emplace(dis[s] = 0, s);
        while (not q.empty()) {
            auto [d, u] = q.top();
            q.pop();
            if (dis[u] != d) {
                continue;
            }
            for (const auto & [v, w] : adj[u]) {
                if (dis[v] > dis[u] + w) {
                    p[v] = p[u];
                    q.emplace(dis[v] = dis[u] + w, v);
                } else if (dis[v] == dis[u] + w) {
                    p[v] += p[u];
                }
            }
        }
        return std::pair(dis, p);
    };
    auto [d1, p1] = Dijkstra(1);
```

```
    auto [dn, pn] = Dijkstra(n);
    auto ok = [&](int u, int v, int w) {
        return (d1[n] == d1[u] + w + dn[v]) and (p1[u] * pn[v] == p1[n]);
    };
    for (int i = 1; i <= m; i += 1) {
        std::cout << (ok(u[i], v[i], w[i]) or ok(v[i], u[i], w[i]) ? "Yes\n" :
"No\n");
    }
    return 0;
}
```

SegmentTreeOptimizeGraph

```
auto main() ->int32_t {
    int n, q, s;
    std::cin >> n >> q >> s;
    const int delta = 4 * n;
    std::vector adj(2 * delta + 1, std::vector<std::pair<int, int>>());
    std::vector<int>Index(n + 1);
    auto addEdge = [&](int u, int v, int w) {
        adj[u].emplace_back(v, w);
    };
    auto build = [&](this auto && self, int p, int l, int r) {
        if (l == r) {
            Index[l] = p;
            addEdge(p, p + delta, 0);
            return ;
        }
        int mid = (l + r) / 2, lsh = 2 * p, rsh = 2 * p + 1;
        addEdge(p, lsh, 0);
        addEdge(p, rsh, 0);
        addEdge(lsh + delta, p + delta, 0);
        addEdge(rsh + delta, p + delta, 0);
        self(lsh, l, mid);
        self(rsh, mid + 1, r);
    };
    auto update = [&](this auto && self, int p, int l, int r, int v, int L, int
R, int w, int o) {
        if (L <= l and r <= R) {
            if (o == 2) {
                addEdge(Index[v] + delta, p, w);
            } else {
                addEdge(p + delta, Index[v], w);
            }
            return;
        }
        int mid = (l + r) / 2;
        if (L <= mid) {
            self(p << 1, l, mid, v, L, R, w, o);
        }
        if (mid < R) {
            self(p << 1 | 1, mid + 1, r, v, L, R, w, o);
        }
    };
    build(1, 1, n);
    while (q--) {
        int o;
        std::cin >> o;
        if (o == 1) {
            int v, u, w;
```

```
            std::cin >> v >> u >> w;
            addEdge(Index[v] + delta, Index[u], w);
        } else {
            int v, l, r, w;
            std::cin >> v >> l >> r >> w;
            update(1, 1, n, v, l, r, w, o);
        }
    }
    std::priority_queue<std::pair<i64, int>, std::vector<std::pair<i64, int>>,
std::greater<>>Q;
    std::vector<i64>dis(2 * delta + 1, -1);
    Q.emplace(dis[Index[s]] = 0, Index[s]);
    while (not Q.empty()) {
        auto [d, u] = Q.top();
        Q.pop();
        if (d != dis[u]) {
            continue;
        }
        for (const auto & [v, w] : adj[u]) {
            if (dis[v] == -1 or dis[v] > dis[u] + w) {
                Q.emplace(dis[v] = dis[u] + w, v);
            }
        }
    }
    for (int i = 1; i <= n; i += 1) {
        std::cout << dis[Index[i]] << ' ';
    }
    return 0;
}
```

Augmenting

```
auto main() ->int {
    int n, m, e;
    std::cin >> n >> m >> e;
    std::vector<std::vector<int>>adj(n + m + 1);
    for (int i = 1; i <= e; i += 1) {
        int u, v;
        std::cin >> u >> v;
        adj[u].push_back(v);
    }
    int ans = 0;
    std::vector<int>vis(m + n + 1), mch(n + m + 1);
    auto dfs = [&](auto && dfs, int u, int c) ->bool {
        for (const auto & v : adj[u]) {
            if (vis[v] == c) {
                continue;
            }
            vis[v] = c;
            if (!mch[v] || dfs(dfs, mch[v], c)) {
                mch[v] = u;
                return true;
            }
        }
        return false;
    };
    for (int u = 1; u <= n; u += 1) {
        if (dfs(dfs, u, u)) {
            ans += 1;
        }
```

```
    }
    std::cout << ans << '\n';
    return 0;
}
```

MincostFlow

0 - index

```
template <class T>
struct MincostFlow {
    struct edge {
        int to;
        T cap;
        T cost;
        edge(const int& to, const T& cap, const T& cost): to(to), cap(cap),
cost(cost) {}
    };
    int n;
    std::vector<edge>e;
    std::vector<std::vector<int>>adj;
    std::vector<T>h, dis;
    std::vector<int>pre;
    bool dijkstra(int s, int t) {
        dis.assign(n, -1);
        pre.assign(n, -1);
        std::priority_queue<std::pair<T,int>, std::vector<std::pair<T,int>>,
std::greater<>>Q;
        Q.push({dis[s] = 0, s});
        while (not Q.empty()) {
            auto [d, u] = Q.top();
            Q.pop();
            if (dis[u] != d) {
                continue;
            }
            for (const auto & j : adj[u]) {
                const auto & [v, cap, cost] = e[j];
                if (cap > 0 and (dis[v] == -1 or dis[v] > d + h[u] - h[v] +
cost)) {
                    pre[v] = j;
                    Q.push({dis[v] = d + h[u] - h[v] + cost, v});
                }
            }
        }
        return dis[t] != -1;
    }
    MincostFlow() {}
    MincostFlow(int n) {
        this->n = n;
        e.clear();
        adj.assign(n, {});
    }
    void addEdge(int u, int v, T cap, T cost) {
        adj[u].push_back(e.size());
        e.emplace_back(v, cap, cost);
        adj[v].push_back(e.size());
        e.emplace_back(u, 0, -cost);
    }
    std::pair<T, T> flow(int s, int t) {
        T flow = 0, cost = 0;
```

```
        h.assign(n, 0);
        while (dijkstra(s, t)) {
            for (int i = 0; i < n; i += 1) {
                h[i] += dis[i];
            }
            T aug = std::numeric_limits<T>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                aug = std::min(aug, e[pre[i]].cap);
            }
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                e[pre[i]].cap -= aug;
                e[pre[i] ^ 1].cap += aug;
            }
            flow += aug;
            cost += aug * h[t];
        }
        return std::pair(flow, cost);
    }
    struct Edge {
        int from;
        int to;
        T cap;
        T cost;
        T flow;
    };
    std::vector<Edge> edges() {
        std::vector<Edge>rec;
        for (int i = 0; i < e.size(); i += 2) {
            rec.push_back({.from = e[i + 1].to, .to = e[i].to, .cap = e[i].cap
+ e[i + 1].cap, .cost = e[i].cost, .flow = e[i + 1].cap});
        }
        return rec;
    }
};
namespace atcoder {
template <class Cap, class Cost> struct mcf_graph {
  public:
    mcf_graph() {}
    explicit mcf_graph(int n) : _n(n) {}
    int add_edge(int from, int to, Cap cap, Cost cost) {
        assert(0 <= from && from < _n);
        assert(0 <= to && to < _n);
        assert(0 <= cap);
        assert(0 <= cost);
        int m = int(_edges.size());
        _edges.push_back({from, to, cap, 0, cost});
        return m;
    }
    struct edge {
        int from, to;
        Cap cap, flow;
        Cost cost;
    };
    edge get_edge(int i) {
        int m = int(_edges.size());
        assert(0 <= i && i < m);
        return _edges[i];
    }
    std::vector<edge> edges() { return _edges; }
    std::pair<Cap, Cost> flow(int s, int t) {
```

```
        return flow(s, t, std::numeric_limits<Cap>::max());
    }
    std::pair<Cap, Cost> flow(int s, int t, Cap flow_limit) {
        return slope(s, t, flow_limit).back();
    }
    std::vector<std::pair<Cap, Cost>> slope(int s, int t) {
        return slope(s, t, std::numeric_limits<Cap>::max());
    }
    std::vector<std::pair<Cap, Cost>> slope(int s, int t, Cap flow_limit) {
        assert(0 <= s && s < _n);
        assert(0 <= t && t < _n);
        assert(s != t);
        int m = int(_edges.size());
        std::vector<int> edge_idx(m);
        auto g = [&]() {
            std::vector<int> degree(_n), redge_idx(m);
            std::vector<std::pair<int, _edge>> elist;
            elist.reserve(2 * m);
            for (int i = 0; i < m; i++) {
                auto e = _edges[i];
                edge_idx[i] = degree[e.from]++;
                redge_idx[i] = degree[e.to]++;
                elist.push_back({e.from, {e.to, -1, e.cap - e.flow, e.cost}});
                elist.push_back({e.to, {e.from, -1, e.flow, -e.cost}});
            }
            auto _g = internal::csr<_edge>(_n, elist);
            for (int i = 0; i < m; i++) {
                auto e = _edges[i];
                edge_idx[i] += _g.start[e.from];
                redge_idx[i] += _g.start[e.to];
                _g.elist[edge_idx[i]].rev = redge_idx[i];
                _g.elist[redge_idx[i]].rev = edge_idx[i];
            }
            return _g;
        }();
        auto result = slope(g, s, t, flow_limit);
        for (int i = 0; i < m; i++) {
            auto e = g.elist[edge_idx[i]];
            _edges[i].flow = _edges[i].cap - e.cap;
        }
        return result;
    }
  private:
    int _n;
    std::vector<edge> _edges;
    // inside edge
    struct _edge {
        int to, rev;
        Cap cap;
        Cost cost;
    };
    std::vector<std::pair<Cap, Cost>> slope(internal::csr<_edge>& g,
                                            int s,
                                            int t,
                                            Cap flow_limit) {
        // variants (C = maxcost):
        // -(n-1)C <= dual[s] <= dual[i] <= dual[t] = 0
        // reduced cost (= e.cost + dual[e.from] - dual[e.to]) >= 0 for all
edge
        // dual_dist[i] = (dual[i], dist[i])
```

```
        std::vector<std::pair<Cost, Cost>> dual_dist(_n);
        std::vector<int> prev_e(_n);
        std::vector<bool> vis(_n);
        struct Q {
            Cost key;
            int to;
            bool operator<(Q r) const { return key > r.key; }
        };
        std::vector<int> que_min;
        std::vector<Q> que;
        auto dual_ref = [&]() {
            for (int i = 0; i < _n; i++) {
                dual_dist[i].second = std::numeric_limits<Cost>::max();
            }
            std::fill(vis.begin(), vis.end(), false);
            que_min.clear();
            que.clear();
            // que[0..heap_r) was heapified
            size_t heap_r = 0;
            dual_dist[s].second = 0;
            que_min.push_back(s);
            while (!que_min.empty() || !que.empty()) {
                int v;
                if (!que_min.empty()) {
                    v = que_min.back();
                    que_min.pop_back();
                } else {
                    while (heap_r < que.size()) {
                        heap_r++;
                        std::push_heap(que.begin(), que.begin() + heap_r);
                    }
                    v = que.front().to;
                    std::pop_heap(que.begin(), que.end());
                    que.pop_back();
                    heap_r--;
                }
                if (vis[v]) continue;
                vis[v] = true;
                if (v == t) break;
                // dist[v] = shortest(s, v) + dual[s] - dual[v]
                // dist[v] >= 0 (all reduced cost are positive)
                // dist[v] <= (n-1)C
                Cost dual_v = dual_dist[v].first, dist_v = dual_dist[v].second;
                for (int i = g.start[v]; i < g.start[v + 1]; i++) {
                    auto e = g.elist[i];
                    if (!e.cap) continue;
                    // |-dual[e.to] + dual[v]| <= (n-1)C
                    // cost <= C - -(n-1)C + 0 = nC
                    Cost cost = e.cost - dual_dist[e.to].first + dual_v;
                    if (dual_dist[e.to].second - dist_v > cost) {
                        Cost dist_to = dist_v + cost;
                        dual_dist[e.to].second = dist_to;
                        prev_e[e.to] = e.rev;
                        if (dist_to == dist_v) {
                            que_min.push_back(e.to);
                        } else {
                            que.push_back(Q{dist_to, e.to});
                        }
                    }
                }
```

```
            }
            if (!vis[t]) {
                return false;
            }
            for (int v = 0; v < _n; v++) {
                if (!vis[v]) continue;
                // dual[v] = dual[v] - dist[t] + dist[v]
                //         = dual[v] - (shortest(s, t) + dual[s] - dual[t]) +
                //         (shortest(s, v) + dual[s] - dual[v]) = - shortest(s,
                //         t) + dual[t] + shortest(s, v) = shortest(s, v) -
                //         shortest(s, t) >= 0 - (n-1)C
                dual_dist[v].first -= dual_dist[t].second -
dual_dist[v].second;
            }
            return true;
        };
        Cap flow = 0;
        Cost cost = 0, prev_cost_per_flow = -1;
        std::vector<std::pair<Cap, Cost>> result = {{Cap(0), Cost(0)}};
        while (flow < flow_limit) {
            if (!dual_ref()) break;
            Cap c = flow_limit - flow;
            for (int v = t; v != s; v = g.elist[prev_e[v]].to) {
                c = std::min(c, g.elist[g.elist[prev_e[v]].rev].cap);
            }
            for (int v = t; v != s; v = g.elist[prev_e[v]].to) {
                auto& e = g.elist[prev_e[v]];
                e.cap += c;
                g.elist[e.rev].cap -= c;
            }
            Cost d = -dual_dist[s].first;
            flow += c;
            cost += c * d;
            if (prev_cost_per_flow == d) {
                result.pop_back();
            }
            result.push_back({flow, cost});
            prev_cost_per_flow = d;
        }
        return result;
    }
};
}  // namespace atcoder
#endif  // ATCODER_MINCOSTFLOW_HPP
```

Maxflow

```
constexpr int inf = 1E9;
template <class T>
struct MaxFlow {
    struct edge {
        int to;
        T cap;
        edge(const int& to, const T& cap): to(to), cap(cap) {}
    };
    int n;
    std::vector<edge>e;
    std::vector<std::vector<int>>adj;
    std::vector<T>cur, h;
    MaxFlow() {}
```

```
    MaxFlow(int n) {
        this->n = n;
        e.clear();
        adj.assign(n, {});
        cur.resize(n);
        h.resize(n);
    }
    bool bfs(int s, int t) {
        h.assign(n, -1);
        std::queue<int>Q;
        h[s] = 0;
        Q.push(s);
        while (not Q.empty()) {
            auto u = Q.front();
            Q.pop();
            for (const auto & i : adj[u]) {
                const auto & [to, cap] = e[i];
                if (cap > 0 and h[to] == -1) {
                    h[to] = h[u] + 1;
                    if (to == t) {
                        return true;
                    }
                    Q.push(to);
                }
            }
        }
        return false;
    }
    T dfs(int u, int t, T f) {
        if (u == t) {
            return f;
        }
        T r = f;
        for (int& i = cur[u]; i < (int)adj[u].size(); i += 1) {
            const int j = adj[u][i];
            auto [v, cap] = e[j];
            if (cap > 0 and h[v] == h[u] + 1) {
                auto a = dfs(v, t, std::min(r, cap));
                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) {
        adj[u].push_back(e.size());
        e.emplace_back(v, c);
        adj[v].push_back(e.size());
        e.emplace_back(u, 0);
    }
    T flow(int s, int t) {
        T res = 0;
        while (bfs(s, t)) {
            cur.assign(n, 0);
            res += dfs(s, t, std::numeric_limits<T>::max());
        }
```

```
        return res;
    }
    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>rec;
        for (int i = 0; i < e.size(); i += 2) {
            rec.push_back({.from = e[i + 1].to, .to = e[i].to, .cap = e[i].cap
+ e[i + 1].cap, .flow = e[i + 1].cap});
        }
        return rec;
    }
};
#ifndef ATCODER_MAXFLOW_HPP
#define ATCODER_MAXFLOW_HPP 1
#include <algorithm>
#include <cassert>
#include <limits>
#include <queue>
#include <vector>
#include "atcoder/internal_queue"
namespace atcoder {
template <class Cap> struct mf_graph {
  public:
    mf_graph() : _n(0) {}
    explicit mf_graph(int n) : _n(n), g(n) {}
    int add_edge(int from, int to, Cap cap) {
        assert(0 <= from && from < _n);
        assert(0 <= to && to < _n);
        assert(0 <= cap);
        int m = int(pos.size());
        pos.push_back({from, int(g[from].size())});
        int from_id = int(g[from].size());
        int to_id = int(g[to].size());
        if (from == to) to_id++;
        g[from].push_back(_edge{to, to_id, cap});
        g[to].push_back(_edge{from, from_id, 0});
        return m;
    }
    struct edge {
        int from, to;
        Cap cap, flow;
    };
    edge get_edge(int i) {
        int m = int(pos.size());
        assert(0 <= i && i < m);
        auto _e = g[pos[i].first][pos[i].second];
        auto _re = g[_e.to][_e.rev];
        return edge{pos[i].first, _e.to, _e.cap + _re.cap, _re.cap};
```

```
    }
    std::vector<edge> edges() {
        int m = int(pos.size());
        std::vector<edge> result;
        for (int i = 0; i < m; i++) {
            result.push_back(get_edge(i));
        }
        return result;
    }
    void change_edge(int i, Cap new_cap, Cap new_flow) {
        int m = int(pos.size());
        assert(0 <= i && i < m);
        assert(0 <= new_flow && new_flow <= new_cap);
        auto& _e = g[pos[i].first][pos[i].second];
        auto& _re = g[_e.to][_e.rev];
        _e.cap = new_cap - new_flow;
        _re.cap = new_flow;
    }
    Cap flow(int s, int t) {
        return flow(s, t, std::numeric_limits<Cap>::max());
    }
    Cap flow(int s, int t, Cap flow_limit) {
        assert(0 <= s && s < _n);
        assert(0 <= t && t < _n);
        assert(s != t);
        std::vector<int> level(_n), iter(_n);
        internal::simple_queue<int> que;
        auto bfs = [&]() {
            std::fill(level.begin(), level.end(), -1);
            level[s] = 0;
            que.clear();
            que.push(s);
            while (!que.empty()) {
                int v = que.front();
                que.pop();
                for (auto e : g[v]) {
                    if (e.cap == 0 || level[e.to] >= 0) continue;
                    level[e.to] = level[v] + 1;
                    if (e.to == t) return;
                    que.push(e.to);
                }
            }
        };
        auto dfs = [&](auto self, int v, Cap up) {
            if (v == s) return up;
            Cap res = 0;
            int level_v = level[v];
            for (int& i = iter[v]; i < int(g[v].size()); i++) {
                _edge& e = g[v][i];
                if (level_v <= level[e.to] || g[e.to][e.rev].cap == 0)
continue;
                Cap d =
                    self(self, e.to, std::min(up - res, g[e.to][e.rev].cap));
                if (d <= 0) continue;
                g[v][i].cap += d;
                g[e.to][e.rev].cap -= d;
                res += d;
                if (res == up) return res;
            }
            level[v] = _n;
```

```
            return res;
        };
        Cap flow = 0;
        while (flow < flow_limit) {
            bfs();
            if (level[t] == -1) break;
            std::fill(iter.begin(), iter.end(), 0);
            Cap f = dfs(dfs, t, flow_limit - flow);
            if (!f) break;
            flow += f;
        }
        return flow;
    }
    std::vector<bool> min_cut(int s) {
        std::vector<bool> visited(_n);
        internal::simple_queue<int> que;
        que.push(s);
        while (!que.empty()) {
            int p = que.front();
            que.pop();
            visited[p] = true;
            for (auto e : g[p]) {
                if (e.cap && !visited[e.to]) {
                    visited[e.to] = true;
                    que.push(e.to);
                }
            }
        }
        return visited;
    }
  private:
    int _n;
    struct _edge {
        int to, rev;
        Cap cap;
    };
    std::vector<std::pair<int, int>> pos;
    std::vector<std::vector<_edge>> g;
};
}  // namespace atcoder
#endif  // ATCODER_MAXFLOW_HPP
```

Minus

差分约束系统 是一种特殊的 n 元一次不等式组，它包含 n 个变量 $x_1, x_2, \ldots, x_n$ 以及 m 个约束条件，每个约束条件是由两个其中的变量作差构成的，形如 $x_i - x_j \leq c_k$，其中 $1 \leq i, j \leq n, i \neq j, 1 \leq k \leq m$ 并且 c_k 是常数（可以是非负数，也可以是\负数）。我们要解决的问题是：求一组解 $x_1 = a_1, x_2 = a_2, \ldots, x_n = a_n$ 使得所有的约束条件得到满足，否则判断出无解。 差分约束系统中的每个约束条件 $x_i - x_j \leq c_k$ 都可以变形成 $x_i \leq x_j + c_k$，这与单源最短路中的三角形不等式 $dist[y] \leq dist[x] + z$ 非常相似。因此，我们可以把每个变量 x_i 看做图中的一个结点，对于每个约束条件 $x_i - x_j \leq c_k$，从结点 j 向结点 i 连一条\长度为 c_k 的有向边。

题意	转化	连边
$X_a - X_b \geq c$	$X_b - X_a \leq c$	`add(a, b, -c);`
$X_a - X_b \leq c$	$X_a - X_b \leq c$	`add(b, a, c);`
$X_a == X_b$	$X_a - X_b \leq 0$, $X_b - X_a \leq 0$	`add(b, a, 0), add(a, b, 0);`

```cpp
// s[r] - s[l - 1] >= x -> s[r] - s[l - 1] <= r - l + 1 - x ->s[r] <= s[l - 1]
+ r - l + 1 - x
int main(void) {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, q;
    std::cin >> n >> q;
    std::vector adj(n + 2, std::vector<std::pair<int, int>>());
    for (int i = 1; i <= q; i += 1) {
        int l, r, x;
        std::cin >> l >> r >> x;
        adj[l - 1].push_back({r, r - l + 1 - x});
    }
    for (int i = 1; i <= n; i += 1) {
        adj[i - 1].push_back({i, 1});
        adj[i].push_back({i - 1, 0});
    }
    std::priority_queue<std::pair<int, int>, std::vector<std::pair<int, int>>,
std::greater<>>Q;
    std::vector<int>dis(n + 1, -1);
    Q.emplace(dis[0] = 0, 0);
    while (!Q.empty()) {
        auto [d, u] = Q.top();
        Q.pop();
        if (d != dis[u]) {
            continue;
        }
        for (const auto & [v, w] : adj[u]) {
            if (dis[v] == -1 or dis[v] > dis[u] + w) {
                Q.emplace(dis[v] = dis[u] + w, v);
            }
        }
    }
    for (int i = 1; i <= n; i += 1) {
        std::cout << ((dis[i] - dis[i - 1]) ^ 1) << " \n"[i == n];
    }
    return 0;
}
```

Pmod

当出现形如「给定 n 个整数，求这 n 个整数能拼凑出多少的其他整数（n 个整数可以重复取）」\以及「给定 n 个整数，求这 n 个整数不能拼凑出的最小（最大）的整数」，\或者「至少要拼几次才能拼出模 K 余 p 的数」的问题时可以使用同余最短路的方法。

同余最短路利用同余来构造一些状态，可以达到优化空间复杂度的目的。

类比 '差分约束' 方法，利用同余构造的这些状态可以看作单源最短路中的点。\同余最短路的状态转移通常是这样的 $f(i+y)=f(i)+y$，\类似单源最短路中 $f(v)=f(u)+\text{edge}(u,v)$。\ 题目大意：给定 n，求 n 的倍数中，数位和最小的那一个的数位和。（$1 \le n \le 10^5$）观察到任意一个正整数都可以从 1 开始，按照某种顺序执行乘 10、加 1 的操作，最终得到，而其中加 1 操作的次数就是这个数的数位和。这提示我们使用最短路。

对于所有 $0 \le k \le n-1$，从 k 向 $10k$ 连边权为 0 的边；从 k 向 $k+1$ 连边权为 1 的边。（点的编号均在模 n 意义下）

每个 n 的倍数在这个图中都对应了 1 号点到 0 号点的一条路径，求出 1 到 0 的最短路即可。\某些路径不合法（如连续走了 10 条边权为 1 的边），\但这些路径产生的答案一定不优，不影响答案。

```cpp
auto main() ->int {
```

```
    int k;
    std::cin >> k;
    std::vector<std::vector<std::pair<int, int>>>adj(k);
    for (int u = 0; u <= k - 1; u += 1) {
        adj[u].push_back({10 * u % k, 0});
        adj[u].push_back({(u + 1) % k, 1});
    }
    std::priority_queue<std::pair<int, int>, std::vector<std::pair<int, int>>,
std::greater<>>q;
    std::vector<int>d(k, -1);
    q.push({d[1] = 0, 1});
    while (!q.empty()) {
        auto [f, u] = q.top();
        q.pop();
        if (f != d[u]) {
            continue;
        }
        for (const auto & [v, w] : adj[u]) {
            if (d[v] == -1 || d[v] > d[u] + w) {
                q.push({d[v] = d[u] + w, v});
            }
        }
    }
    std::cout << d[0] + 1 << '\n';
    return 0;
}
```

题目大意：给定x，y，z，h，对于$k \in [1,h]$，有多少个k能够满足$ax+by+cz=k$。（$0 \le a,b,c$，$1 \le x,y,z$

首先可以将 `h` 减去 `1`，同时起始楼层设为 `0`。

设 d_i 为能够到达的最低的楼层，其中 `i mod x = i`。

则有：

- `i → y (i + y) mod x`
- `i → z (i + z) mod x`

像这样建图后，d_i 就相当于 `0 → i` 的最短路，Dijkstra 即可。

最后统计时，对于 $d_i \le h$，有贡献 $\lfloor (h-d_i)/x \rfloor + 1$。

总时间复杂度 `O(n log n)`。

```
auto main() ->int {
    i64 h;
    int x, y, z;
    std::cin >> h >> x >> y >> z;
    if (x == 1 || y == 1 || z == 1 || h == 1) {
        std::cout << h << '\n';
        return 0;
    }
    h -= 1;
    std::vector<std::vector<std::pair<int, int>>>adj(x);
    for (int u = 0; u < x; u += 1) {
        adj[u].push_back({(u + y) % x, y});
        adj[u].push_back({(u + z) % x, z});
    }
    std::vector<i64>d(x, -1);
    std::priority_queue<std::pair<i64, int>, std::vector<std::pair<i64, int>>,
std::greater<>>q;
    q.push({d[0] = 0, 0});
```

```
    while (!q.empty()) {
        auto [f, u] = q.top();
        q.pop();
        if (f != d[u]) {
            continue;
        }
        for (const auto & [v, w] : adj[u]) {
            if (d[v] == -1 || d[v] > d[u] + w) {
                q.push({d[v] = d[u] + w, v});
            }
        }
    }
    i64 res = 0;
    for (int u = 0; u < x; u += 1) {
        if (h >= d[u] && d[u] != -1) {
            res += (h - d[u]) / x + 1;
        }
    }
    std::cout << res << '\n';
    return 0;
}
```

Point

在二维平面上，给定一组点 $((x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n))$，它们的重心（或质心）可以通过以下公式计算：

$$\text{重心}(G_x,G_y)=\left(\frac{x_1+x_2+\ldots+x_n}{n},\frac{y_1+y_2+\ldots+y_n}{n}\right)$$

这里，(G_x)是重心的x坐标，(G_y)是重心的y坐标，(n)是点的数量。

其实可以在 $\mathrm{O}(\log n)$ 级别的复杂度求解一个点是否在凸包内，先取凸包左下角第一个点对其他点和待求点进行偏移，方便进行判断。

将偏移后的凸包按极角进行排序，然后就可以直接根据极角序二分找到第一个大于等于待求点的位置然后通过叉积判断即可。

给定一个随机点集，它的凸包的期望顶点数为 $\log_2 n$。

给定 n 个二维向量 (x_i,y_i)，从中选择若干个，使得所选的向量之和的模长最大。

```
std::vector<std::vector<P>> p;
for (int i = 1; i <= n; i += 1) {
    i64 a, b, c, d;
    std::cin >> a >> b >> c >> d;
    if (a == b && c == d) {
        continue;
    }
    p.push_back({P(), P(a - b, c - d)});
}
if (p.empty()) {
    std::cout << 0 << '\n';
    return 0;
}
n = p.size();
auto dfs = [&](this auto &&dfs, int l, int r) -> std::vector<P> {
    if (l == r) {
        return p[l];
    }
    int mid = (l + r) >> 1;
    auto a = dfs(l, mid), b = dfs(mid + 1, r);
    return minkowskisum(a, b);
```

```
};
auto r = dfs(0, n - 1);
i128 h = 0;
for (const auto &c: r) {
    chmax(h, i128(c.x) * c.x + i128(c.y) * c.y);
}
```

```
auto c = minkowskisum(a, b);
std::sort(c.begin(), c.end());
P p = c[0];
for (int i = 1; i < c.size(); i += 1) {
    c[i] -= p;
}
std::sort(c.begin() + 1, c.end(), [&](const auto &u, const auto &v) {
    return compute(u, v);
});
while (q--) {
    int dx, dy;
    std::cin >> dx >> dy;
    P x = P(dx, dy) - p;
    if (cross(c.back(), x) > 0 || cross(x, c[1]) > 0) {
        std::cout << 0 << '\n';
        continue;
    }
    int i = std::lower_bound(c.begin() + 1, c.end(), x, [&](const auto &u,
const auto &v) {
                return compute(u, v);
            }) -
            c.begin();
    if (i == c.size() || i == 1) {
        std::cout << 0 << '\n';
        continue;
    }
    int j = (i - 1 + c.size()) % c.size();
    std::cout << (cross(c[j] - x, c[i] - x) >= 0) << '\n';
}
```

```
template<class T>
struct Point {
    T x;
    T y;
    Point(const T &x_ = 0, const T &y_ = 0) : x(x_), y(y_) {}

    template<class U>
    operator Point<U>() {
        return Point<U>(U(x), U(y));
    }
    Point &operator+=(const Point &p) & {
        x += p.x;
        y += p.y;
        return *this;
    }
    Point &operator-=(const Point &p) & {
        x -= p.x;
        y -= p.y;
        return *this;
    }
    Point &operator*=(const T &v) & {
        x *= v;
        y *= v;
```

```
        return *this;
    }
    Point &operator/=(const T &v) & {
        x /= v;
        y /= v;
        return *this;
    }
    Point operator-() const {
        return Point(-x, -y);
    }
    constexpr friend Point operator+(Point a, const Point &b) {
        return a += b;
    }
    constexpr friend Point operator-(Point a, const Point &b) {
        return a -= b;
    }
    constexpr friend Point operator*(Point a, const T &b) {
        return a *= b;
    }
    constexpr friend Point operator/(Point a, const T &b) {
        return a /= b;
    }
    constexpr friend Point operator*(const T &a, Point b) {
        return b *= a;
    }
    constexpr friend bool operator==(const Point &a, const Point &b) {
        return a.x == b.x && a.y == b.y;
    }
    constexpr friend bool operator<(const Point& a, const Point& b) {
        return (a.x == b.x) ? a.y < b.y : a.x < b.x;
    }
    friend std::istream &operator>>(std::istream &is, Point &p) {
        return is >> p.x >> p.y;
    }
    friend std::ostream &operator<<(std::ostream &os, const Point &p) {
        return os << "(" << p.x << ", " << p.y << ")";
    }
};

template<class T>
struct Line {
    Point<T> a;
    Point<T> b;
    Line(const Point<T> &a_ = Point<T>(), const Point<T> &b_ = Point<T>()) :
a(a_), b(b_) {}
};

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

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

template<class T>
T square(const Point<T> &p) {
    return dot(p, p);
```

```
}

template<class T>
f64 length(const Point<T> &p) {
    return sqrtl(square(p));
}

template<class T>
f64 length(const Line<T> &l) {
    return length(l.a - l.b);
}

template<class T>
Point<T> normalize(const Point<T> &p) {
    return p / length(p);
}

template<class T>
bool parallel(const Line<T> &l1, const Line<T> &l2) {
    return cross(l1.b - l1.a, l2.b - l2.a) == 0;
}

template<class T>
f64 distance(const Point<T> &a, const Point<T> &b) {
    return length(a - b);
}

template<class T>
f64 distancePL(const Point<T> &p, const Line<T> &l) {
    return std::abs(cross(l.a - l.b, l.a - p)) / length(l);
}

template<class T>
f64 distancePS(const Point<T> &p, const Line<T> &l) {
    if (dot(p - l.a, l.b - l.a) < 0) {
        return distance(p, l.a);
    }
    if (dot(p - l.b, l.a - l.b) < 0) {
        return distance(p, l.b);
    }
    return distancePL(p, l);
}

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

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

template<class T>
bool compute(const Point<T>& a, const Point<T>& b) {
    if (sgn(a) == sgn(b)) {
        return cross(a, b) > 0;
    }
    return sgn(a) > sgn(b);
}
```

```cpp
template<class T>
f64 angle(const Point<T>& p) {
    return std::atan2(p.y, p.x);
}

template<class T>
f64 angle(const Line<T>& l) {
    return angle(l.b - l.a);
}

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

template<class T>
Point<T> lineIntersection(const Line<T> &l1, const 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(const Point<T> &p, const Line<T> &l) {
    return cross(p - l.a, l.b - l.a) == 0 and std::min(l.a.x, l.b.x) <= p.x and
p.x <= std::max(l.a.x, l.b.x) and std::min(l.a.y, l.b.y) <= p.y and p.y <=
std::max(l.a.y, l.b.y);
}

template<class T>
bool pointOnLine(const Point<T> &p, const Line<T> &l) {
    return pointOnSegment(p, l) or pointOnSegment(l.a, Line(p, l.b)) or
pointOnSegment(l.b, Line(p, l.a));
}

template<class T>
bool pointInPolygon(const Point<T> &a, const 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++) {
        const auto &u = p[i];
        const auto &v = p[(i + 1) % n];
        if (u.x < a.x and v.x >= a.x and pointOnLineLeft(a, Line(v, u))) {
            t ^= 1;
        }
        if (u.x >= a.x and v.x < a.x and pointOnLineLeft(a, Line(u, v))) {
            t ^= 1;
        }
    }
    return t == 1;
}

// 0 : not intersect
// 1 : strictly intersect
// 2 : overlap
```

```cpp
// 3 : intersect at endpoint
template<class T>
std::tuple<int, Point<T>, Point<T>> segmentIntersection(const Line<T> &l1,
const 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 {
            const auto &maxx1 = std::max(l1.a.x, l1.b.x);
            const auto &minx1 = std::min(l1.a.x, l1.b.x);
            const auto &maxy1 = std::max(l1.a.y, l1.b.y);
            const auto &miny1 = std::min(l1.a.y, l1.b.y);
            const auto &maxx2 = std::max(l2.a.x, l2.b.x);
            const auto &minx2 = std::min(l2.a.x, l2.b.x);
            const auto &maxy2 = std::max(l2.a.y, l2.b.y);
            const 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};
            }
        }
    }
    const auto &cp1 = cross(l2.a - l1.a, l2.b - l1.a);
    const auto &cp2 = cross(l2.a - l1.b, l2.b - l1.b);
    const auto &cp3 = cross(l1.a - l2.a, l1.b - l2.a);
    const auto &cp4 = cross(l1.a - l2.b, l1.b - l2.b);
    if ((cp1 > 0 and cp2 > 0) or (cp1 < 0 and cp2 < 0) or (cp3 > 0 and cp4 > 0)
or (cp3 < 0 and cp4 < 0)) {
        return {0, Point<T>(), Point<T>()};
    }
    Point p = lineIntersection(l1, l2);
    if (cp1 != 0 and cp2 != 0 and cp3 != 0 and cp4 != 0) {
        return {1, p, p};
    } else {
        return {3, p, p};
    }
}

template<class T>
f64 distanceSS(const Line<T> &l1, const Line<T> &l2) {
    if (std::get<0>(segmentIntersection(l1, l2)) != 0) {
        return 0.0;
```

```
    }
    return std::min({distancePS(l1.a, l2), distancePS(l1.b, l2),
distancePS(l2.a, l1), distancePS(l2.b, l1)});
}

template<class T>
bool segmentInPolygon(const Line<T> &l, const 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++) {
        const auto &u = p[i];
        const auto &v = p[(i + 1) % n];
        const auto &w = p[(i + 2) % n];
        const 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) and v != l.a and v != l.b) {
                if (cross(v - u, w - v) > 0) {
                    return false;
                }
            }
        } else {
            if (p1 != u and p1 != v) {
                if (pointOnLineLeft(l.a, Line(v, u)) or pointOnLineLeft(l.b,
Line(v, u))) {
                    return false;
                }
            } else if (p1 == v) {
                if (l.a == v) {
                    if (pointOnLineLeft(u, l)) {
                        if (pointOnLineLeft(w, l) and pointOnLineLeft(w,
Line(u, v))) {
                            return false;
                        }
                    } else {
                        if (pointOnLineLeft(w, l) or 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)) and
pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    } else {
                        if (pointOnLineLeft(w, Line(l.b, l.a)) or
pointOnLineLeft(w, Line(u, v))) {
```

```
                            return false;
                        }
                    }
                } else {
                    if (pointOnLineLeft(u, l)) {
                        if (pointOnLineLeft(w, Line(l.b, l.a)) or
pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    } else {
                        if (pointOnLineLeft(w, l) or 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(), [&](const auto & l1, const auto & l2)
{
        const auto &d1 = l1.b - l1.a;
        const 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 (const auto &l : lines) {
        if (ls.empty()) {
            ls.push_back(l);
            continue;
        }
        while (!ps.empty() and !pointOnLineLeft(ps.back(), l)) {
            ps.pop_back();
            ls.pop_back();
        }
        while (!ps.empty() and !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() and !pointOnLineLeft(ps.back(), ls[0])) {
        ps.pop_back();
        ls.pop_back();
    }
    if (ls.size() <= 2) {
        return {};
    }
    ps.push_back(lineIntersection(ls[0], ls.back()));
    return std::vector(ps.begin(), ps.end());
}

template<class T>
T PolygonArea(const std::vector<Point<T>> &p) {
    T res = T(0);
    int n = p.size();
    for (int i = 0; i < n; i += 1) {
        res += cross(p[i], p[(i + 1) % n]);
    }
    return std::abs(res);
}

template<class T>
std::vector<Point<T>> getHull(std::vector<Point<T>> p) {
    std::vector<Point<T>>h, l;
    std::sort(p.begin(), p.end(), [&](const auto & a, const auto & b) {
        return a.x == b.x ? a.y < b.y : a.x < b.x;
    });
    p.erase(std::unique(p.begin(), p.end()), p.end());
    if (p.size() <= 1) {
        return p;
    }
    for (const auto & a : p) {
        while ((int)h.size() > 1 and cross(a - h.back(), a - h[(int)h.size() -
2]) <= 0) {
            h.pop_back();
        }
        while ((int)l.size() > 1 and cross(a - l.back(), a - l[(int)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;
}

auto getHull(std::vector<P> p) {
    if (p.size() <= 1) {
        return p;
    }
    int n = p.size();
    for (int i = 1; i < n; i += 1) {
        if (p[i].y < p[0].y || p[i].y == p[0].y && p[i].x < p[0].x) {
            std::swap(p[i], p[0]);
```

```
        }
    }

    std::sort(p.begin() + 1, p.end(), [&](const auto & u, const auto & v) {
        auto a = u - p[0], b = v - p[0];
        if (sgn(a) == sgn(b)) {
            return cross(a, b) > 0;
        }
        return sgn(a) < sgn(b);
    });

    std::vector<P>stk {p[0]};
    for (int i = 1; i < n; i += 1) {
        while (stk.size() > 1 && cross(stk.back() - stk[stk.size() - 2], p[i] -
stk.back()) <= 0) {
            stk.pop_back();
        }
        stk.push_back(p[i]);
    }
    return stk;
}

template<class T>
std::tuple<T, Point<T>, Point<T>> getLongest(const std::vector<Point<T>>& ret)
{
    std::vector<Point<T>> p = getHull(ret);
    int n = p.size();

    T res = T(0);
    Point<T> a = Point<T>(), b = Point<T>();
    int x = 0, y = 0;
    for (int i = 0; i < n; i += 1) {
        if (p[i].y < p[x].y)x = i;
        if (p[i].y > p[y].y)y = i;
    }
    res = square(p[x] - p[y]);
    a = p[x], b = p[y];
    int i = x, j = y;
    do {
        if (cross(p[(i + 1) % n] - p[i], p[(j + 1) % n] - p[j]) < 0) {
            i = (i + 1) % n;
        } else {
            j = (j + 1) % n;
        }
        if (square(p[i] - p[j]) > res) {
            res = square(p[i] - p[j]);
            a = p[i], b = p[j];
        }
    } while (i != x or j != y);
    return {res, a, b};
}

template<class T>
std::tuple<T, Point<T>, Point<T>> getClostest(std::vector<Point<T>> p) {
    std::sort(p.begin(), p.end(), [&](const auto & a, const auto & b) {
        return a.x == b.x ? a.y < b.y : a.x < b.x;
    });
    T res = std::numeric_limits<T>::max();
    Point<T> a = Point<T>(), b = Point<T>();
    int n = p.size();
```

```
    auto update = [&](const Point<T>& u, const Point<T>& v) {
        if (res > square(u - v)) {
            res = square(u - v);
            a = u;
            b = v;
        }
    };

    auto s = std::multiset < Point<T>, decltype([](const Point<T>& u, const
Point<T>& v) {
        return u.y == v.y ? u.x < v.x : u.y < v.y;
    }) > ();
    std::vector<typename decltype(s)::const_iterator>its(n);
    for (int i = 0, f = 0; i < n; i += 1) {
        while (f < i and (p[i] - p[f]).x * (p[i] - p[f]).x >= res) {
            s.erase(its[f++]);
        }
        auto u = s.upper_bound(p[i]); {
            auto t = u;
            while (true) {
                if (t == s.begin()) {
                    break;
                }
                t = std::prev(t);
                update(*t, p[i]);
                if ((p[i] - *t).y * (p[i] - *t).y >= res) {
                    break;
                }
            }
        }{
            auto t = u;
            while (true) {
                if (t == s.end()) {
                    break;
                }
                if ((p[i] - *t).y * (p[i] - *t).y >= res) {
                    break;
                }
                update(*t, p[i]);
                t = std::next(t);
            }
        }
        its[i] = s.emplace_hint(u, p[i]);
    }

    return {res, a, b};
}

template<class T>
std::pair<T, std::vector<Point<T>>> rectCoverage(const std::vector<Point<T>>&
p) {
    T res = std::numeric_limits<T>::max();
    std::vector<Point<T>>rect;
    std::array<int, 4>pos {};
    int n = p.size();
    if (n < 3) {
        return std::pair(res, rect);
    }
    for (int i = 0, r = 1, j = 1, q = 0; i < n; i += 1) {
```

```
        while (cross(p[(i + 1) % n] - p[i], p[(r + 1) % n] - p[i]) >=
cross(p[(i + 1) % n] - p[i], p[r] - p[i])) {
            r = (r + 1) % n;
        }
        while (dot(p[(i + 1) % n] - p[i], p[(j + 1) % n] - p[i]) >= dot(p[(i +
1) % n] - p[i], p[j] - p[i])) {
            j = (j + 1) % n;
        }
        if (i == 0) {
            q = j;
        }
        while (dot(p[(i + 1) % n] - p[i], p[(q + 1) % n] - p[i]) <= dot(p[(i +
1) % n] - p[i], p[q] - p[i])) {
            q = (q + 1) % n;
        }
        T d = square(p[i] - p[(i + 1) % n]);
        T area = cross(p[(i + 1) % n] - p[i], p[r] - p[i]) * (dot(p[(i + 1) %
n] - p[i], p[j] - p[i]) - dot(p[(i + 1) % n] - p[i], p[q] - p[i])) / d;
        if (area < res) {
            res = area;
            pos[0] = r;
            pos[1] = j;
            pos[2] = q;
            pos[3] = i;
        }
    }
    const auto& [r, j, q, i] = pos;
    Line<T> l1 = Line(p[i], p[(i + 1) % n]);
    Point t = p[(i + 1) % n] - p[i];
    Line<T> l2 = Line(p[r], p[r] + t);
    t = rotate(t);
    Line<T> l3 = Line(p[j], p[j] + t);
    Line<T> l4 = Line(p[q], p[q] + t);

    rect.push_back(lineIntersection(l1, l3));
    rect.push_back(lineIntersection(l1, l4));
    rect.push_back(lineIntersection(l2, l3));
    rect.push_back(lineIntersection(l2, l4));

    rect = getHull(rect);
    return std::pair(res, rect);
}

template<class T>
Point<T> triangleHeart(const Point<T>& A, const Point<T>& B, const Point<T>& C)
{
    return (A * square(B - C) + B * square(C - A) + C * square(A - B)) /
(square(B - C) + square(C - A) + square(A - B));
}

template<class T>
Point<T> circumcenter(const Point<T>& a, const Point<T>& b, const Point<T>& c)
{
    T D = 2 * (a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.y - b.y));
    assert(D != 0);
    Point<T> p;
    p.x = ((square(a) * (b.y - c.y) + (square(b) * (c.y - a.y)) + (square(c) *
(a.y - b.y)))) / D;
    p.y = ((square(a) * (c.x - b.x) + (square(b) * (a.x - c.x)) + (square(c) *
(b.x - a.x)))) / D;
```

```
    return p;
}

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

template<class T>
std::pair<Point<T>, T> cirlCoverage(std::vector<Point<T>> p) {
    for (int t = 0; t < 7; t += 1) {
        std::shuffle(p.begin(), p.end(), rng);
    }
    int n = p.size();
    Point<T> o = p[0];
    T r = T(0);
    for (int i = 1; i < n; i += 1) {
        if (length(o - p[i]) > r) {
            o = p[i];
            r = T(0);
            for (int j = 0; j < i; j += 1) {
                if (length(o - p[j]) > r) {
                    o = (p[i] + p[j]) / T(2);
                    r = length(o - p[i]);
                    for (int k = 0; k < j; k += 1) {
                        if (length(o - p[k]) > r) {
                            o = circumcenter(p[i], p[j], p[k]);
                            r = length(o - p[i]);
                        }
                    }
                }
            }
        }
    }
    return std::pair(o, r);
}

template <class T>
std::vector<Point<T>> minkowskisum(std::vector<Point<T>>& a,
std::vector<Point<T>>& b) {
    std::vector<Point<T>> c;
    std::vector<Line<T>>e(a.size() + b.size()), e1(a.size()), e2(b.size());
    const auto cmp = [&](const Line<T>& l1, const Line<T>& l2) {
        const auto a = l1.b - l1.a, b = l2.b - l2.a;
        if (sgn(a) == sgn(b)) {
            return cross(a, b) > 0;
        }
        return sgn(a) < sgn(b);
    };
    for (int i = 0; i < a.size(); i += 1) {
        e1[i] = Line(a[i], a[(i + 1) % a.size()]);
    }
    for (int i = 0; i < b.size(); i += 1) {
        e2[i] = Line(b[i], b[(i + 1) % b.size()]);
    }
    std::rotate(e1.begin(), std::min_element(e1.begin(), e1.end(), cmp),
e1.end());
    std::rotate(e2.begin(), std::min_element(e2.begin(), e2.end(), cmp),
e2.end());
    std::merge(e1.begin(), e1.end(), e2.begin(), e2.end(), e.begin(), cmp);
    const auto ok = [&](std::vector<Point<T>>& r, const Point<T>& u) {
        return cross(r.end()[-1] - r.end()[-2], u - r.end()[-1]) == 0 &&
dot(r.end()[-1] - r.end()[-2], u - r.end()[-1]) >= 0;
```

```
    };
    Point<T> u = e1[0].a + e2[0].a;
    for (const auto & v : e) {
        while (c.size() > 1 && ok(c, u)) {
            c.pop_back();
        }
        c.push_back(u);
        u = u + v.b - v.a;
    }
    if (c.size() > 1 && ok(c, u)) {
        c.pop_back();
    }
    return c;
}

template<class F>
f64 integral(f64 l, f64 r, const F& f) {
    static constexpr f64 eps = 1e-9;
    auto simpson = [&](f64 l, f64 r) {
        return (f(l) + 4 * f((l + r) / 2) + f(r)) * (r - l) / 6;
    };
    auto func = [&](auto && func, f64 l, f64 r, f64 eps, f64 st) {
        f64 mid = (l + r) / 2;
        f64 sl = simpson(l, mid), sr = simpson(mid, r);
        if (std::abs(sl + sr - st) <= 15 * eps) {
            return (sl + sr + (sl + sr - st) / 15);
        }
        return func(func, l, mid, eps / 2, sl) + func(func, mid, r, eps / 2,
sr);
    };
    return func(func, l, r, eps, simpson(l, r));
}

using P = Point<i64>;
using L = Line<i64>;
```

scanline

```
template<class T = i64>
struct SegmentTree {
    int n;
    std::vector<T>info;
    std::vector<T>tag;
    std::vector<T>X;
    SegmentTree(const auto & xs) {
        X = xs;
        this->n = (int)xs.size() - 2;
        info.assign(8 * n, T());
        tag.assign(8 * n, T());
    }
    void pull(int p, int l, int r) {
        if (tag[p] != 0) {
            info[p] = X[r + 1] - X[l];
        } else {
            info[p] = info[p << 1 | 1] + info[p << 1];
        }
    }
    void update(int p, int l, int r, int L, int R, const T& val) {
        if (X[r + 1] <= L or X[l] >= R) {
            return;
```

```
        }
        if (L <= X[l] and X[r + 1] <= R) {
            tag[p] += val;
            pull(p, l, r);
            return;
        }
        int mid = l + r >> 1;
        update(p << 1, l, mid, L, R, val);
        update(p << 1 | 1, mid + 1, r, L, R, val);
        pull(p, l, r);
    }
    void update(int l, int r, const T& val) {
        update(1, 1, n, l, r, val);
    }
    T Query() {
        return info[1];
    }
};
struct Node {
    i64 y, l, r;
    int type;
    Node(i64 y, i64 l, i64 r, int type): y(y), l(l), r(r), type(type) {}
    friend bool operator<(const Node& lsh, const Node& rsh) {
        return lsh.y < rsh.y;
    }
};
i64 getArea(const auto & ret) {
    std::vector<Node>e;
    std::vector<i64>xs{0};
    for (const auto &[x1, y1, x2, y2] : ret) {
        e.push_back({y1, x1, x2, 1});
        e.push_back({y2, x1, x2, -1});
        xs.push_back(x1);
        xs.push_back(x2);
    }
    std::sort(e.begin(), e.end());
    std::sort(xs.begin() + 1, xs.end());
    xs.erase(std::unique(xs.begin() + 1, xs.end()), xs.end());
    SegmentTree seg(xs);
    i64 res = 0;
    for (int i = 0; i + 1 < (int)e.size(); i += 1) {
        seg.update(e[i].l, e[i].r, e[i].type);
        res += seg.Query() * (e[i + 1].y - e[i].y);
    }
    return res;
}
```

3D

```
template <class T, class G>
struct BaseVector3 {
    T x, y, z;
    constexpr BaseVector3() : BaseVector3(T{}, T{}, T{}) {}
    constexpr BaseVector3(T x, T y, T z)
        : x{x}, y{y}, z{z} {}
    constexpr BaseVector3 operator-(BaseVector3 a) const {
        return BaseVector3(x - a.x, y - a.y, z - a.z);
    }
    constexpr BaseVector3 operator-() const {
        return BaseVector3(-x, -y, -z);
```

```
    }
    constexpr BaseVector3 operator+(BaseVector3 a) const {
        return BaseVector3(x + a.x, y + a.y, z + a.z);
    }
    constexpr G operator*(BaseVector3 a) const {
        return x * a.x + y * a.y + z * a.z;
    }
    constexpr BaseVector3 operator%(BaseVector3 a) const {
        return BaseVector3(
            y * a.z - a.y * z,
            a.x * z - x * a.z,
            x * a.y - a.x * y);
    }
    constexpr friend BaseVector3 operator*(T d, BaseVector3 a) {
        return BaseVector3(d * a.x, d * a.y, d * a.z);
    }
    constexpr friend bool sameLine(BaseVector3 a, BaseVector3 b) {
        return dis2(a % b) == 0;
    }
    constexpr friend bool sameDir(BaseVector3 a, BaseVector3 b) {
        return samLine(a, b) and a * b >= 0;
    }
    // 字典序
    constexpr bool operator<(BaseVector3 a) const {
        if (x != a.x)
            return x < a.x;
        if (y != a.y)
            return y < a.y;
        return z < a.z;
    }
    constexpr bool operator==(BaseVector3 a) const {
        return x == a.x and y == a.y and z == a.z;
    }
    constexpr friend auto &operator>>(std::istream &is, BaseVector3 &p) {
        return is >> p.x >> p.y >> p.z;
    }
    constexpr friend auto &operator<<(std::ostream &os, BaseVector3 p) {
        return os << '(' << p.x << ", " << p.y << ", " << p.z << ')';
    }
};

template <class T, class G>
G dis2(BaseVector3<T, G> a, BaseVector3<T, G> b = BaseVector3<T, G>{}) {
    auto p = a - b;
    return p * p;
}
template <class T, class G>
auto dis(BaseVector3<T, G> a, BaseVector3<T, G> b = BaseVector3<T, G>{}) {
    return sqrtl(dis2(a, b));
}

template <class T, class G>
auto area(BaseVector3<T, G> a, BaseVector3<T, G> b = BaseVector3<T, G>{}) {
    return dis(a % b) / 2;
}

template <class T>
struct TrianglePatch {
    std::array<T, 3> vertexs;
```

```
    constexpr TrianglePatch() : vertexs{} {}

    constexpr TrianglePatch(T u, T v, T w)
        : vertexs({u, v, w}) {}

    constexpr T &operator[](int i) {
        return vertexs[i];
    }

    constexpr T normal() const {
        return (vertexs[1] - vertexs[0]) % (vertexs[2] - vertexs[0]);
    }

    constexpr T vectorTo(T a) const {
        return a - vertexs[0];
    }

    constexpr auto area() const {
        return ::area(vertexs[1] - vertexs[0], vertexs[2] - vertexs[0]);
    }

    constexpr bool coPlane(T a) const {
        return vectorTo(a) * normal() == 0;
    }

    constexpr bool left(T a) const {
        return vector(a) * normal() > 0;
    }

    constexpr bool contains(T a) const {
        if (!coPlane(a)) {
            return false;
        }
        T norm = normal();
        int leftCnt = 0;
        for (int i = 0; i < 3; i++) {
            T edge = vertexs[(i + 1) % 3] - vertexs[i];
            T to_a = a - vertexs[i];

            leftCnt += sameDir(edge % to_a, norm);
        }
        return leftCnt == 3;
    }
};

using Point = BaseVector3<Float, Float>;
using Vector = Point;

using PS = std::vector<Point>;
using PatchS = std::vector<TrianglePatch<Point>>;

PatchS convex(PS ps) {
    std::sort(begin(ps), end(ps));
    ps.erase(std::unique(ps.begin(), ps.end()), ps.end());
    // 去除重复点
    const int n = ps.size();
    if (n < 4) {
        return {};
    }
    std::vector vis(n, std::vector<int>(n, -1));
```

```
    using IndexPatch = std::array<int, 3>;
    std::vector<IndexPatch> pls(2);
    pls[0] = IndexPatch{0, 1, 2};
    pls[1] = IndexPatch{0, 2, 1};
    auto isAbove = [&](Point p, IndexPatch &pl) {
        auto vec = (ps[pl[1]] - ps[pl[0]]) % (ps[pl[2]] - ps[pl[0]]);
        return vec * (p - ps[pl[0]]) >= 0;
    };
    for (int i = 3; i < n; i++) {
        std::vector<IndexPatch> res, del;
        for (int j = 0; j < size(pls); j++) {
            if (!isAbove(ps[i], pls[j])) {
                res.push_back(pls[j]);
            } else {
                del.push_back(pls[j]);
                for (int k = 0; k < 3; k++) {
                    int r = (k + 1) % 3;
                    vis[pls[j][k]][pls[j][r]] = i;
                }
            }
        }
        for (auto pl : del) {
            for (int k = 0; k < 3; k++) {
                int r = (k + 1) % 3;
                if (vis[pl[k]][pl[r]] == i and vis[pl[r]][pl[k]] != i) {
                    res.push_back({pl[k], pl[r], i});
                }
            }
        }
        std::swap(res, pls);
    }
    PatchS ans;
    for (auto pl : pls) {
        ans.push_back({ps[pl[0]], ps[pl[1]], ps[pl[2]]});
    }
    return ans;
}
```

Math

一个长度为 n 的排列的期望逆序对为 $\frac{n\times(n-1)}{4}$

$\sum_{k=1}^{n} k^2 = \frac{n\times(n+1)\times(2n+1)}{6}$

Polynomial

对于 $a \le 1, b \le 1$ 有 $a \oplus b = a(1-b) + b(1-a) = a + b - 2ab$

primes whose root is 3

$V <= 1E9 : 1004535809，469762049，998244353$
$V <= 1E15 : 1337006139375617$
$V <= 4E18 : 4179340454199820289$

$998244353 : 3, 1E9 + 7 : 5$

求解如下式子：$res[k] = \sum_{i=0}^{n-k} f_i \times f_{i+k}$

令$g_i = f_{n-i}, res[k] = h[n-k] = \sum_{i=0}^{n-k} f_i \times g_{n-i-k}$, 直接$FFT$即可

根据这个技巧，可以求解一个序列的区间和的所有可能的出现次数

$\sum_{i=l}^{r} a_i = s_r - s_{l-1}$, 那么记$k = s_i - s_j$, f_i为前缀和数组s中i的出现次数，则$cnt[k] = \sum_{i=0}^{\max S-k} f_i \times f_{i+k}$

记 $g_i = f_{\max S-i}$, 则 $cnt[k] = p[\max S - k] = \sum_{i=0}^{\max S-k} f_i \times g_{\max S-i-k}$, NTT即可，注意要特殊处理区间和为0的情况。

求解两个01串的最小汉明距离

naive solution: $\sum_{j=0}^{|T|-1} S_{i+j} \oplus T_j$

let $C_i = \sum_{i=j+k} S_j \oplus T_k$

$T_i' = T_{n-i-1}, \sum_{j=0}^{|T|-1} S_{i+j} \oplus T_j = S_{i+j} \oplus T'_{n-j-1} = S_{i+j}(1 - T'_{n-j-1}) + (1 - S_{i+j})T'_{n-j-1}$

$C_i = \sum_{i=j+k} S_j(1 - T_k) + (1 - S_j)T_k,\ m-1 \le j+k \le n-1,\ res = \min_{i=m-1}^{n-1} C_i$

```
//怎么找一个质数的原根
auto factorize(int x) {
    std::vector<int>d;
    for (int y = 2; y <= x; y += 1) {
        if (x % y == 0) {
            d.push_back(y);
            while (x % y == 0) {
                x /= y;
            }
        }
    }
    if (x > 1) {
        d.push_back(x);
    }
    return d;
}
for (int r = 2; ; r += 1) {
    bool ok = true;
    // v : {P - 1} 的质因子
    for (const auto & c : v) {
        if (power(g, (P - 1) / g, P) == 1) {
            ok = false;
        }
    }
    if (ok) {
        std::cout << r << '\n';
    }
}
// NTT
template <class T>
constexpr T power(T a, i64 b) {
    T r = 1;
    for (; b != 0; b >>= 1, a = a * a) {
        if (b & 1) {
            r = r * a;
        }
    }
    return r;
}
template <class T, T P>
class ModuloInteger {
    T x;
    static T Mod;
```

```
    using i64 = long long;
    using i128 = __int128;
    static constexpr int multiply(int a, int b, const int Mod) {
        return i64(a) * b % Mod;
    }
    static constexpr i64 multiply(i64 a, i64 b, const i64 Mod) {
        return static_cast<i128>(a) * b % Mod;
    }
    T norm(T x) const {
        return (x < 0 ? x + getMod() : (x >= getMod() ? x - getMod() : x));
    }
public:
    ModuloInteger() : x{} {}
    ModuloInteger(i64 x) : x{norm(x % getMod())} {}
    static constexpr T getMod() {
        return (P > 0 ? P : Mod);
    }
    static void setMod(T m) {
        Mod = m;
    }
    T val() const {
        return x;
    }
    explicit operator T() const {
        return x;
    }
    ModuloInteger operator-() const {
        return ModuloInteger(getMod() - x);
    }
    ModuloInteger inv() const {
        assert(*this->val() != 0);
        return ModuloInteger(power(*this, getMod() - 2));
    }
    ModuloInteger &operator*=(ModuloInteger& rsh) & {
        x = multiply(x, rsh.val(), getMod());
        return *this;
    }
    ModuloInteger &operator+=(ModuloInteger rhs) & {
        x = norm(x + rhs.x);
        return *this;
    }
    ModuloInteger &operator-=(ModuloInteger rhs) & {
        x = norm(x - rhs.x);
        return *this;
    }
    ModuloInteger &operator/=(ModuloInteger rhs) & {
        return *this *= rhs.inv();
    }
    friend  ModuloInteger operator+(ModuloInteger lhs, ModuloInteger rhs) {
        return lhs += rhs;
    }
    friend  ModuloInteger operator-(ModuloInteger lhs, ModuloInteger rhs) {
        return lhs -= rhs;
    }
    friend  ModuloInteger operator*(ModuloInteger lhs, ModuloInteger rhs) {
        return lhs *= rhs;
    }
    friend  ModuloInteger operator/(ModuloInteger lhs, ModuloInteger rhs) {
        return lhs /= rhs;
    }
```

```
    constexpr friend bool operator==(ModuloInteger lsh, ModuloInteger rsh) {
        return lsh.val() == rsh.val();
    }
    constexpr friend bool operator!=(ModuloInteger lsh, ModuloInteger rsh) {
        return lsh.val() != rsh.val();
    }
    constexpr friend std::strong_ordering operator<=>(ModuloInteger lsh,
ModuloInteger rsh) {
        return lsh.val() <=> rsh.val();
    }
    friend  std::istream &operator>>(std::istream &is, ModuloInteger &a) {
        i64 v;
        is >> v;
        a = ModuloInteger(v);
        return is;
    }
    friend  std::ostream &operator<<(std::ostream &os, const ModuloInteger &a)
{
        return os << a.val();
    }
};
template <>
int ModuloInteger<int, 0>::Mod = 998244353;
template <>
long long ModuloInteger<long long, 0>::Mod = 4179340454199820289;
constexpr i64 P = 4179340454199820289;
using Z = ModuloInteger<i64, P>;

template <class T>
struct Polynomial : public std::vector<T> {
    static std::vector<T> w;
    static constexpr auto P = T::getMod();
    static void initW(int r) {
        if (w.size() >= r) {
            return ;
        }
        w.assign(r, 0);
        w[r >> 1] = 1;
        //primitiveroot of Mod
        T s = ::power(T(3), (P - 1) / r);
        for (int i = r / 2 + 1; i < r; i += 1) {
            w[i] = w[i - 1] * s;
        }
        for (int i = r / 2 - 1; i > 0; i -= 1) {
            w[i] = w[i * 2];
        }
    }
    constexpr friend void dft(Polynomial& a) {
        const int n = a.size();
        assert((n & (n - 1)) == 0);
        initW(n);
        for (int k = (n >> 1); k; (k >>= 1)) {
            for (int i = 0; i < n; i += (k << 1)) {
                for (int j = 0; j < k; j += 1) {
                    auto v = a[i + j + k];
                    a[i + j + k] = (a[i + j] - v) * w[j + k];
                    a[i + j] = a[i + j] + v;
                }
            }
        }
```

```
    }
    constexpr friend void idft(Polynomial& a) {
        const int n = a.size();
        assert((n & (n - 1)) == 0);
        initW(n);
        for (int k = 1; k < n; k <<= 1) {
            for (int i = 0; i < n; i += (k << 1)) {
                for (int j = 0; j < k; j += 1) {
                    auto x = a[i + j], y = a[i + j + k] * w[j + k];
                    a[i + j] = x + y;
                    a[i + j + k] = x - y;
                }
            }
        }
        a *= P - (P - 1) / n;
        std::reverse(a.begin() + 1, a.end());
    }
public:
    using std::vector<T>::vector;
    constexpr Polynomial truncate(int k) const {
        auto p = *this;
        p.resize(k);
        return p;
    }
    constexpr friend Polynomial operator+(const Polynomial& a, const
Polynomial& b) {
        Polynomial p(std::max(a.size(), b.size()));
        for (int i = 0; i < p.size(); i += 1) {
            if (i < a.size()) {
                p[i] = p[i] + a[i];
            }
            if (i < b.size()) {
                p[i] = p[i] + b[i];
            }
        }
        return p;
    }
    constexpr friend Polynomial operator-(const Polynomial& a, const
Polynomial& b) {
        Polynomial p(std::max(a.size(), b.size()));
        for (int i = 0; i < p.size(); i += 1) {
            if (i < a.size()) {
                p[i] = p[i] + a[i];
            }
            if (i < b.size()) {
                p[i] = p[i] - b[i];
            }
        }
        return p;
    }
    constexpr friend Polynomial operator-(const Polynomial& a) {
        int n = a.size();
        Polynomial p(n);
        for (int i = 0; i < n; i += 1) {
            p[i] = -a[i];
        }
        return p;
    }
    constexpr friend Polynomial operator*(T a, Polynomial b) {
        for (int i = 0; i < (int)b.size(); i += 1) {
```

```
            b[i] = b[i] * a;
        }
        return b;
    }
    constexpr friend Polynomial operator*(Polynomial a, T b) {
        for (int i = 0; i < int(a.size()); i += 1) {
            a[i] = a[i] * b;
        }
        return a;
    }
    constexpr friend Polynomial operator/(Polynomial a, T b) {
        b = b.inv();
        for (int i = 0; i < int(a.size()); i += 1) {
            a[i] = a[i] * b;
        }
        return b;
    }
    constexpr Polynomial mulxk(int k) const {
        auto p = *this;
        p.insert(p.begin(), k, 0);
        return p;
    }
    constexpr Polynomial modxk(int k) const {
        return Polynomial(this->begin(), this->begin() + k);
    }
    constexpr Polynomial divxk(int k) const {
        if (this->size() <= k) {
            return Polynomial{};
        }
        return Polynomial(this->begin() + k, this->end());
    }
    constexpr T whenXis(T x) const {
        T res = T{};
        for (int i = int(this->size()) - 1; i >= 0; i -= 1) {
            res = res * x + this->at(i);
        }
        return res;
    }
    constexpr Polynomial &operator+=(Polynomial b) {
        return (*this) = (*this) + b;
    }
    constexpr Polynomial &operator-=(Polynomial b) {
        return (*this) = (*this) - b;
    }
    constexpr Polynomial &operator*=(Polynomial b) {
        return (*this) = (*this) * b;
    }
    constexpr Polynomial &operator*=(T b) {
        return (*this) = (*this) * b;
    }
    constexpr Polynomial &operator/=(T b) {
        return (*this) = (*this) / b;
    }
    constexpr friend Polynomial operator*(const Polynomial& a, const
Polynomial& b) {
        if (a.size() == 0 || b.size() == 0) {
            return Polynomial{};
        }
        int n = a.size() + b.size() - 1;
        int s = 1 << std::__lg(2 * n - 1);
```

```
        if ((((P - 1) & (s - 1)) != 0) || std::min(a.size(), b.size()) < 128) {
            Polynomial p(n);
            for (int i = 0; i < a.size(); i += 1) {
                for (int j = 0; j < b.size(); j += 1) {
                    p[i + j] = p[i + j] + a[i] * b[j];
                }
            }
            return p;
        }
        auto f = a.truncate(s), g = b.truncate(s);
        dft(f), dft(g);
        for (int i = 0; i < s; i += 1) {
            f[i] = f[i] * g[i];
        }
        idft(f);
        return f.truncate(n);
    }
    constexpr Polynomial deriv() const {
        int n = this->size();
        if (n <= 1) {
            return Polynomial{};
        }
        Polynomial p(n - 1);
        for (int i = 1; i < n; i += 1) {
            p[i - 1] = i * this->at(i);
        }
        return p;
    }
    constexpr Polynomial integr() const {
        int n = this->size();
        Polynomial p(n + 1);
        std::vector<T> _inv(n + 1);
        _inv[1] = 1;
        for (int i = 2; i <= n; i += 1) {
            _inv[i] = _inv[P % i] * (P - P / i);
        }

        for (int i = 0; i < n; i += 1) {
            p[i + 1] = this->at(i) * _inv[i + 1];
        }
        return p;
    }
};
template <class T>
std::vector<T> Polynomial<T>::w;
using Poly = Polynomial<Z>;

// FFT
template <class T>
struct Polynomial : std::vector<T> {
    using F = std::complex<T>;
    static std::vector<int>r;
    static std::vector<F>w[2];
    static void initR(int log) {
        if (r.size() == (1 << log)) {
            return ;
        }
        int n = 1 << log;
        r.assign(n, 0);
        for (int i = 1; i < n; i += 1) {
```

```
            r[i] = (r[i >> 1] >> 1) | ((i & 1) << (log - 1));
        }
        w[0].assign(n, F());
        w[1].assign(n, F());
        const T pi = std::numbers::pi_v<T>;
        for (int i = 0; i < n; i += 1) {
            auto th = pi * i / n;
            auto cth = std::cos(th);
            auto sth = std::sin(th);
            w[0][i] = F(cth, sth);
            w[1][i] = F(cth, -sth);
        }
    }

    static void fft(std::vector<F>& a, bool invert) {
        int n = a.size();
        initR(std::__lg(n));
        for (int i = 0; i < n; i += 1) {
            if (i < r[i]) {
                std::swap(a[i], a[r[i]]);
            }
        }
        for (int m = 1; m < n; m <<= 1) {
            const int d = n / m;
            for (int R = m << 1, j = 0; j < n; j += R) {
                for (int k = 0; k < m; k += 1) {
                    auto x = a[j + k];
                    auto y = w[invert][d * k] * a[j + m + k];
                    a[j + k] = x + y;
                    a[j + m + k] = x - y;
                }
            }
        }
    }
public:
    using std::vector<T>::vector;
    constexpr friend Polynomial operator*(const Polynomial& a, const
Polynomial& b) {
        if (a.size() == 0 || b.size() == 0) {
            return Polynomial{};
        }
        int n = a.size() + b.size() - 1;
        int l = std::__lg(2 * n - 1);
        int s = 1 << l;
        if (std::min(a.size(), b.size()) < 128) {
            Polynomial p(n);
            for (int i = 0; i < a.size(); i += 1) {
                for (int j = 0; j < b.size(); j += 1) {
                    p[i + j] += a[i] * b[j];
                }
            }
            return p;
        }
        std::vector<F>p(s), q(s);
        for (int i = 0; i < a.size(); i += 1) {
            p[i] = F(a[i], 0);
        }
        for (int i = 0; i < b.size(); i += 1) {
            q[i] = F(b[i], 0);
        }
```

```
        fft(p, false), fft(q, false);
        for (int i = 0; i < s; i += 1) {
            p[i] *= q[i];
        }
        fft(p, true);
        Polynomial h(n);
        for (int i = 0; i < n; i += 1) {
            h[i] = p[i].real() / s;
        }
        return h;
    }
};
template <class T>
std::vector<std::complex<T>> Polynomial<T>::w[2]{};
template <class T>
std::vector<int> Polynomial<T>::r;

using Poly = Polynomial<f64>;
```

分治NTT

把一个 n 个点的树染 n 种颜色，使得任意节点的颜色不等于父节点的颜色减一，且每个点的颜色互不相同，求方案数取模 998244353 。

考虑容斥：考虑$n-1$条约束条件，$i-th$的约束为$C_i \neq C_{p_i}-1$，假设违反其中 k 个约束（并选择了要违反的 k 个约束），那么符合约束的染色数为 $(n-k)!$，则答案为 $\sum_{k=0}^{n-1}(-1)^k f(k)(n-k)!$，其中 $f(k)$ 表示选择要违反 k 个约束的方案数。

由于选择的条件不独立，$f(k)$ 不简单地等于 $\binom{n-1}{k}$，观察到染色完之后每个点的颜色互不相同，违反意味着 $C_i = C_{p_i} - 1$，因此被钦定违反的点一定从上至下形成若干条链，因此每个点的生成函数为 $1 + cntson_u x$，1 表示不选这个点，siz_u 表示选择这个点的方案数。

因此 $f(k) = [x^k]\prod_{i=1}^{n}(1+cntson_u x)$。

```
auto dfs = [&](this auto &&dfs, int l, int r) -> Poly {
    if (l == r) {
        return Poly{1, int(adj[l].size()) - (l != 1)};
    }
    int mid = (l + r) >> 1;
    return dfs(l, mid) * dfs(mid + 1, r);
};
auto p = dfs(1, n);

Z res = 0;
for (int k = 0; k <= n - 1; k += 1) {
    res += Z(k % 2 == 0 ? 1 : -1) * fac[n - k] * p[k];
}
```

生成函数

序列 a 的普通生成函数定义为形式幂级数： $F(x) = \sum_n a_n x^n$ (a_n 为序列 a 的通项公式)

封闭形式：在运用生成函数的过程中形式幂级数的形式可以和封闭形式相互转换。

例如： $\langle 1,1,1,\cdots\rangle$的普通生成函数为 $F(x) = \sum_{n\geq 0} x^n$，根据 $F(x)x+1 = F(x)$，解这个方程得到 $F(x) = \frac{1}{1-x}$，这个就是 $\sum_{n\geq 0} x^n$ 的封闭形式。

$\langle 1, p, p^2, p^3, \cdots\rangle$ 的生成函数为 $\sum_{n\geq 0} p^n x^n$，根据 $F(x)px+1 = F(x)$，可得对应封闭形式 $F(x) = \frac{1}{1-px}$。

二项式定理：$(a+b)^n = \sum_{k=0}^{n}\binom{n}{k}a^k b^{n-k}$

$1+\binom{k}{1}x+\binom{k}{2}x^2+\cdots+\binom{k}{k-1}x^{k-1}=(1+x)^k-x^k$ 。

Matrix

首先就是如何推导一个矩阵乘法的形式，以下根据斐波那契数列给出一个相对好理解的例子

$$F_1=F_2=1,F_i=F_{i-1}+F_{i-2}(i\geq 3)\iff\begin{cases}F_i=F_{i-1}\times 1+F_{i-2}\times 1\\F_{i-1}=F_{i-1}\times 1+F_{i-2}\times 0\end{cases}$$

$$[F_{n-1}\ F_{n-2}]\begin{bmatrix}1&1\\1&0\end{bmatrix}=[F_n\ F_{n-1}]\iff F_n=[1\ 1]\begin{bmatrix}1&1\\1&0\end{bmatrix}^{n-2}$$

$$k=k+d\iff[k\ t\ 1]\begin{bmatrix}1&0&0\\0&1&0\\d&0&1\end{bmatrix}=[k+d\ t\ 1]$$

$$t=t+d\times k\iff[k\ t\ 1]\begin{bmatrix}1&d&0\\0&1&0\\0&0&1\end{bmatrix}=[k\ t+d\times k\ 1]$$

给出以下设计的转移方程，可以得出对应的矩阵

$$f[0]=f'[0]$$

$$f[1]=\max(f'[0]+a[i]+b[i],f'[1]+a[i])$$

$$f[2]=\max(f'[0]+a[i]+2b[i],f'[1]+a[i]+b[i],f'[2])$$

$$f[3]=\max(f'[2]+a[i]+b[i],f'[3]+a[i])$$

$$f[4]=\max(f'[2]+a[i]+2b[i],f'[3]+a[i]+b[i],f'[4])$$

$$\begin{bmatrix}0&a_i+b_i&a_i+2b_i&-\infty&-\infty\\-\infty&a_i&a_i+b_i&-\infty&-\infty\\-\infty&-\infty&0&a_i+b_i&a_i+2b_i\\-\infty&-\infty&-\infty&a_i&a_i+b_i\\-\infty&-\infty&-\infty&-\infty&0\end{bmatrix}$$

矩阵乘法的转移: $C_{i,j}=\sum_{k=0}^{M-1}A_{i,k}B_{k,j}$ 手挽手玩手腕

```
constexpr int K = 5;
constexpr i64 inf = 1E18;
using Matrix = std::array<std::array<i64, K>, K>;
struct Info {
    Matrix m {};
    Info () {
        for (auto & c : m) {
            c.fill(-inf);
        }
        //要注意边界条件
        m[0][0] = m[2][2] = m[4][4] = 0;
    }
    Info (i64 a, i64 b) {
        for (auto & c : m) {
            c.fill(-inf);
        }
        m[0][0] = m[2][2] = m[4][4] = 0;
        m[1][1] = m[3][3] = a;
        m[0][2] = m[2][4] = a + 2 * b;
        m[0][1] = m[2][3] = m[1][2] = m[3][4] = a + b;
    }
    friend Info operator+(const Info& a, const Info& b) {
        Info c{};
```

```
        for (int k = 0; k < K; k += 1) {
            for (int i = 0; i < K; i += 1) {
                for (int j = 0; j < K; j += 1) {
                    chmax(c.m[i][j], a.m[i][k] + b.m[k][j]);
                }
            }
        }
        return c;
    }
};
```

MillerRabin and Rho1

```
std::mt19937 rng(std::chrono::steady_clock::now().time_since_epoch().count());
int64_t rand(int64_t l, int64_t r) {
    return std::uniform_int_distribution<int64_t>(l, r)(rng);
}
int64_t mul(int64_t x, int64_t y, int64_t mod) {
    return static_cast<__int128_t>(x) * y % mod;
}
int64_t power(int64_t a, int64_t n, int64_t mod) {
    int64_t res = 1;
    for (; n != 0; n >>= 1, a = mul(a, a, mod)) {
        if (n % 2 == 1) {
            res = mul(res, a, mod);
        }
    }
    return res;
}
bool isPrime(int64_t n) {
    if (n == 2 or n == 3) {
        return true;
    }
    if (n <= 1 or n % 2 == 0) {
        return false;
    }
    static constexpr int64_t A[] = {2, 325, 9375, 28178, 450775, 9780504,
1795265022};
    int64_t u = n - 1, k = 0;
    while (u % 2 == 0) {
        u >>= 1;
        k += 1;
    }
    for (const auto & x : A) {
        if (x % n == 0) {
            continue;
        }
        int64_t v = power(x, u, n);
        if (v == 1 or v == n - 1) {
            continue;
        }
        for (int j = 1; j <= k; j += 1) {
            int64_t last = v;
            v = mul(v, v, n);
            if (v == 1) {
                if (last != n - 1) {
                    return false;
                }
```

```
                break;
            }
        }
        if (v != 1) {
            return false;
        }
    }
    return true;
}
int64_t Pollardsrho(i64 n) {
    int64_t c = rand(1, n - 1);
    int64_t x = 0, y = 0, s = 1;
    for (int k = 1;; k <<= 1, y = x, s = 1) {
        for (int i = 1; i <= k; i += 1) {
            x = (static_cast<__int128_t>(x) * x + c) % n;
            s = mul(s, std::abs(x - y), n);
            if (i % 127 == 0) {
                int64_t d = std::gcd(s, n);
                if (d > 1) {
                    return d;
                }
            }
        }
        int64_t d = std::gcd(s, n);
        if (d > 1) {
            return d;
        }
    }
    return n;
}
std::vector<int64_t> factorize(int64_t n) {
    std::vector<int64_t>res;
    auto f = [&](auto && f, int64_t n) ->void {
        if (n <= 1) {
            return ;
        }
        if (isPrime(n)) {
            res.push_back(n);
            return ;
        }
        int64_t x = n;
        while (x == n) {
            x = Pollardsrho(x);
        }
        assert(x != 0);
        f(f, x);
        f(f, n / x);
    };
    f(f, n);
    std::sort(res.begin(), res.end());
    return res;
}
std::vector<int64_t> divisors(int64_t n) {
    const auto facp = factorize(n);
    std::vector<std::pair<int64_t, int>>pf;
    for (const auto & c : facp) {
        if (pf.empty() or c != pf.back().first) {
            pf.push_back({c, 1});
        } else {
            pf.back().second += 1;
```

```
        }
    }
    std::vector<int64_t> d;
    auto f = [&](auto && f, int u, int64_t pw) ->void {
        if (u >= std::ssize(pf)) {
            d.push_back(pw);
            return ;
        }
        for (int i = 0; i <= pf[u].second; i += 1) {
            f(f, u + 1, pw);
            pw *= pf[u].first;
        }
    };
    f(f, 0, 1);
    return d;
}
```

Exgcd

线性同余方程 可以改写为如下线性不定方程：

$ax + nk = b$

其中 x 和 k 是未知数。这两个方程是等价的，有整数解的充要条件为 $\gcd(a, n) \mid b$。

应用扩展欧几里德算法可以求解该线性不定方程。根据定理 1，对于线性不定方程 $ax + nk = b$，可以先用扩展欧几里得算法求出一组 x_0, k_0，也就是 $ax_0 + nk_0 = \gcd(a, n)$，然后两边同时除以 $\gcd(a, n)$，再乘 b。就得到了方程 $a\frac{b}{\gcd(a, n)}x_0 + n\frac{b}{\gcd(a, n)}k_0 = b$

于是找到方程的一个解。

```
template<typename T>
T exgcd(T a, T b, T &x, T &y) {
    if (b == T(0)) {
        x = 1, y = 0;
        return a;
    }
    T g = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return g;
}
```

求解 $xK + yN = N - S$ 的最小正整数解 x。(调整原理：最小步长为N)

```
auto solve = [&]() {
    int N, S, K;
    std::cin >> N >> S >> K;
    int g = std::gcd(N, K);
    if ((N - S) % g == 0) {
        int x = 0, y = 0;
        S = N - S;
        S /= g;
        N /= g;
        K /= g;
        auto v = exgcd(K, N, x, y);
        std::cout << i64(x + N) % N * S % N << '\n';
    } else {
        std::cout << -1 << '\n';
    }
};
```

CRT(Chinese Remainder Theorem)

中国剩余定理 (Chinese Remainder Theorem, CRT) 可求解如下形式的一元线性同余方程组（其中 $n_1, n_2, \cdots, n_k$ 两两互质）：

$$\begin{cases} x & \equiv a_1 \pmod{n_1} \\ x & \equiv a_2 \pmod{n_2} \\ & \vdots \\ x & \equiv a_k \pmod{n_k} \end{cases}$$

```
i64 crt(int k, std::vector<i64>& a, std::vector<i64>& n) {
    i64 p = 1, h = 0;
    for (int i = 1; i <= k; i += 1) {
        p = p * n[i];
    }
    for (int i = 1; i <= k; i += 1) {
        i64 m = p / n[i], x = 0, y = 0;
        exgcd(n[i], m, x, y);
        h = (h + i128(y) * m % p * a[i] % p) % p;
    }
    return (h % p + p) % p;
}
```

Block

下取整的数论分块：

对于常数n来说，使得式子$\lfloor \frac{n}{i} \rfloor = \lfloor \frac{n}{j} \rfloor$成立且满足$i \le j \le n$的$j$的最大值为$\lfloor \frac{n}{\lfloor \frac{n}{i} \rfloor} \rfloor$

即$\lfloor \frac{n}{i} \rfloor$所在块的右端点为$\lfloor \frac{n}{\lfloor \frac{n}{i} \rfloor} \rfloor$

上取整的数论分块：

对于常数n来说，使得式子$\lceil \frac{n}{i} \rceil = \lceil \frac{n}{j} \rceil$成立且满足$i \le j \le n$的$j$的最大值为$\lfloor \frac{n-1}{\lfloor \frac{n-1}{i} \rfloor} \rfloor$

即$\lceil \frac{n}{i} \rceil$所在块的右端点为$\lfloor \frac{n-1}{\lfloor \frac{n-1}{i} \rfloor} \rfloor$此时要注意分母可能为$0$

```
for (int l = 1, r = 1; l <= n; l = r + 1) {
    r = n / (n / l);
}
for (int l = 1, r = 1; l <= n; l = r + 1) {
    if (l == n) {
        r = n;
    } else {
        r = (n - 1) / ((n - 1) / l)
    }
}
```

类欧几里得

$$f(a,b,c,n)=\sum_{i=0}^{n}\lfloor\frac{a\times i+b}{c}\rfloor$$

$$\text{if c}\le\text{a or c}\le\text{b :}$$

$$=\sum_{i=0}^{n}\lfloor\frac{(\lfloor\frac{a}{c}\rfloor\times c+a\mod c)\times i+(\lfloor\frac{b}{c}\rfloor\times c+b\mod c)}{c}\rfloor$$

$$=n\times\frac{n+1}{2}\times\lfloor\frac{a}{c}\rfloor+(n+1)\times\lfloor\frac{b}{c}\rfloor+f(a\mod c,b\mod c,c,n)$$

$$\text{else}:\sum_{i=0}^{n}\lfloor\frac{a\times i+b}{c}\rfloor=\sum_{i=0}^{n}\sum_{j=0}^{\lfloor\frac{a\times i+b}{c}\rfloor-1}1$$

$$=\sum_{j=0}^{\lfloor\frac{a\times n+b}{c}\rfloor-1}\sum_{i=0}^{n}[j<\lfloor\frac{a\times i+b}{c}\rfloor]$$

$$j<\lfloor\frac{a\times i+b}{c}\rfloor\iff j+1\le\lfloor\frac{a\times i+b}{c}\rfloor\iff j+1\le\frac{a\times i+b}{c}\iff jc+c\le ai+b$$

$$jc+c-b\le ai\iff jc+c-b-1<ai\iff\lfloor\frac{jc+c-b-1}{a}\rfloor<i$$

$$m:=\lfloor\frac{a\times n+b}{c}\rfloor$$

$$f(a,b,c,n)=\sum_{j=0}^{m-1}\sum_{i=0}^{n}[i>\lfloor\frac{j\times c+c-b-1}{a}\rfloor]$$

$$=\sum_{j=0}^{m-1}(n-\lfloor\frac{jc+c-b-1}{a}\rfloor)$$

$$=n*m-f(c,c-b-1,a,m-1)$$

此时发现这个式子规约下来a和c互换了位置，然后递归，这是一个辗转相除的过程，即复杂度为$O(\log n)$.

$$\sum_{i=1}^{n}\text{popcount}(i)[i\equiv r\pmod m]$$

$$\sum_{i=0}^{\lfloor\frac{n-r}{m}\rfloor}\text{popcount}(i\times m+r)$$

$$\sum_{i=0}^{\lfloor\frac{n-r}{m}\rfloor}\sum_{j=0}^{\lfloor\log_2 n\rfloor}[\frac{i\times m+r}{2^j}\equiv 1\pmod 2]$$

$$\sum_{i=0}^{\lfloor\frac{n-r}{m}\rfloor}\sum_{j=0}^{\lfloor\log_2 n\rfloor}(\lfloor\frac{i\times m+r+2^j}{2^{j+1}}\rfloor-\lfloor\frac{i\times m+r}{2^{j+1}}\rfloor)$$

```
auto f = [&](auto && f, i64 a, i64 b, i64 c, i64 n) {
    if (n < 0) {
        return 0;
    }
    if (a == 0 || n == 0) {
        return b / c * (n + 1);
    }
    if (a >= c || b >= c) {
        return n * (n + 1) / 2 * (a / c) + (n + 1) * (b / c) + f(f, a % c, b %
c, c, n);
    }
    i64 m = (a * n + b) / c;
    return n * m - f(f, c, c - b - 1, a, m - 1);
};
```

Sieve

理论上欧拉筛复杂度优于埃筛，但是 bitset 优化实现的埃筛实测要优于欧拉筛。

```
constexpr int Kn = 1E7;
int minp[Kn + 1];
std::vector<int>primes;
void sieve(int n) {
    for (int x = 2; x <= n; x += 1) {
        if (minp[x] == 0) {
            minp[x] = x;
            primes.push_back(x);
        }
        for (const auto & p : primes) {
            if (p * x > n) {
                break;
            }
            minp[p * x] = x;
            if (x % p == 0) {
                break;
            }
        }
    }
}

p.set();
for (int x = 2; x <= Kn; x = p._Find_next(x)) {
    if (!p.test(x)) {
        continue;
    }
    primes.push_back(x);
    for (i64 y = 1LL * x * x; y <= Kn; y += x) {
        p.reset(y);
    }
}

int l, r;
std::cin >> l >> r;
sieve(int(sqrtl(r)));
std::vector<int>comp;
for (const auto & p : primes) {
    for (i64 x = std::max(2, l / p); x * p <= r; x += 1) {
        if (x * p >= l && x * p <= r) {
            comp.push_back(x * p);
        }
    }
}
std::sort(comp.begin(), comp.end());
comp.erase(std::unique(comp.begin(), comp.end()), comp.end());
std::cout << (r - l + 1 - comp.size() - (l == 1)) << '\n';
```

Comb

$$\text{二项式定理：}\sum_{k=0}^{n}\binom{n}{k}=2^n$$

$$\text{根据多项式卷积计算：}(a+b)^n=\sum_{k=0}^{n}\binom{n}{k}a^kb^{n-k}\iff(1+1)^n=\sum_{k=0}^{n}\binom{n}{k}$$

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

范德蒙卷积：$\sum_{i=0}^{k}\binom{n}{i}\binom{m}{k-i}=\binom{n+m}{k}$

```
struct Comb {
    std::vector<Z>fac, ifac;
    Comb() {}
    Comb(int n) {
        fac.resize(n + 1), ifac.resize(n + 1);
        fac[0] = ifac[0] = 1ULL;
        for (int i = 1; i <= n; i += 1) {
            fac[i] = fac[i - 1] * i;
        }
        ifac[n] = fac[n].inv();
        for (int i = n; i >= 1; i -= 1) {
            ifac[i - 1] = ifac[i] * i;
        }
    }

    Z C(int n, int m) {
        if (m > n or n < 0 or m < 0) {
            return Z(0);
        }
        return fac[n] * ifac[m] * ifac[n - m];
    }

    Z Lucas(i64 n, i64 m) {
        return (m == 0 ? Z(1) : C(n % P, m % P) * Lucas(n / P, m / P));
    }
};
```

Euler

计算小于 n 的正整数中与 n 互质的数的数量

欧拉反演：$\sum_{d|n}\phi(d)=n$

```
for (int x = 1; x <= Kn; x += 1) {
    phi[x] += x;
    for (int y = 2 * x; y <= Kn; y += x) {
        phi[y] -= phi[x];
    }
}
constexpr int N = 1E7;
constexpr int P = 1000003;
bool isprime[N + 1];
int phi[N + 1];
std::vector<int> primes;
std::fill(isprime + 2, isprime + N + 1, true);
phi[1] = 1;
for (int i = 2; i <= N; i++) {
    if (isprime[i]) {
```

```
            primes.push_back(i);
            phi[i] = i - 1;
        }
        for (auto p : primes) {
            if (i * p > N) {
                break;
            }
            isprime[i * p] = false;
            if (i % p == 0) {
                phi[i * p] = phi[i] * p;
                break;
            }
            phi[i * p] = phi[i] * (p - 1);
        }
    }

    int phi(int n) {
        int res = n;
        //这一步可以使用破腊肉优化
        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;
    }
```

Dirichlet

给定一个长度为n的数组a，求一个长度为n的数组b：

满足$b_k = \sum_{i|k} a_i$

```
for (const auto & p : primes) {
    for (int i = 1; i * p <= n; i += 1) {
        a[p * i] += a[i];
    }
}
```

满足$b_k = \sum_{k|i} a_i$

```
for (const auto & p : primes) {
    for (int i = n / p; i >= 1; i -= 1) {
        a[i] += a[p * i];
    }
}
```

Mobius

$$\text{莫比乌斯函数定义如下：}\mu(n)=\begin{cases}1 & \text{如果 } n \text{ 是没有重复质因数的正整数,}\\ -1 & \text{如果 } n \text{ 是质因数个数为奇数的正整数,}\\ 0 & \text{如果 } n \text{ 含有重复的质因数.}\end{cases}$$

$$-\sum_{d|n}\mu(d)=[n==1]$$

$$-[\gcd(x,y)==1]=\sum_{d|\gcd(x,y)}\mu(d)$$

$$-\text{设}f(n),g(n)\text{为两个数论函数。}$$

$$\text{形式一：如果有}f(n)=\sum_{d|n}g(d),\text{那么有}g(n)=\sum_{d|n}\mu(d)f(\frac{n}{d})。$$

$$\text{形式二：如果有}f(n)=\sum_{n|d}g(d),\text{那么有}g(n)=\sum_{n|d}\mu(\frac{d}{n})f(d)。$$

可以使用杜教筛求解莫比乌斯函数的前缀和 $O(n^{\frac{2}{3}})$。

$\sum_{i<j<k}\gcd(\min(a_i,a_j,a_k),\max(a_i,a_j,a_k))=\sum_{i<j}[\gcd(a_i,a_j)==1](j-i-1)$

$\sum_{i<j}(j-i-1)\sum_{d|\gcd(a_i,a_j)}\mu(d)=\sum_d\mu(d)\sum_i[a_i|d]\sum_j[a_j|d](j-i-1)$

复杂度 $O(W\log W)$。

```
for (int d = 1; d <= a.back(); d += 1) {
    std::vector<int>t;
    for (int y = d; y <= a.back(); y += d) {
        t.insert(t.end(), adj[y].begin(), adj[y].end());
    }
    i64 sum = 0;
    for (int i = 0; i < t.size(); i += 1) {
        cnt += mu[d] * (1LL * i * t[i] - sum - i);
        sum += t[i];
    }
}
```

```
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];
```

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

BRAIN

ExchangeArgument

一般使用在图或者树上，类似于微调法，按某种特定方式将图或者树合并的同时要求代价最大或者最小。

从叶子向根贪心：类拓扑排序。

```
#include<bits/stdc++.h>
struct Node {
    int p = 0;
    int x = 0;
    int y = 0;
    constexpr friend bool operator<(const Node& a, const Node& b) {
        return i64(a.x) * b.y < i64(a.y) * b.x;
    }
};
auto main() ->int {
    int n;
    std::cin >> n;
    std::vector<int>par(n + 1);
    for (int u = 2; u <= n; u += 1) {
        std::cin >> par[u];
    }
    std::vector<int>v(n + 1);
    for (int i = 1; i <= n; i += 1) {
        std::cin >> v[i];
    }
    std::priority_queue<Node>q;
    std::vector<int>x(n + 1), y(n + 1);
    for (int i = 1; i <= n; i += 1) {
        x[i] = v[i] == 0;
        y[i] = v[i] == 1;
        q.push({i, x[i], y[i]});
```

```
    }
    i64 res = 0;
    std::vector<int>f(n + 1);
    std::iota(f.begin() + 1, f.end(), 1);
    auto find = [&](int x) {
        while (x != f[x]) {
            x = f[x] = f[f[x]];
        }
        return x;
    };
    while (!q.empty()) {
        auto u = q.top();
        q.pop();
        if (u.x != x[u.p] || u.y != y[u.p]) {
            continue;
        }
        int h = find(par[u.p]);
        res += 1LL * y[h] * x[u.p];
        f[u.p] = h;
        x[h] += u.x;
        y[h] += u.y;
        if (h != 0) {
            q.push({h, x[h], y[h]});
        }
    }
    std::cout << res << '\n';
    return 0;
}
```

树上拓扑序计数

$\text{cnt} = \frac{n!}{\prod_{u=1}^{n} size_u}$

容斥原理

设(U)中元素有(n)种不同的属性，而第(i)种属性为(P_i)，拥有属性(P_i)的元素构成集(S_i)，那么：

$$|\bigcup_{i=1}^{n} S_i| = \sum_{i} |S_i| - \sum_{i<j} |S_i \cap S_j| + \sum_{i<j<k} |S_i \cap S_j \cap S_k| + \cdots +$$

$$(-1)^m \sum_{a_i<a_i+1} |\bigcap_{i=1}^{m} S_{a_i}| + \cdots + (-1)^{n-1} |S_1 \cap \cdots \cap S_n|$$

即$|\bigcup_{i=1}^{n} S_i| = \sum_{m=1}^{n} (-1)^{m-1} \sum_{a_i<a_{i+1}} |\bigcap_{i=1}^{m} S_{a_i}|$

补集：$|\bigcap_{i=1}^{n} S_i| = |U| - |\bigcup_{i=1}^{n} \overline{S_i}|$

异或哈希

通过异或的性质可以解决一些区间问题：

如：给定一个长度为 n 的序列 a，求有多少连续子数组满足子数组中所有数的出现次数都为 k。

显然，当 $k = 1$时，简单的双指针可以解决。

当 $k \geq 2$，考虑给每个数维护一个长度为 k 的随机数列，$[x_1, x_2, \cdots, x_k]$。

当然，为了防止空间问题，需要按照$size$动态维护，直到$size$ 为 k。

注意，每个不同的数要维护不同的随机数列，否则正确性无保证。

然后将每个数重新赋值为 $x_{cnt} \oplus x_{(cnt+1) \mod k}$，通过这样的方式，不难发现，当一个数的出现次数为

k 或者 k 的倍数是，异或起来是 0。

当然，因为这里要求出现次数为 k。因此可以双指针维护一下，然后统计合法区间内$s_r \oplus s_{l-1} = 0$ 的方案数。

```
for (int i = 1; i <= n; i += 1) {
    auto& v = h[a[i]];
    if (v.empty()) {
        v.push_back(rng());
    }
    auto x = v.back();
    if (v.size() < k) {
        v.push_back(rng());
        x ^= v.back();
    } else {
        x = v[p[a[i]] % k] ^ v[(p[a[i]] + 1) % k];
    }
    p[a[i]] += 1;
    s[i] = s[i - 1] ^ x;
}
```

```
std::map<u64, std::vector<int>>rec;
rec[0].push_back(0);
i64 res = 0;
for (int l = 1, r = 1; r <= n; r += 1) {
    cnt[a[r]] += 1;
    while (cnt[a[r]] > k) {
        cnt[a[l]] -= 1;
        l += 1;
    }
    res += rec[s[r]].end() - std::lower_bound(rec[s[r]].begin(),
rec[s[r]].end(), l - 1);
    rec[s[r]].push_back(r);
}
```

随机化

常见的可以使用随机化的场景：

求解区间众数：随机取区间里面的数，判断是不是众数。

求解区间里面的数的出现次数是不是都为 k 的倍数：考虑给相同的数重新随机赋值之后查询是否满足 $k|\sum_{i=l}^{r} a_i$。

每一次判断失误的概率为 $\frac{1}{k}$。

BETA

$$\mod \to [0, \lfloor \frac{x-1}{2} \rfloor]$$

subMask

```
int s = y;
do {
    s = (s - 1) & y;
} while (s != y);
```

- `int __builtin_ffs(int x)`：返回 `x` 的二进制末尾最后一个 1 的位置，位置的编号从 1 开始（最低位编号为 1 ）。当 `x` 为 0 时返回 0
- `int __builtin_clz(unsigned int x)`：返回 `x` 的二进制的前导 0 的个数。当 `x` 为 0 时，结果未定义。
- `int __builtin_ctz(unsigned int x)`：返回 `x` 的二进制末尾连续 0 的个数。当 `x` 为 0 时，结果未定义。
- `int __builtin_clrsb(int x)`：当 `x` 的符号位为 0 时返回 `x` 的二进制的前导 0 的个数减一，否则返回 `x` 的二进制的前导 1 的个数减1。
- `int __builtin_popcount(unsigned int x)`：返回 `x` 的二进制中 1 的个数。
- `int __builtin_parity(unsigned int x)`：判断 `x` 的二进制中 1 的个数的奇偶性。

这些函数都可以在函数名末尾添加 `l` 或 `ll`（如 `__builtin_popcountll`）来使参数类型变为 `(unsigned) long` 或 `(unsigned) long long`（返回值仍然是 `int` 类型）。例如，我们有时候希望求出一个数以二为底的对数，如果不考虑 0 的特殊情况，就相当于这个数二进制的位数 -1 ，而一个数 `n` 的二进制表示的位数可以使用 `32 - __builtin_clz(n)` 表示，因此 `31 - __builtin_clz(n)` 就可以求出 `n` 以二为底的对数。

```
int c = 1;
while (true) {
    system("gen.exe > f.in");
    system("std.exe < f.in > std.out");
    system("sol.exe < f.in > sol.out");
    if (system("fc std.out sol.out > nul") != 0) {
        break;
    } else {
        std::cout << std::format("test : {} \nAC\n", c++) << std::endl;
    }
}
std::cout << std::format("test : {} \nWA\n", c) << std::endl;
```

```
std::istream &operator>>(std::istream &is, i128 &n) {
    std::string s;
    is >> s;
    n = 0;
    for (int i = (s[0] == '-'); i < s.size(); i += 1) {
        n = n * 10 + s[i] - '0';
    }
    if (s[0] == '-') {
        n *= -1;
    }
    return is;
}
std::ostream &operator<<(std::ostream &os, i128 n) {
    std::string s;
    if (n == 0) {
        return os << 0;
    }
    bool sign = false;
    if (n < 0) {
        sign = true;
        n = -n;
    }
    while (n > 0) {
        s += '0' + n % 10;
        n /= 10;
    }
    if (sign) {
```

```
        s += '-';
    }
    std::reverse(s.begin(), s.end());
    return os << s;
}
```

```
int eval(const std::string & s) {
    int n = s.size();
    std::vector<int>stk;
    std::vector<char>sig;

    auto f = [&](const char & c) {
        assert(not stk.empty());
        int r = stk.back();
        stk.pop_back();
        assert(not stk.empty());
        int l = stk.back();
        stk.pop_back();
        if (c == '*') {
            stk.emplace_back(l * r);
        } else if (c == '/') {
            stk.emplace_back(l / r);
        } else if (c == '+') {
            stk.emplace_back(l + r);
        } else if (c == '-') {
            stk.emplace_back(l - r);
        } else assert(false);
    };
    auto r = [&](const char & c) {
        if (c == '-' or c == '+') {
            return 1;
        } else if (c == '*' or c == '/') {
            return 2;
        }
        return -1;
    };
    for (int i = 0; i < n; i += 1) {
        char c = s[i];
        if (c == ' ') {
            continue;
        }
        if (c == '(') {
            sig.emplace_back(c);
        } else if (c == ')') {
            while (not sig.empty() and sig.back() != '(') {
                f(sig.back());
                sig.pop_back();
            }
            sig.pop_back();
        } else if (c == '+' or c == '-' or c == '*' or c == '/') {
            while (not sig.empty() and r(sig.back()) >= r(c)) {
                f(sig.back());
                sig.pop_back();
            }
            sig.push_back(c);
        } else {
            int x = 0;
            while (i < n and std::isdigit(s[i])) {
                x = x * 10 + s[i] - '0';
                i += 1;
```

```
            }
            i -= 1;
            stk.emplace_back(x);
        }
    }
    while (not sig.empty()) {
        f(sig.back());
        sig.pop_back();
    }
    return stk.back();
}
```

```
// 基姆拉尔森公式
const int d[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
bool isLeap(int y) {
    return y % 400 == 0 || (y % 4 == 0 && y % 100 != 0);
}
int daysInMonth(int y, int m) {
    return d[m - 1] + (isLeap(y) && m == 2);
}
int getDay(int y, int m, int d) {
    int ans = 0;
    for (int i = 1970; i < y; i++) {
        ans += 365 + isLeap(i);
    }
    for (int i = 1; i < m; i++) {
        ans += daysInMonth(y, i);
    }
    ans += d;
    return (ans + 2) % 7 + 1;
}
```

```
struct customHash {
    static unsigned long long salt(unsigned long long x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(unsigned long long x) const {
        static const unsigned long long r =
std::chrono::steady_clock::now().time_since_epoch().count();
        return salt(x + r);
    }
};
```

```
// #include<bits/extc++.h>
#include<ext/pb_ds/assoc_container.hpp>
// #include<ext/pb_ds/tree_policy.hpp>
// #include<ext/pb_ds/hash_policy.hpp>
// #include<ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;
template<class KT, class VT = null_type>
using RBT = tree<KT, VT, std::less<KT>, rb_tree_tag,
tree_order_statistics_node_update>;

#pragma GCC optimize("O2")
#pragma GCC optimize("O3")
#pragma GCC optimize("Ofast")
#pragma GCC optimize("unroll-loops")
```

```
#pragma GCC target("avx,avx2,fma")
#pragma GCC target("sse4,popcnt,abm,mmx")
```

```
// 符合条件的数的和
auto dfs = [&](this auto && dfs, int u, int s, int limit, int zero) ->Node {
    if (u == n) {
        return (std::popcount(u32(s)) <= k ? Node(0, 1) : Node(0, 0));
    }
    if (not limit and zero and dp[u][s] != Node(-1, -1)) {
        return dp[u][s];
    }
    Node res = Node();
    if (not zero) {
        res += dfs(u + 1, s, 0, 0);
    }
    int r = limit ? str[u] - '0' : 9;
    for (int d = 1 - zero; d <= r; d += 1) {
        auto v = dfs(u + 1, s | 1 << (d), limit and d == r, 1);
        res.cnt = add(res.cnt, v.cnt);
        res.sum = add(res.sum, v.sum);
        res.sum = add(res.sum, mul(mul(d, v.cnt), pw[n - u - 1]));
    }
    if (not limit and zero) {
        dp[u][s] = res;
    }
    return res;
};
// 上下界数位DP
std::string a = std::to_string(l), b = std::to_string(r);
int n = b.size();
a = std::string(n - a.size(), '0') + a;
auto dfs = [&](auto && dfs, int p, int s, int dw, int up) {
    if (p == n) {
        return s;
    }
    if (not dw and not up and dp[p][s] != -1) {
        return dp[p][s];
    }
    int res = 0;
    int d = dw ? a[p] - '0' : 0, u = up ? b[p] - '0' : 9;
    for (int j = d; j <= u; j += 1) {
        chmax(res, dfs(dfs, p + 1, s + j, dw and j == d, up and j == u));
    }
    if (not dw and not up) {
        dp[p][s] = res;
    }
    return res;
};
return dfs(dfs, 0, 0, 1, 1);
```

```
// SOS
for (int i = 1; i <= n; i += 1)for (int j = 1; j <= n; j += 1)for (int k = 1; k
<= n; k += 1)s[i][j][k] += s[i - 1][j][k];
for (int i = 1; i <= n; i += 1)for (int j = 1; j <= n; j += 1)for (int k = 1; k
<= n; k += 1)s[i][j][k] += s[i][j - 1][k];
for (int i = 1; i <= n; i += 1)for (int j = 1; j <= n; j += 1)for (int k = 1; k
<= n; k += 1)s[i][j][k] += s[i][j][k - 1];
// s[x2, y2, z2] ~ s[x1, y1, z1]
auto query = [&](int x1, int y1, int z1, int x2, int y2, int z2) {
    int res = s[x1][y1][z1];
```

```
    res += s[x2 - 1][y1][z1];
    res += s[x1][y2 - 1][z1];
    res += s[x1][y1][z2 - 1];
    res -= s[x2 - 1][y2 - 1][z1];
    res -= s[x1][y2 - 1][z2 - 1];
    res -= s[x2 - 1][y1][z2 - 1];
    res += s[x2 - 1][y2 - 1][z2 - 1];
    return res;
};
```

```
// 快速维护有序序列
std::merge(a.val.begin(), a.val.end(), b.val.begin(), b.val.end(), t.begin(),
std::greater());
std::vector<int>t(n + 1);
auto merge = [&](int l, int r) {
    int mid = (l + r) >> 1;
    int i = l, j = mid + 1, c = l;
    while (i <= mid and j <= r) {
        if (a[i] <= a[j]) {
            t[c++] = a[i++];
        } else {
            t[c++] = a[j++];
        }
    }
    while (i <= mid) {
        t[c++] = a[i++];
    }
    while (j <= r) {
        t[c++] = a[j++];
    }
    std::copy(t.begin() + l, t.begin() + r + 1, a.begin() + l);
};
auto dfs = [&](this auto && dfs, int l, int r) {
    if (l == r) {
        return ;
    }
    int mid = (l + r) >> 1;
    dfs(l, mid);
    dfs(mid + 1, r);
    merge(l, r);
};
```