Data Structure

SegmentTree

```
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info>info;
    SegmentTree() = default;
    SegmentTree(int n): n{n}, info(4 << std::__lg(n)) {}
    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;
        }
```

```cpp
        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;
}
```

```cpp
// 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() = default;
    LazySegmentTree(int n): n{n}, info(4 << std::__lg(n), Info()), tag(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);
```

```cpp
            }
            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;
    }
```

```cpp
    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;
    }
};
```

如何维护区间只有一个数在某个 bit 上为 0

info:(s = inf ,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

LCM = $\prod p^{\max_l^r cnt}$

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 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。

```cpp
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);
    }
};

i64 query(int p, int l, int r, int k) {
    if (k == 0 || !p) {
        return 0;
    }
    if (t[p].cnt == k) {
        return t[p].sum;
    }
    if (l == r) {
        return 1LL * l * k;
    }
    int mid = (l + r) >> 1;
    if (t[t[p].ch[1]].cnt >= k) {
        return query(t[p].ch[1], mid + 1, r, k);
    }
    return query(t[p].ch[0], l, mid, k - t[t[p].ch[1]].cnt) +
t[t[p].ch[1]].sum;
}

// pst on tree && kth min
auto query = [&](int u, int v, int k) {
    int j = hld.lca(u, v);
```

```cpp
    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);
}
```

```cpp
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);
```

```cpp
        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);
```

```cpp
    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;
}
```

PresidentTrie

```cpp
constexpr int Kw = 30;
constexpr int Kn = 300000;
struct Node {
    int ch[2] {0, 0};
    int cnt = 0;
};
int tot = 0;
Node t[Kn * Kw];
void add(int& cur, int pre, int x, int k) {
    t[cur = ++tot] = t[pre];
    t[cur].cnt += 1;
    if (k < 0) {
        return ;
    }
    add(t[cur].ch[(x >> k & 1) ? 1 : 0], t[pre].ch[(x >> k & 1) ? 1 : 0], x, k
- 1);
}
int query(int l, int r, int x) {
    int ans = 0;
    for (int k = Kw; k >= 0; k -= 1) {
```

```
        int c = (x >> k & 1);
        if (t[t[r].ch[c ^ 1]].cnt - t[t[l].ch[c ^ 1]].cnt >= 1) {
            ans |= 1 << k;
            r = t[r].ch[c ^ 1];
            l = t[l].ch[c ^ 1];
        } else {
            r = t[r].ch[c];
            l = t[l].ch[c];
        }
    }
    return ans;
}
std::vector<int>t(n + 1);
for (int i = 1; i <= n; i += 1) {
    add(t[i], t[i - 1], a[i], Kw);
}
```

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;
    }
}
```

```cpp
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

```cpp
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() = default;
    Fenwick(int n): n{n}, a(n + 1, T{}) {}
    void add(int p, const T& x) {
        if (p >= 1 && p <= n) {
            for (int i = p; i <= n; i += i & -i) {
                a[i] += x;
            }
        }
    }

    T sum(int p) {
        T ans = T();
```

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

    T rangeQuery(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

扩展域并查集可以实现在维护二分图过程中计算偶环个数

```
if (dsu.same(u, v)) {
    // 奇环
    continue;
}
if (dsu.merge(u + n, v) && dsu.merge(u, v + n)) {
    // 维护二分图
} else {
    // 偶环
    cnt ^= 1;
}
```

```
struct DSU {
    std::vector<int>par, siz;
    DSU() = default;
    DSU(int n): par(n + 1), siz(n + 1, 1) {
        std::iota(par.begin(), par.end(), 0);
    }
    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) {
```

```cpp
            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

```cpp
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) {
```

```cpp
            int fx = 1, 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

```
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";
}
```

树剖结合线段树也可以维护一些矩阵类信息，如最大子段和之类的。

但是因为矩阵不满足交换律，记录答案时候的顺序会直接影响求解的答案。

```
// Info.f 和 Info.g 分别表示按顺序和按逆序维护矩阵
auto work = [&](int u, int v) {
    Info fl = Info(), fr = Info();
    while (hld.top[u] != hld.top[v]) {
        if (hld.dep[hld.top[u]] > hld.dep[hld.top[v]]) {
            auto x = seg.rangeQuery(hld.dfn[hld.top[u]], hld.dfn[u]);
            fl = fl + x.g;
            u = hld.par[hld.top[u]];
        } else {
            auto x = seg.rangeQuery(hld.dfn[hld.top[v]], hld.dfn[v]);
            fr = x.f + fr;
            v = hld.par[hld.top[v]];
        }
    }
    if (hld.dep[u] > hld.dep[v]) {
        auto x = seg.rangeQuery(hld.dfn[v], hld.dfn[u]);
        fl = fl + x.g;
    } else {
        auto x = seg.rangeQuery(hld.dfn[u], hld.dfn[v]);
        fr = x.f + fr;
    }
    return fl + fr;
};
```

```
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) {
```

```cpp
        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;
    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);
    }
```

```cpp
}

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

```cpp
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;
    }
};
constexpr int inf = 1E9;
template<class T, int M>
struct Linearbasis {
    bool has_zero = false;
    int dimension{0};
    std::array<T, M>basis {};
    std::array<int, M>time {};
    Linearbasis() {
        basis.fill(0);
        time.fill(-1);
    }
    bool insert(T val, int t = inf) noexcept {
        while (val) {
            int log = std::__lg(val);
            if (!basis[log]) {
                basis[log] = val;
                time[log] = t;
                ++dimension;
                return true;
            }
            if (time[log] < t) {
                std::swap(time[log], t);
                std::swap(basis[log], val);
            }
            val ^= basis[log];
        }
        has_zero = true;
        return false;
    }
    T max(T x = {0}, int t = 0) const noexcept {
```

```
        for (int k = M - 1; k >= 0; k -= 1) {
            if (basis[k] && time[k] >= t) {
                x = std::max(x, x ^ basis[k]);
            }
        }
        return x;
    }
    T min(T y = std::numeric_limits<T>::max(), int t = 0) const noexcept{
        for (int k = M - 1; k >= 0; k -= 1) {
            if (basis[k] && time[k] >= t) {
                y = std::min(y, y ^ basis[k]);
            }
        }
        return y;
    }
    bool contains(T val, int t = 0) const noexcept {
        for (int k = M - 1; k >= 0; k -= 1) {
            if ((val >> k & 1) && time[k] >= t) {
                val ^= basis[k];
            }
        }
        return val == 0;
    }
    T select(T k, int t = 0) const noexcept {
        u64 total = (1ULL << dimension) + (has_zero ? 1 : 0);
        if (k >= total) {
            return T{-1};
        }
        if (has_zero) {
            if (k == 0) {
                return T{};
            }
            k -= 1;
        }
        std::vector<T>r;
        for (int k = 0; k < M; k += 1) {
            if (!basis[k]) {
                continue;
            }
            T v = basis[k];
            for (int i = 0; i < k; i += 1) {
                if ((v >> i & 1)) {
                    v ^= basis[i];
                }
            }
            r.push_back(v);
        }
        T res = 0;
        for (int i = 0; i < int(r.size()); i += 1) {
            if (k >> i & 1) {
                res ^= r[i];
            }
        }
        return res;
    }
    void merge(const Linearbasis<T, M>& other, int t = 0) {
        for (int k = M - 1; k >= 0; k -= 1) {
            if (other.time[k] >= t) {
                insert(other.basis[k]);
            }
```

```
        }
    }
};
using basis = Linearbasis<int, 30>;
```

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;
```

```cpp
        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) {
```

```cpp
        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;
```

```cpp
    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';
```

Bitset

std::bitset 维护 endpos

```cpp
for (int i = 1; i <= s.length(); i += 1) {
    b[s[i - 1] - 'a'].set(i);
}
h.set();
std::string t;
for (int i = 1; i <= t.length(); i += 1) {
    h &= (b[t[i - 1] - 'a'] << (t.length() - i));
}
// h 即为 t 的 {S} endpos
```

Extend : segplusHash

```cpp
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;
}
```

warning

```
// 这样才能正确统计子树大小
for(int p : end) {
    f[p] += 1;
}
```

AhoCorasick

fail 指向当前节点的最长真后缀。

查询一个串的所有子串的出现次数只需要枚举这个串的前缀，然后查询到根的和即可，可以简单维护。

文本串在节尾上打标记，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();
        }
    }
};
```

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

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

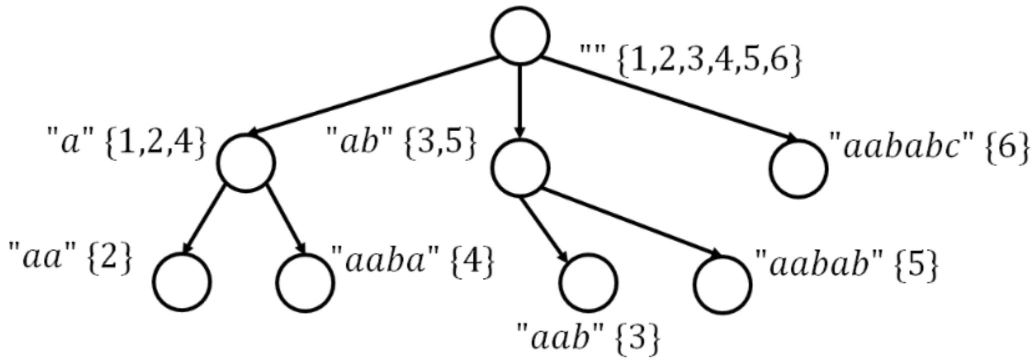
在此基础上可以给每个节点赋予一个最长字符串，是其终点集合中 **任意** 一个终点开始 **往前** 取 `len` 个字符得到的字符串。每个这样的字符串都一样，且 `len` 恰好是满足这个条件的最大值。

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，$s[1...i]$ 和 $s[1...j]$ 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 $\text{len}(i) - \text{len}(link(i))$（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。

子树大小必须预处理，不能一边处理一边用

```
i64 ans = 0;
auto dfs = [&](auto && self, int u) ->void {
    for (int v : adj[u]) {
        self(self, v);
        f[u] += f[v];
    }
};
dfs(dfs, 1);
for (int i = 1; i < sam.size(); i += 1) {
    for (int x = 0; x <= 8; x += 1) {
        int p = sam.next(i, x), q = sam.next(i, x + 1);
        while (p && q) {
            ans += f[p] * f[q] * (sam.len(i) - sam.len(sam.link(i)));
            p = sam.next(p, 9);
            q = sam.next(q, 0);
        }
    }
}
```

```
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;
            }
```

```cpp
            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;
```

```cpp
        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 $S[l...r]$ 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。

```cpp
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 $s[l...r]$ 在 $s[pl...pr]$ 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';
    }
}
```

Given two strings S_1 and S_2 of equal length (indexed from 1).

Now you need to answer q queries, with each query consists of a string T. The query asks how many triplets of integers (i, j, k) $(1 \leq i \leq j < k \leq |S_1|)$ satisfy the condition $S_1[i, j] + S_2[j + 1, k] = T$.

由于 $T = s_1[i, j] + s_2[j + 1, k]$,那枚举 T 的这个分界点 p。提前对 s_1 的正串和 s_2 的反串各自建两个 SAM，那么把 $T[1, p]$ 放到 s_1 的 SAM 上去跑，跑到节点 u，那么合法的 j 一定在 $\text{endpos}(u)$ 中；同理，把 $T[p + 1, \text{len}]$ 放到 s_2 的反串 SAM 上去跑，也能得到相应的节点与 endpos 集合。

问题就转化为：在两个 SAM 的 link 树中，每次询问一对 (u, v)，求两个 SAM 中的子树"颜色集合"（即 endpos 集合）的交集大小。由于每个位置只会出现一次颜色，所以可以看每种颜色对这个询问的贡献。设颜色 i 在两个 SAM 上节点分别是 x_i, y_i，那么对于一个询问 (u, v)，若 x_i 在 u 子树中且 y_i 在 v 子树中就有一个贡献。转化为 *parent tree* 树上的 DFS 序，就有

```
dfn_{x_i} ∈ [dfn_u, dfn_u + siz_u - 1]
∧
dfn_{y_i} ∈ [dfn_v, dfn_v + siz_v - 1].
```

离线二维数点即可。

```
SAM(const std::string& s) {
    int p = 1;
    end = {};
    for (char c : s) {
        p = extend(p, c - 'a');
        end.push_back(p);
    }
    adj.assign(size(), {});
    for (int i = 2; i < size(); i += 1) {
        adj[link(i)].push_back(i);
    }
    dfn = 0;
    auto dfs = [&](auto && self, int u) ->void {
        lo[u] = ++dfn;
        for (int v : adj[u]) {
            self(self, v);
        }
        hi[u] = dfn;
    };
    dfs(dfs, 1);
}
auto work(const std::string& t) {
    int p = 1;
    std::vector<int>r(t.size(), -1);
    for (int i = 0; i < t.size(); i += 1) {
        p = next(p, t[i] - 'a');
        if (!p) {
            break;
        }
        r[i] = p;
    }
    return r;
```

```cpp
}
auto main() ->int {
    std::string s1, s2;
    std::cin >> s1 >> s2;
    auto sam1 = SAM(s1);
    std::reverse(s2.begin(), s2.end());
    auto sam2 = SAM(s2);

    int n = s1.size();
    std::vector<std::vector<int>>e(sam1.size());
    for (int i = 0; i + 1 < n; i += 1) {
        int u1 = sam1.end[i];
        int u2 = sam2.end[n - i - 2];
        // 合法的子树交
        e[sam1.lo[u1]].push_back(sam2.lo[u2]);
    }

    int q;
    std::cin >> q;
    std::vector<std::vector<std::array<int, 3>>>g(sam1.size());
    for (int c = 1; c <= q; c += 1) {
        std::string t;
        std::cin >> t;
        auto ends1 = sam1.work(t);
        std::reverse(t.begin(), t.end());
        auto ends2 = sam2.work(t);
        for (int i = 0; i + 1 < t.size(); i += 1) {
            if (ends1[i] == -1 || ends2[t.size() - i - 2] == -1) {
                continue;
            }
            int l1 = sam1.lo[ends1[i]], r1 = sam1.hi[ends1[i]];
            int l2 = sam2.lo[ends2[t.size() - i - 2]], r2 =
sam2.hi[ends2[t.size() - i - 2]];
            // 转为二维前缀和问题
            g[r1].push_back({r2, 1, c});
            g[r1].push_back({l2 - 1, -1, c});
            g[l1 - 1].push_back({l2 - 1, 1, c});
            g[l1 - 1].push_back({r2, -1, c});
        }
    }
    std::vector<i64>res(q + 1);
    Fenwick<i64>fen(sam2.size());
    for (int i = 0; i < sam1.size(); i += 1) {
        for (int y : e[i]) {
            fen.add(y, 1);
        }
        for (auto [v, coef, q] : g[i]) {
            res[q] += fen.sum(v) * coef;
        }
    }

    for (int i = 1; i <= q; i += 1) {
        std::cout << res[i] << '\n';
    }
    return 0;
}
```

PAM

对于回文串 S 来说，显然它的回文前后缀都等价于它的 border.

对于 Palindrome Automaton：

每个节点表示一个回文子串 $len(u) = |s_u|$

$\text{link}(u)$表示 u 真最长回文后缀，即 $s_{link(u)}$是s_u的最大 border。

根据前缀函数的证明，$len(u) - len(link(u))$是 s_u的最短压缩（前提是$len(u)$能被整除，否则s_u 的最短压缩为 $len(u)$）。

维护出现次数：

```
std::vector<int>f(pam.size());
for (int p : end) {
    f[p] += 1;
}
auto dfs = [&](auto && self, int u) ->void {
    for (int v : adj[u]) {
        self(self, v);
        f[u] += f[v];
    }
    ans += f[u] * __builtin_popcount(pam.t[u].msk);
};
```

维护字符集:

```
t[cur].msk = t[par].msk | (1 << (c - 'a'));
```

```
//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;
        }
    }
    // [0, n - 1]
    // 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;
}
```

中心扩展法

适用于预处理一些具有对称性质的问题。

```
for (int i = 1; i <= n; i += 1) {
    for (int l = i, r = i; 1 <= l && r <= n && s[l] == s[r]; l -= 2, r += 2) {
        ok[l][r] = true;
    }
    for (int l = i, r = i + 1; 1 <= l && r <= n && s[l] == s[r]; l -= 2, r +=
2) {
        ok[l][r] = true;
    }
    for (int l = i - 1, r = i + 1; 1 <= l && r <= n && s[l] == s[r]; l -= 2, r
+= 2) {
        ok[l][r] = true;
    }
```

```
    for (int l = i - 1, r = i + 2; 1 <= l && r <= n && s[l] == s[r]; l -= 2, r += 2) {
        ok[l][r] = true;
    }
}
```

Trie

```
std::vector<Node>t;
t.assign(2, {});
auto newNode = [&]() ->int {
    t.emplace_back();
    return t.size() - 1;
};
for (int l = 1; l <= n; l += 1) {
    int p = 1;
    for (int r = l; r <= n; r += 1) {
        int q = t[p].ch[s[r] - 'a'];
        if (q == 0) {
            t[p].ch[s[r] - 'a'] = newNode();
        }
        p = t[p].ch[s[r] - 'a'];
        if (ok[l][r]) {
            t[p].cnt += 1;
        }
    }
}
// Trie的结构是天然排序的，根据这个性质可以在线性复杂度内排序 string 或者 求解第k大的string。
std::vector<int>sum(t.size());
auto dfs = [&](auto && dfs, int p) ->void {
    sum[p] = t[p].cnt;
    if (t[p].ch[0]) {
        dfs(dfs, t[p].ch[0]);
        sum[p] += sum[t[p].ch[0]];
    }
    if (t[p].ch[1]) {
        dfs(dfs, t[p].ch[1]);
        sum[p] += sum[t[p].ch[1]];
    }
};
dfs(dfs, 1);

std::string res = "";
auto work = [&](auto && work, int p, int k) {
    int cur = sum[p] - sum[t[p].ch[0]] - sum[t[p].ch[1]];
    if (k <= cur) {
        return ;
    }
    k -= cur;
    if (k <= sum[t[p].ch[0]]) {
        res += 'a';
        work(work, t[p].ch[0], k);
    } else {
        res += 'b';
        work(work, t[p].ch[1], k - sum[t[p].ch[0]]);
    }
};
work(work, 1, k);
```

KMP

π :前缀的border 用来处理border问题时相当有用。

f_i表示前缀$s_{1...i}$的border。

一个字符串的border表示最大的len满足$S_{1...len} == S_{n-len+1...n}$。

π_{n-1} 是 s 的最长 border， $n - \pi_{n-1}$是 s 的最短周期

```
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;
}
for (int i = 1; i <= n; i += 1) {
    g[i] = g[f[i]] + w[i];
}
```

SEQ-automaton

```
std::vector<std::array<int, 26>>nxt(n + 1);
nxt[n].fill(n);
std::vector<int>f(n + 2, n);
f[n + 1] = 0;
f[n] = 1;
for (int i = n - 1; i >= 0; i -= 1) {
    nxt[i] = nxt[i + 1];
    nxt[i][s[i] - 'a'] = i;
    for (int j = 0; j < k; j += 1) {
        chmin(f[i], f[nxt[i][j] + 1] + 1);
    }
}
int p = 0;
for (char c : t) {
    if (p > n) {
        break;
    }
    p = nxt[p][c - 'a'] + 1;
}
```

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;
```

```cpp
        };
        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

functinoal graph: scc个数 $= n -$ 环上的点 $+$ 环的数量

```
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;
```

```cpp
    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

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

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

树哈希

可以通过哈希值将子树形态特征化

本质上哈希的方式有很多种，应该根据图的形态来选择:

在处理无向图的时候可以使用随机哈希：每个点随机点权，每个联通块的key就是点权的异或和，图的key就是所有联通块的sum。

```
static const u64 r =
std::chrono::steady_clock::now().time_since_epoch().count();
u64 salt(u64 x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
}
u64 f(u64 x) {
    return salt(x + r);
}
std::vector h(n + 1, 0ULL);
auto dfs = [&](auto && dfs, int u, int p) ->void {
    h[u] = 1;
    for (const auto & v : adj[u]) {
        if (v == p) {
            continue;
        }
        dfs(dfs, v, u);
        h[u] += f(h[v]);
    }
};
dfs(dfs, 1, 0);
```

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;
```

```cpp
    }
    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！

```cpp
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;
}
```

partite

二分图即相邻点颜色不同，按照颜色分域。

```
bool isBipartite(int n, vector<pair<int, int>>& edges) {
    for (auto [u, v] : edges) {
        if (dsu.same(u, v)) return false;
        dsu.merge(u, v + n);
        dsu.merge(v, u + n);
    }
    return true;
}
```

KM

```
constexpr int inf = 1E7;
template<class T>
struct MaxAssignment {
    public:
        T solve(int nx, int ny, std::vector<std::vector<T>> a) {
            assert(0 <= nx && nx <= ny);
            assert(int(a.size()) == nx);
            for (int i = 0; i < nx; ++i) {
                assert(int(a[i].size()) == ny);
                for (auto x : a[i])
                    assert(x >= 0);
            }

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

```
        }
    };

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

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

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

            T delta = std::numeric_limits<T>::max();
            for (int y = 0; y < ny; ++y)
                if (!visy[y])
                    delta = std::min(delta, slack[y]);
            for (int x = 0; x < nx; ++x)
                if (visx[x])
                    lx[x] -= delta;
            for (int y = 0; y < ny; ++y) {
                if (visy[y]) {
                    ly[y] += delta;
                } else {
```

```
                        slack[y] -= delta;
                    }
                }
                for (int y = 0; y < ny; ++y) {
                    if (!visy[y] && slack[y] == 0) {
                        if (yx[y] == -1) {
                            ex = slackx[y];
                            ey = y;
                            found = true;
                            break;
                        }
                        que.push(yx[y]);
                        p[yx[y]] = slackx[y];
                        visy[y] = visx[yx[y]] = true;
                        update(yx[y]);
                    }
                }
            }

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

MincostFlow

```
template<class T>
struct MinCostFlow {
    struct edge {
        int to;
        T cap;
        T cost;
        edge(int to, T cap, 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) {
        pre.assign(n, -1);
        dis.assign(n, std::numeric_limits<T>::max());
        std::priority_queue<std::pair<T, int>, std::vector<std::pair<T, int>>,
std::greater<>>q;
        q.push({dis[s] = 0, s});
        while (!q.empty()) {
            auto [d, u] = q.top();
            q.pop();
            if (dis[u] != d) {
                continue;
            }
            for (int i : adj[u]) {
                auto [v, cap, cost] = e[i];
                if (cap > 0 && dis[v] > d + h[u] - h[v] + cost) {
                    pre[v] = i;
                    q.push({dis[v] = d + h[u] - h[v] + cost, v});
                }
            }
        }
        return dis[t] != std::numeric_limits<T>::max();
    }

    MinCostFlow() {}
    MinCostFlow(int n): n{n}, e{}, adj(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;
```

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

```cpp
        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

$\text{MaxFlow} == \text{MinCut}$
网络流的定义:

- 有一个 s 点有一个 t 点，以及若干其它点
- 有若干有向边。其中 s 点没有入边，t 点没有出边
- 对于除了 s 点和 t 点以外的所有点，入流量之和等于出流量之和
- 形象的比喻想象水管，s 点是无限的水源，t 点是无限的出口，边的权值等于管的容量
- 整个网络同时能通过的最大流量，这个流量等于 s 点的所有出流量，等于 t 点的所有入流量

割的定义:

- 设置两个集合 S 集合和 T 集合，且 s 点属于 S 集合，t 点属于 T 集合
- 其它所有点属于 S 集合或 T 集合之一
- 所有来自于 S 集合，指向 T 集合的边，权值之和为割的大小
  定理:
- 最小割 = 最大流
- 直观理解就是 最小割的边满流的情况下，无法再找到增加流量的办法
  能否套用最小割方法的判断:

1. 把最少/最小代价对应到最小割
2. 把二元选择问题 改变为集合选择问题
3. 不合法的方案，定义成流量无穷大的边
   建图过程:
4. 根据可以做二元选择的对象定义所有的点

5. 根据不合法方案连接无穷大的边，明确点的集合归属含义
6. 根据点的集合归属含义，明确所有需要支付的成本
   额外思路：
7. 先假设能获得所有收益，然后把得不到的收益也记作成本，来解决最大化收益问题

棋子问题

有 N*M 的棋盘，上面有 K 个棋子，位于 Ai 行 Bi 列。

一次操作可以消除一行的棋子或者消除一列的棋子。

问最少要多少次操作，才能消灭所有的棋子？

1 ≤ N,M ≤ 100

思考过程：

- 最少次操作 => 最小代价 => 最小割（把边看做代价）
- 每一行，可以选择消除或不消除
- 每一列，可以选择消除或不消除
- 所以网络中的点是 行 与 列
- 不合法的方案，表示有棋子没有被消除。

建模过程：

- 行属于 S 集合表示不消除，属于 T 集合表示消除
- 列属于 T 集合表示不消除，属于 S 集合表示消除
- 不合法方案的边：
  - 对于为 1 的单元格，行不消除且列不消除非法
  - 对应单元格建立行指向列的无穷大流量边
- 支付成本：
  - s 到行，流量为 1
  - 列到 t，流量为 1

设备采购问题

有 N 种设备，采购价格分别是 A1,A1,...AN

和 M 个项目，项目收益分别是 B1,B1,...BM

另外有 K 个约束条件，Ci,Di 表示完成 Di 号项目，必须要拥有 Ci 号设备。

问最高能获得的利润是多少（利润等于收益-成本）

思考过程：

- 先假设能获得所有收益，然后把得不到的收益也记作成本
- 设备：采购与不采购；项目：做与不做
- 不合法方案：项目依赖的设备没有采购，但又要做对应项目

建图过程：

- 设备属于 S 集合表示不采购，属于 T 集合表示采购
- 项目属于 S 集合表示不做，属于 T 集合表示做
- 不合法方案：
  - 项目依赖设备，但设备未采购，不合法

    - 有依赖时，从设备指向项目，流量无穷大
- 支付成本：
    - 设备采购支付成本，s 点指向设备，流量为采购成本
    - 项目不做支付损失，项目指向 t 点，流量为项目收益

取数问题

在一个 M*N 的棋盘中，每个方格有一个正整数。现在要从方格中取若干个数，使任意 2 个数所在方格没有公共边，并使取出的数总和最大。试设计一个满足要求的取数算法。

思考过程：

- 先假设能获得所有数，然后建图使方案合法
- 每个数选择与不选择
- 不合法的方案：相邻的格子不能同时选

建图过程：

- 经过尝试和思考，得出结论，区分坐标奇偶性（横坐标和纵坐标之和的奇偶性）
- 偶数格属于 S 集合表示选择，属于 T 集合表示不选择
- 奇数格属于 T 集合表示选择，属于 S 集合表示不选择
- 不合法方案：
    - 相邻的格子不能同时选
    - 相邻的格子，从偶数格向奇数格建边，流量无穷大
- 支付代价
    - 偶数格不选择，s 向偶数格建边，流量为格子的值
    - 奇数格不选择，奇数格向 t 建边，流量为格子的值

```
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<int>cur, h;

    MaxFlow() {}
    MaxFlow(int n): n{n}, e{}, adj(n), cur(n), h(n) {}

    bool bfs(int s, int t) {
        h.assign(n, -1);
        std::queue<int>q;
        h[s] = 0;
        q.push(s);
        while (!q.empty()) {
            auto u = q.front();
            q.pop();
            for (int i : adj[u]) {
                auto [v, c] = e[i];
```

```
                if (c > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    if (v == t) {
                        return true;
                    }
                    q.push(v);
                }
            }
        }
        return false;
    }

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

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

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

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

    struct Edge {
        int from;
        int to;
```

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

```
// 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;
}
```

给定 m 个约束：$\sum_{i=l}^{r} A_i = s$，求解满足约束的A的最小的 $sum\ of\ A$。

```
auto main() ->int {
    int n, m;
    std::cin >> n >> m;
    std::vector<std::array<int, 3>>edges;
    // s[i] - s[i - 1] >= 1 -> s[i - 1] - s[i] <= -1 -> s[i - 1] <= s[i] - 1;
    for (int i = 1; i <= n; i += 1) {
        edges.push_back({i, i - 1, -1});
    }
    for (int i = 1; i <= m; i += 1) {
        int l, r, x;
        std::cin >> l >> r >> x;
        // s[r] - s[l - 1] >= x && s[r] - s[l - 1] <= x
        // s[r] - s[l - 1] >= x -> s[l - 1] - s[r] <= -x
        edges.push_back({r, l - 1, -x});
        // s[r] - s[l - 1] <= x
        edges.push_back({l - 1, r, x});
    }
    std::vector<i64>f(n + 1), g(n + 1);
    for (int t = 0; t <= n + 3; t += 1) {
        for (auto [u, v, w] : edges) {
            chmin(f[v], g[u] + w);
        }
        if (t == n + 3 && g != f) {
            std::cout << -1 << '\n';
            std::exit(0);
        }
```

```
        g = f;
    }
    std::cout << f[n] - f[0] << '\n';
    return 0;
}
```

```
bool spfa(int n, int s) {
  memset(dis, 0x3f, (n + 1) * sizeof(int));
  dis[s] = 0, vis[s] = 1;
  q.push(s);
  while (!q.empty()) {
    int u = q.front();
    q.pop(), vis[u] = 0;
    for (auto ed : e[u]) {
      int v = ed.v, w = ed.w;
      if (dis[v] > dis[u] + w) {
        dis[v] = dis[u] + w;
        cnt[v] = cnt[u] + 1;  // 记录最短路经过的边数
        if (cnt[v] >= n) return false;
        // 在不经过负环的情况下，最短路至多经过 n - 1 条边
        // 因此如果经过了多于 n 条边，一定说明经过了负环
        if (!vis[v]) q.push(v), vis[v] = 1;
      }
    }
  }
  return true;
}
```

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。（$1 \leq n \leq 10^5$） 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 $0 \leq k \leq n-1$，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。

```
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

平面上一点到圆内随机一个点的欧几里得距离的期望为 $\frac{r^2}{2}+d^2$。

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 $\mathrm{O(logn)}$ 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);
}
```

求两直线的交点：可以尝试找公共边，存在公共边的话，就可以通过面积的比值方便地求解交点问题了。

```cpp
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';
}
```

有 n 种前进方式，第 i 种每秒消耗 A_i 体力，前进 B_i 米。

Note: 每种前进方式每次的执行时间为 0到1 的正实数。

判断是否有可能在总消耗不超过 C_i 的情况下前进 D_i 米，找出可能的最短时间。

构建 (A_i, B_i) 的凸壳，那么显然只有凸壳右下方的有解。

记 B/A 最大的点位最省力的点，即最大的斜率，若该情况都无解，那么答案为 -1。

当前一定有解的话，显然要找时间效率最高的点，即求最小的 t 使得 $(\frac{C}{t}, \frac{D}{t})$ 位于凸壳上。

利用极角序二分找到有交的边，构建共边三角形即可通过相似三角形得出比值(懒得思考也可以直接 lineintrsect)。

```cpp
auto main() ->int {
    p.emplace_back(0, 0);
    std::sort(p.begin(), p.end());
    std::vector<P>stk{};
    for (const auto & v : p) {
        while (stk.size() > 1 && cross(stk.back() - stk.end()[-2], v -
stk.back()) >= 0) {
            stk.pop_back();
        }
        stk.push_back(v);
    }
    stk.erase(stk.begin());
    // 只维护向上的凸壳，这样才能保证最优。
    while (stk.size() > 1 && stk.end()[-2].y >= stk.back().y) {
        stk.pop_back();
    }
    int q;
```

```cpp
    std::cin >> q;
    while (q--) {
        P p;
        std::cin >> p.x >> p.y;
        if (cross(stk[0], p) > 0) {
            std::cout << -1 << '\n';
            continue;
        }
        double ans;
        if (cross(p, stk.back()) >= 0) {
            ans = 1.0 * p.y / stk.back().y;
        } else {
            int lo = 0, hi = stk.size() - 1;
            while (lo <= hi) {
                int mid = (lo + hi) >> 1;
                if (cross(p, stk[mid]) >= 0) {
                    lo = mid + 1;
                } else {
                    hi = mid - 1;
                }
            }
            ans = 1.0 * cross(stk[lo] - stk[lo - 1], p) / cross(stk[lo], stk[lo
- 1]);
        }
        std::cout << ans << '\n';
    }
    return 0;
}
```

```cpp
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);
}

P rotate(const P& p, const double rad) {
    return {p.x * std::cos(rad) - p.y * std::sin(rad), p.x * std::sin(rad) +
p.y * std::cos(rad)};
}

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);
}

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) {
```

```cpp
    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
// 3 : intersect at endpoint
// 不包含端点
auto strictlyintersect = [&](L l1, L l2) {
    if (pointOnLineLeft(l2.a, l1) * pointOnLineLeft(l2.b, l1) >= 0) {
        return false;
    }
    std::swap(l1, l2);
    if (pointOnLineLeft(l2.a, l1) * pointOnLineLeft(l2.b, l1) >= 0) {
        return false;
    }
    return true;
};
```

```cpp
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;
```

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

```cpp
    }
    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();
```

```cpp
    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 T>
std::pair<size_t, size_t>tangent(const std::vector<Point<T>>& p, const Point<T>
a) {
    int n = p.size();
    auto extreme = [&](auto dir) ->size_t {
        const auto check = [&](size_t i) {
            return cross(dir(p[i]), p[(i + 1) % n] - p[i]) >= 0;
        };
        const auto dir0 = dir(p[0]);
        const auto check0 = check(0);

        if (!check(0) && check(n - 1)) {
            return 0;
        }
        const auto cmp = [&](const Point<T>& v) {
            const size_t vi = &v - p.data();
            if (vi == 0) {
                return 1;
            }
            const auto checkv = check(vi);
            const auto t = cross(dir0, v - p[0]);
            if (vi == 1 && checkv == check0 && t == 0) {
                return 1;
            }
            return checkv ^ (checkv == check0 && t <= 0);
        };
        return std::partition_point(p.begin(), p.end(), cmp) - p.begin();
    };
    auto i = extreme([&](const Point<T>& u) { return u - a; });
    auto j = extreme([&](const Point<T>& u) { return a - u; });
    return {i, j};
}

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));
}

// 三点外接圆 点和线 :-> 方向
P m1 = (p[i] + p[j]) / 2;
P m2 = (p[i] + p[k]) / 2;
P v1 = rotate(p[j] - p[i]);
P v2 = rotate(p[k] - p[i]);
P o = lineIntersection(L(m1, m1 + v1), L(m2, m2 + v2));
```

minkowskisum

```
Point sa = a[0], sb = b[0];
for (int i = 0; i < n; i++) {
    if (a[i].y() < sa.y() || (a[i].y() == sa.y() && a[i].x() < sa.x())) {
        sa = a[i];
    }
}
for (int i = 0; i < m; i++) {
    if (b[i].y() < sb.y() || (b[i].y() == sb.y() && b[i].x() < sb.x())) {
        sb = b[i];
    }
}
auto s = sa + sb;
std::vector<Point> d(n + m);
for (int i = 0; i < n; i++) {
    d[i] = a[(i + 1) % n] - a[i];
}
for (int i = 0; i < m; i++) {
    d[n + i] = b[(i + 1) % m] - b[i];
}
std::sort(d.begin(), d.end(), [&](Point a, Point b) {
    if (sgn(a) != sgn(b)) {
        return sgn(a) == 1;
    }
    return cross(a, b) > 0;
});
std::vector<Point> c(n + m);
c[0] = s;
for (int i = 0; i < n + m - 1; i++) {
    c[i + 1] = c[i] + d[i];
}
```

选中一些$P(a-b, c-d)$，使得它们的和的模长最大。

```
auto main() ->int {
    int n;
    std::cin >> n;
    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);
    }
    std::cout << h << '\n';
    return 0;
}
```

给两个凸包，第二个移动(dx, dy)，判断二者有没有交集。

```
auto main() ->int {
    int n, m, q;
    std::cin >> n >> m >> q;
    std::vector<P>a(n), b(m);
    for (auto &e : a) {
        std::cin >> e;
    }
    for (auto &e : b) {
        std::cin >> e;
        e *= -1;
    }
    a = getHull(a), b = getHull(b);
    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';
    }
    return 0;
}
```

两个凸包移动，求碰撞的时刻：

转为相对移动速度，$p1$静止，$p2$以$v = (dx2 - dx1, dy2 - dy1)$移动。

即求$Line(P(0,0), v)$和凸包$r = p1 + (-p2)$的交点

```
bool ok = true;
for (int i = 0; i < r.size(); i += 1) {
    if (cross(r[i], r[(i + 1) % r.size()]) < 0) {
        ok = false;
    }
}
if (ok) {
    std::cout << 0 << '\n';
    return 0;
}
if (v == Point<double>(0, 0)) {
    std::cout << -1 << '\n';
    return 0;
}
double ans = std::numeric_limits<double>::infinity();
for (int i = 0; i < r.size(); i += 1) {
    Point<double> a = r[i], b = r[(i + 1) % r.size()];
    if (cross(b - a, v) == 0) {
        continue;
    }
    auto p = lineIntersection(L(P(0, 0), v), L(a, b));
    if (dot(v, p) >= 0 && dot(p - a, b - a) >= 0 && dot(p - b, a - b) >= 0) {
        chmin(ans, std::sqrt(square(p) / square(v)));
    }
}
if (ans > 1e18) {
    std::cout << -1 << '\n';
} else {
    std::cout << ans << '\n';
}
```

heltion

```
using point_t = i64; // 全局数据类型，可修改为 long long 等

constexpr point_t eps = 1e-8;
constexpr long double PI = 3.1415926535897932384l;

// 点与向量
template <typename T> struct point {
    T x, y;
    bool operator==(const point &a) const {
        return (abs(x - a.x) <= eps && abs(y - a.y) <= eps);
    }
    bool operator<(const point &a) const {
        if (abs(x - a.x) <= eps)
```

```
            return y < a.y - eps;
        return x < a.x - eps;
    }
    bool operator>(const point &a) const { return !(*this < a || *this == a); }
    point operator+(const point &a) const { return {x + a.x, y + a.y}; }
    point operator-(const point &a) const { return {x - a.x, y - a.y}; }
    point operator-() const { return {-x, -y}; }
    point operator*(const T k) const { return {k * x, k * y}; }
    point operator/(const T k) const { return {x / k, y / k}; }
    T operator*(const point &a) const { return x * a.x + y * a.y; } // 点积
    i128 operator^(const point &a) const {
        return (i128)x * a.y - (i128)y * a.x;
    } // 叉积，注意优先级
    int toleft(const point &a) const {
        const auto t = (*this) ^ a;
        return (t > eps) - (t < -eps);
    }                                               // to-left 测试
    T len2() const { return (*this) * (*this); } // 向量长度的平方
    T dis2(const point &a) const {
        return (a - (*this)).len2();
    } // 两点距离的平方

    // 涉及浮点数
    long double len() const { return sqrtl(len2()); } // 向量长度
    long double dis(const point &a) const { return sqrtl(dis2(a)); } // 两点距离
    long double ang(const point &a) const {
        return acosl(max(-1.0l, min(1.0l, ((*this) * a) / (len() * a.len()))));
    } // 向量夹角
    point rot(const long double rad) const {
        return {x * cos(rad) - y * sin(rad), x * sin(rad) + y * cos(rad)};
    } // 逆时针旋转（给定角度）
    point rot(const long double cosr, const long double sinr) const {
        return {x * cosr - y * sinr, x * sinr + y * cosr};
    } // 逆时针旋转（给定角度的正弦与余弦）
};

using Point = point<point_t>;

// 极角排序
struct argcmp {
    bool operator()(const Point &a, const Point &b) const {
        const auto quad = [](const Point &a) {
            if (a.y < -eps)
                return 1;
            if (a.y > eps)
                return 4;
            if (a.x < -eps)
                return 5;
            if (a.x > eps)
                return 3;
            return 2;
        };
        const int qa = quad(a), qb = quad(b);
        if (qa != qb)
            return qa < qb;
        const auto t = a ^ b;
        // if (abs(t)<=eps) return a*a<b*b-eps;  // 不同长度的向量需要分开
        return t > eps;
    }
};
```

```
// 直线
template <typename T> struct line {
    point<T> p, v; // p 为直线上一点，v 为方向向量

    bool operator==(const line &a) const {
        return v.toleft(a.v) == 0 && v.toleft(p - a.p) == 0;
    }
    int toleft(const point<T> &a) const {
        return v.toleft(a - p);
    }                                    // to-left 测试
    bool operator<(const line &a) const // 半平面交算法定义的排序
    {
        if (abs(v ^ a.v) <= eps && v * a.v >= -eps)
            return toleft(a.p) == -1;
        return argcmp()(v, a.v);
    }

    // 涉及浮点数
    point<T> inter(const line &a) const {
        return p + v * ((a.v ^ (p - a.p)) / (v ^ a.v));
    } // 直线交点
    long double dis(const point<T> &a) const {
        return abs(v ^ (a - p)) / v.len();
    } // 点到直线距离
    point<T> proj(const point<T> &a) const {
        return p + v * ((v * (a - p)) / (v * v));
    } // 点在直线上的投影
};

using Line = line<point_t>;

// 线段
template <typename T> struct segment {
    point<T> a, b;

    bool operator<(const segment &s) const {
        return make_pair(a, b) < make_pair(s.a, s.b);
    }

    // 判定性函数建议在整数域使用

    // 判断点是否在线段上
    // -1 点在线段端点 | 0 点不在线段上 | 1 点严格在线段上
    int is_on(const point<T> &p) const {
        if (p == a || p == b)
            return -1;
        return (p - a).toleft(p - b) == 0 && (p - a) * (p - b) < -eps;
    }

    // 判断线段直线是否相交
    // -1 直线经过线段端点 | 0 线段和直线不相交 | 1 线段和直线严格相交
    int is_inter(const line<T> &l) const {
        if (l.toleft(a) == 0 || l.toleft(b) == 0)
            return -1;
        return l.toleft(a) != l.toleft(b);
    }

    // 判断两线段是否相交
    // -1 在某一线段端点处相交 | 0 两线段不相交 | 1 两线段严格相交
```

```cpp
    int is_inter(const segment<T> &s) const {
        if (is_on(s.a) || is_on(s.b) || s.is_on(a) || s.is_on(b))
            return -1;
        const line<T> l{a, b - a}, ls{s.a, s.b - s.a};
        return l.toleft(s.a) * l.toleft(s.b) == -1 &&
               ls.toleft(a) * ls.toleft(b) == -1;
    }

    // 点到线段距离
    long double dis(const point<T> &p) const {
        if ((p - a) * (b - a) < -eps || (p - b) * (a - b) < -eps)
            return min(p.dis(a), p.dis(b));
        const line<T> l{a, b - a};
        return l.dis(p);
    }

    // 两线段间距离
    long double dis(const segment<T> &s) const {
        if (is_inter(s))
            return 0;
        return min({dis(s.a), dis(s.b), s.dis(a), s.dis(b)});
    }
};

using Segment = segment<point_t>;

// 多边形
template <typename T> struct polygon {
    vector<point<T>> p; // 以逆时针顺序存储

    size_t nxt(const size_t i) const { return i == p.size() - 1 ? 0 : i + 1; }
    size_t pre(const size_t i) const { return i == 0 ? p.size() - 1 : i - 1; }

    // 回转数
    // 返回值第一项表示点是否在多边形边上
    // 对于狭义多边形，回转数为 0 表示点在多边形外，否则点在多边形内
    pair<bool, int> winding(const point<T> &a) const {
        int cnt = 0;
        for (size_t i = 0; i < p.size(); i++) {
            const point<T> u = p[i], v = p[nxt(i)];
            if (abs((a - u) ^ (a - v)) <= eps && (a - u) * (a - v) <= eps)
                return {true, 0};
            if (abs(u.y - v.y) <= eps)
                continue;
            const Line uv = {u, v - u};
            if (u.y < v.y - eps && uv.toleft(a) <= 0)
                continue;
            if (u.y > v.y + eps && uv.toleft(a) >= 0)
                continue;
            if (u.y < a.y - eps && v.y >= a.y - eps)
                cnt++;
            if (u.y >= a.y - eps && v.y < a.y - eps)
                cnt--;
        }
        return {false, cnt};
    }

    // 多边形面积的两倍
    // 可用于判断点的存储顺序是顺时针或逆时针
    T area() const {
```

```cpp
        T sum = 0;
        for (size_t i = 0; i < p.size(); i++)
            sum += p[i] ^ p[nxt(i)];
        return sum;
    }

    // 多边形的周长
    long double circ() const {
        long double sum = 0;
        for (size_t i = 0; i < p.size(); i++)
            sum += p[i].dis(p[nxt(i)]);
        return sum;
    }
};

using Polygon = polygon<point_t>;

// 凸多边形
template <typename T> struct convex : polygon<T> {
    // 闵可夫斯基和
    convex operator+(const convex &c) const {
        const auto &p = this->p;
        vector&lt;Segment&gt; e1(p.size()), e2(c.p.size()),
            edge(p.size() + c.p.size());
        vector<point<T>> res;
        res.reserve(p.size() + c.p.size());
        const auto cmp = [](const Segment &u, const Segment &v) {
            return argcmp()(u.b - u.a, v.b - v.a);
        };
        for (size_t i = 0; i < p.size(); i++)
            e1[i] = {p[i], p[this->nxt(i)]};
        for (size_t i = 0; i < c.p.size(); i++)
            e2[i] = {c.p[i], c.p[c.nxt(i)]};
        rotate(e1.begin(), min_element(e1.begin(), e1.end(), cmp), e1.end());
        rotate(e2.begin(), min_element(e2.begin(), e2.end(), cmp), e2.end());
        merge(e1.begin(), e1.end(), e2.begin(), e2.end(), edge.begin(), cmp);
        const auto check = [](const vector<point<T>> &res, const point<T> &u) {
            const auto back1 = res.back(), back2 = *prev(res.end(), 2);
            return (back1 - back2).toleft(u - back1) == 0 &&
                   (back1 - back2) * (u - back1) >= -eps;
        };
        auto u = e1[0].a + e2[0].a;
        for (const auto &v : edge) {
            while (res.size() > 1 && check(res, u))
                res.pop_back();
            res.push_back(u);
            u = u + v.b - v.a;
        }
        if (res.size() > 1 && check(res, res[0]))
            res.pop_back();
        return {res};
    }

    // 旋转卡壳
    // func 为更新答案的函数，可以根据题目调整位置
    template <typename F> void rotcaliper(const F &func) const {
        const auto &p = this->p;
        const auto area = [](const point<T> &u, const point<T> &v,
                             const point<T> &w) { return (w - u) ^ (w - v); };
        for (size_t i = 0, j = 1; i < p.size(); i++) {
```

```
            const auto nxti = this->nxt(i);
            func(p[i], p[nxti], p[j]);
            while (area(p[this->nxt(j)], p[i], p[nxti]) >=
                   area(p[j], p[i], p[nxti])) {
                j = this->nxt(j);
                func(p[i], p[nxti], p[j]);
            }
        }
    }

    // 凸多边形的直径的平方
    T diameter2() const {
        const auto &p = this->p;
        if (p.size() == 1)
            return 0;
        if (p.size() == 2)
            return p[0].dis2(p[1]);
        T ans = 0;
        auto func = [&](const point<T> &u, const point<T> &v,
                        const point<T> &w) {
            ans = max({ans, w.dis2(u), w.dis2(v)});
        };
        rotcaliper(func);
        return ans;
    }

    // 判断点是否在凸多边形内
    // 复杂度 O(logn)
    // -1 点在多边形边上 | 0 点在多边形外 | 1 点在多边形内
    int is_in(const point<T> &a) const {
        const auto &p = this->p;
        if (p.size() == 1)
            return a == p[0] ? -1 : 0;
        if (p.size() == 2)
            return segment<T>{p[0], p[1]}.is_on(a) ? -1 : 0;
        if (a == p[0])
            return -1;
        if ((p[1] - p[0]).toleft(a - p[0]) == -1 ||
            (p.back() - p[0]).toleft(a - p[0]) == 1)
            return 0;
        const auto cmp = [&](const Point &u, const Point &v) {
            return (u - p[0]).toleft(v - p[0]) == 1;
        };
        const size_t i =
            lower_bound(p.begin() + 1, p.end(), a, cmp) - p.begin();
        if (i == 1)
            return segment<T>{p[0], p[i]}.is_on(a) ? -1 : 0;
        if (i == p.size() - 1 && segment<T>{p[0], p[i]}.is_on(a))
            return -1;
        if (segment<T>{p[i - 1], p[i]}.is_on(a))
            return -1;
        return (p[i] - p[i - 1]).toleft(a - p[i - 1]) > 0;
    }

    // 凸多边形关于某一方向的极点
    // 复杂度 O(logn)
    // 参考资料：https://codeforces.com/blog/entry/48868
    template <typename F> size_t extreme(const F &dir) const {
        const auto &p = this->p;
        const auto check = [&](const size_t i) {
```

```cpp
            return dir(p[i]).toleft(p[this->nxt(i)] - p[i]) >= 0;
        };
        const auto dir0 = dir(p[0]);
        const auto check0 = check(0);
        if (!check0 && check(p.size() - 1))
            return 0;
        const auto cmp = [&](const Point &v) {
            const size_t vi = &v - p.data();
            if (vi == 0)
                return 1;
            const auto checkv = check(vi);
            const auto t = dir0.toleft(v - p[0]);
            if (vi == 1 && checkv == check0 && t == 0)
                return 1;
            return checkv ^ (checkv == check0 && t <= 0);
        };
        return partition_point(p.begin(), p.end(), cmp) - p.begin();
    }

    // 过凸多边形外一点求凸多边形的切线，返回切点下标
    // 复杂度 O(logn)
    // 必须保证点在多边形外
    pair<size_t, size_t> tangent(const point<T> &a) const {
        const size_t i = extreme([&](const point<T> &u) { return u - a; });
        const size_t j = extreme([&](const point<T> &u) { return a - u; });
        return {i, j};
    }

    // 求平行于给定直线的凸多边形的切线，返回切点下标
    // 复杂度 O(logn)
    pair<size_t, size_t> tangent(const line<T> &a) const {
        const size_t i = extreme([&](...) { return a.v; });
        const size_t j = extreme([&](...) { return -a.v; });
        return {i, j};
    }
};
```

scanline

```cpp
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

假设有两个非共线的向量 A, B，它们的法向量为 $\mathrm{cross}(A, B)$。

```
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>
```

```cpp
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) {
```

```cpp
        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

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

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

多项式初等函数的推导方法：

多项式$\ln$(根据定义需要满足$[x^0]f(x)=1$)

$\frac{\mathrm{d}\ln f(x)}{\mathrm{d}x}=\frac{f'(x)}{f(x)} \mod x^n$

$\ln f(x)=\int \mathrm{d}\ln f(x)=\int \frac{f'(x)}{f(x)} \mod x^n$

多项式幂函数

$(f(x))^k=e^{k\ln f(x)}$, 需要保证$[x^0]f(x)=1$

设$f(x)$的最低次项为f_ix^i, $f^k(x)=f_i^kx^{ik}exp(k\ln\frac{f(x)}{f_ix^i})$

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

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

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

$\sum_1^{n-k+1}a_i\binom{n-i}{k-1}=\frac{1}{(k-1)!}\sum_1^{n-k+1}a_i\frac{(n-i)!}{(n-i-k+1)!}$

$F(x)=\sum_1^n a_i(n-i)![x^i],\ G(x)=\sum_0^n\frac{1}{i!}[x^i]$

$\mathrm{ans}_k=\frac{1}{(k-1)!}[x^{n-k+1}]F\times G$

求解类似 $\sum_{u=1}^n\binom{s_u}{k}$ 的式子：

s_u 是离散的值，变换到频域之后可以使用卷积求解：$f_x=\sum_u s_u==x$

$\sum_{u=1}^n\binom{s_u}{k}=\sum_{s=1}^n f_s\binom{s}{k}=\sum_{s=1}^n f_s\frac{s!}{k!(s-k)!}$

令 $F(x)=f_xx!, G(x)=\frac{1}{x!}$

令 $T(n-x)=G(x)$

$\sum_{s=1}^n F(s)\times G(s-k)=F(s)\times T(n-s+k)$

$\mathrm{ans}_k=\frac{1}{k!}[x^{n+k}]F(s)\times T(n-s+k)$

```
//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;
            }
        }
```

```cpp
    }
    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) & {
        x = multiply(x, rhs.inv().val(), getMod());
        return *this;
    }

    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>;
constexpr int P = 998244353;
using Z = ModuloInteger<int, 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];
        }
    }
    static 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;
                }
            }
        }
    }
    static 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 a;
    }
    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;
    }
    // assert(this->at(0) != 0)
    constexpr Polynomial inv(int m = -1) const {
        const int n = this->size();
        if (m < 0) {
            m = n;
        }
        Polynomial p = Polynomial{this->at(0).inv()};
        p.reserve(4 * m);
        for (int k = 2; k / 2 < m; k <<= 1) {
            Polynomial q = Polynomial(this->begin(), this->begin() +
std::min(n, k)).truncate(2 * k);
            p.resize(2 * k);
            dft(p), dft(q);
            for (int i = 0; i < 2 * k; i += 1) {
                p[i] = p[i] * (2 - p[i] * q[i]);
            }
            idft(p);
            p.resize(k);
        }
        return p.truncate(m);
    }
```

```cpp
    constexpr Polynomial ln(int m = -1) const {
        const int n = this->size();
        if (m < 0) {
            m = n;
        }
        return (deriv() * inv(m)).integr().truncate(m);
    }
    constexpr Polynomial exp(int m = -1) const {
        const int n = this->size();
        if (m < 0) {
            m = n;
        }

        Polynomial p{1};
        int k = 1;
        while (k < m) {
            k <<= 1;
            p = (p * (Polynomial{1} - p.ln(k) + truncate(k))).truncate(k);
        }
        return p.truncate(m);
    }
    constexpr Polynomial sqrt(int m = -1) const {
        const int n = this->size();
        if (m < 0) {
            m = n;
        }
        Polynomial p{1};
        int k = 1;
        constexpr T inv2 = T(2).inv();
        while (k < m) {
            k <<= 1;
            p = (p + (truncate(k) * p.inv(k)).truncate(k)) * inv2;
        }
        return p.truncate(m);
    }
    constexpr Polynomial power(long long k, int m, long long k2 = -1) const {
        if (0 < k and k <= (1 << 10)) {
            Polynomial p = (*this);
            Polynomial ans{1};
            for (; k; k >>= 1) {
                if (k & 1) {
                    ans *= p;
                    if (static_cast<int>(ans.size()) > m) {
                        ans.truncate(m);
                    }
                }
                p *= p;
                if (static_cast<int>(p.size()) > m) {
                    p.truncate(m);
                }
            }
            return ans.truncate(m);
        }
        k2 = k2 < 0 ? k : k2;
        unsigned int i = 0;
        while (i < this->size() and this->at(i) == T{}) {
            i++;
        }
        if (i == this->size() or k * i >= m) {
            return Polynomial(m, T{});
```

```
        }
        T v = this->at(i);
        Polynomial f = divxk(i) / v;
        return (f.ln(m - i * k) * k).exp(m - i * k).mulxk(i * k) * ::power(v,
k2);
    }
};
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 $f(k) = [x^k]\prod_{i=1}^{n}(1 + cntson_u x)$。

```cpp
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 $F(x)=\sum_{n\ge 0} x^n$，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 $\sum_{n\ge 0} x^n$ 的封闭形式。

$\langle 1,p,p^2,p^3,\cdots\rangle$ 的生成函数为 $\sum_{n\ge 0} p^n x^n$，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：$(a+b)^n=\sum_{k=0}^{n}\binom{n}{k}a^k b^{n-k}$

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

麦克劳林展开式

$e^x=\sum_{n=0}^{\infty}\frac{x^n}{n!}$

$\sin x=\sum_{n=0}^{\infty}(-1)^n\frac{x^{2n+1}}{(2n+1)!}$

$\cos x=\sum_{n=0}^{\infty}(-1)^n\frac{x^{2n}}{(2n)!}$

$\ln(1+x)=\sum_{n=1}^{\infty}(-1)^{n+1}\frac{x^n}{n},x\in(-1,1]$

$\ln(1-x)=-\sum_{n=1}^{\infty}\frac{x^n}{n},|x|<1$

$\frac{1}{1-x}=\sum_{n=0}^{\infty}x^n,|x|<1$

$\arctan x=\sum_{n=0}^{\infty}(-1)^n\frac{x^{2n+1}}{2n+1},x\in[-1,1]$

等比数列求和：$\frac{a_0(1-q^n)}{1-q^n}$

付公主有 n 种商品，她要准备出摊了

每种商品体积为 v_i，都有无限件

给定 m，对于 $s\in[1,m]$，请你回答用这些商品恰好装 s 体积的方案数

对于一个体积为 v 的物品来说，$a_n=\{1，x^v，x^{2v}\cdots\}$

$G(x)*x^v+1=G(x) \iff G(x)=\frac{1}{1-x^v}$

显然有 $ans_s=[x^s]\prod_{i=1}^{n}\frac{1}{1-x^{v_i}}$

考虑 $F=\prod_{i=1}^{n}\frac{1}{1-x^{v_i}}$

$\ln F=\ln\prod_{i=1}^{n}\frac{1}{1-x^{v_i}}=\sum_{i=1}^{n}\ln\frac{1}{1-x^{v_i}}$

考虑麦克劳林展开式，$\ln\frac{1}{1-x^{v_i}}=-\ln(1-x^{v_i})=\sum_{n=1}^{\infty}\frac{x^{nv_i}}{n}$

$\ln F=\sum_{i=1}^{n}\sum_{j=1}^{\infty}\frac{x^{jv_i}}{j}$

由于只考虑 $s\in[1,m]$，因此保留前 m 项即可，显然内层暴力枚举单点加的复杂度是调和级数的，最后 exp 即可得到 F

```
std::vector<int>a(m + 1);
for (int i = 1; i <= n; i += 1) {
    int v;
    std::cin >> v;
    a[v] += 1;
}
Poly p(m + 1);
for (int i = 1; i <= m; i += 1) {
    for (int j = 1; j <= m / i; j += 1) {
        p[i * j] += a[i] * Z(j).inv();
    }
}
auto r = p.exp();
for (int i = 1; i <= m; i += 1) {
    std::cout << r[i] << '\n';
}
```

二项式反演

记 f_n 表示恰好使用 n 个不同元素形成特定结构的方案数，g_n 表示从 n 个不同元素中选出 $i\geq 0$ 个元素形成特定结构的方案数。

$g_n=\sum_{i=0}^{n}\binom{n}{i}f_i$

$f_n=\sum_{i=0}^{n}(-1)^{n-i}\binom{n}{i}g_i$

求长度为 n 的字符串中恰好出现 k 次 “bit” 的字符串数目：

由于 "bit" 不会重叠，考虑整个 “bit” 作为一个特殊字符，则至少出现 k 次的数目为 $F_k=\binom{n-2k}{k}26^{n-3k}$。

根据容斥原理可得 $G_k=\sum_{j=k}^{n}(-1)^{j-k}\binom{j}{k}F_k$。

显然为和差卷积的形式。

n个人以相同概率选择自己之外的一名玩家并射击，击中的概率为 p，被击中者死亡。求恰好有 k 人存活的概率。

$F(k)$ 表示恰好杀死 k 人的概率，$G(k)$表示杀死不超过 k 人的概率。

$F(k)=\binom{n}{k}\sum_{i=0}^{k}(-1)^{k-i}\binom{k}{i}G(i)$

$G(i)=(1-p+p\frac{i}{n-1})^{n-i}(1-p+p\frac{i-1}{n-1})^{i}$

被击中的 i 个人要么没打中，要么打中其他的 $i-1$ 个人。

不被击中的 $n-i$ 个人要么没打中，要么打中 i 个人中的一个。

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);
```

```cpp
    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 $a\dfrac{b}{\gcd(a, n)}x_0 + n\dfrac{b}{\gcd(a, n)}k_0 = b$

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

$$\text{下取整的数论分块：}$$
$$\text{对于常数}n\text{来说，使得式子}\lfloor\frac{n}{i}\rfloor=\lfloor\frac{n}{j}\rfloor\text{成立且满足}i\le j\le n\text{的}j\text{的最大值为}\lfloor\frac{n}{\lfloor\frac{n}{i}\rfloor}\rfloor$$
$$\text{即}\lfloor\frac{n}{i}\rfloor\text{所在块的右端点为}\lfloor\frac{n}{\lfloor\frac{n}{i}\rfloor}\rfloor$$
$$\text{上取整的数论分块：}$$
$$\text{对于常数}n\text{来说，使得式子}\lceil\frac{n}{i}\rceil=\lceil\frac{n}{j}\rceil\text{成立且满足}i\le j\le n\text{的}j\text{的最大值为}\lfloor\frac{n-1}{\lfloor\frac{n-1}{i}\rfloor}\rfloor$$
$$\text{即}\lceil\frac{n}{i}\rceil\text{所在块的右端点为}\lfloor\frac{n-1}{\lfloor\frac{n-1}{i}\rfloor}\rfloor\text{此时要注意分母可能为}0$$

```
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] = p;
            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';
```

线性筛扩展可以求很多积性函数

```
void sieve (int n) {
    std::fill (is_composite, is_composite + n, false);
    func[1] = 1;
    for (int i = 2; i < n; ++i) {
        if (!is_composite[i]) {
            prime.push_back (i);
            func[i] = 1; cnt[i] = 1;
```

```
        }
        for (int j = 0; j < prime.size () && i * prime[j] < n; ++j) {
            is_composite[i * prime[j]] = true;
            if (i % prime[j] == 0) {
                func[i * prime[j]] = func[i] / cnt[i] * (cnt[i] + 1);
                cnt[i * prime[j]] = cnt[i] + 1;
                break;
            } else {
                func[i * prime[j]] = func[i] * func[prime[j]];
                cnt[i * prime[j]] = 1;
            }
        }
    }
}
```

Comb

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

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

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

范德蒙卷积:

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

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

Lucas:

$\binom{n}{m} \mod 2 = \binom{\lfloor\frac{n}{2}\rfloor}{\lfloor\frac{m}{2}\rfloor}\binom{n \mod 2}{m \mod 2}$

考虑到 $\binom{0}{0} = 1, \binom{0}{1} = 0, \binom{1}{0} = 1, \binom{1}{1} = 1$，故 $\binom{n}{m} \equiv 1 \mod 2$ 当且仅当 $n|m == m$。

```
binom[n][m] = fac[n] * ifac[m] * ifac[n - m];
lucas[n][m] = (m == 0 ? 1 : binom[n % P][m % P] * lucas(n / P, m / P))
binom[0][0] = 1;
for (int n = 1; n < P; n += 1) {
    binom[n][0] = 1;
    for (int m = 1; m < P; m += 1) {
        binom[n][m] = (binom[n - 1][m - 1] + binom[n - 1][m]) % P;
    }
}
```

任意模数 Binom

```
std::vector<std::pair<int, int>> factorize(int n) {
    std::vector<std::pair<int, int>> factors;
    for (int i = 2; static_cast<long long>(i) * i <= n; i++) {
        if (n % i == 0) {
            int t = 0;
            for (; n % i == 0; n /= i)
                ++t;
            factors.emplace_back(i, t);
        }
    }
    if (n > 1)
        factors.emplace_back(n, 1);
```

```cpp
    return factors;
}
constexpr int power(int base, i64 exp) {
    int res = 1;
    for (; exp > 0; base *= base, exp /= 2) {
        if (exp % 2 == 1) {
            res *= base;
        }
    }
    return res;
}
constexpr int power(int base, i64 exp, int mod) {
    int res = 1 % mod;
    for (; exp > 0; base = 1LL * base * base % mod, exp /= 2) {
        if (exp % 2 == 1) {
            res = 1LL * res * base % mod;
        }
    }
    return res;
}
int inverse(int a, int m) {
    int g = m, r = a, x = 0, y = 1;
    while (r != 0) {
        int q = g / r;
        g %= r;
        std::swap(g, r);
        x -= q * y;
        std::swap(x, y);
    }
    return x < 0 ? x + m : x;
}
// CRT
int solveModuloEquations(const std::vector<std::pair<int, int>> &e) {
    int m = 1;
    for (std::size_t i = 0; i < e.size(); i++) {
        m *= e[i].first;
    }
    int res = 0;
    for (std::size_t i = 0; i < e.size(); i++) {
        int p = e[i].first;
        res = (res + 1LL * e[i].second * (m / p) * inverse(m / p, p)) % m;
    }
    return res;
}
constexpr int N = 1E5;
class Binomial {
    const int mod;
private:
    const std::vector<std::pair<int, int>> factors;
    std::vector<int> pk;
    std::vector<std::vector<int>> prod;
    static constexpr i64 exponent(i64 n, int p) {
        i64 res = 0;
        for (n /= p; n > 0; n /= p) {
            res += n;
        }
        return res;
    }
    int product(i64 n, std::size_t i) {
        int res = 1;
```

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

Euler

计算小于 n 的正整数中与 n 互质的数的数量

欧拉反演：$\sum_{d|n} \phi(d) = n$

$smalltrick$: 求小于等于 n 的正整数里面有多少是 d 的倍数 $\iff$ $\lfloor \frac{n}{d} \rfloor$。

```
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

$$\text{给定一个长度为}n\text{的数组}a\text{，求一个长度为}n\text{的数组}b\text{:}$$
$$\text{满足}b_k = \sum_{i|k} a_i$$

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

$$\text{满足}b_k = \sum_{k|i} a_i$$

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

Mobius

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

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

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

$$-设 f(n),g(n)为两个数论函数。$$

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

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

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 $O(W\log W)$。

```cpp
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];
    }
}
```

```cpp
std::unordered_map<int, Z> fMu;
constexpr int N = 1E7;
std::vector<int> minp, primes;
std::vector<Z> mu;
void sieve(int n) {
    minp.assign(n + 1, 0);
    mu.resize(n);
    primes.clear();
    mu[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (minp[i] == 0) {
            mu[i] = -1;
            minp[i] = i;
            primes.push_back(i);
        }
        for (auto p : primes) {
            if (i * p > n) {
```

```
                break;
            }
            minp[i * p] = p;
            if (p == minp[i]) {
                break;
            }
            mu[i * p] = -mu[i];
        }
    }
    // 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

括号树

```
int p = 0, q = 0;
std::vector<int>stk{0};
std::vector<std::vector<int>>adj(n);
for (char c : s) {
    if (c == '(') {
        q += 1;
        adj[p].push_back(q);
        stk.push_back(q);
        p = q;
    } else {
        stk.pop_back();
        p = stk.back();
    }
}
```

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>
```

```cpp
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

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

最小二乘法

$\hat{k} = \frac{\sum x_i y_i - n\overline{xy}}{\sum x_i^2 - n\overline{x}^2}$

$\hat{b} = \overline{y} - \hat{k}$

给定 n 个点(i, a_i)，求直线$l: y = kx + b$使得 $\sum(a_i - y_i)^2$ 最小。

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;
}
```

```cpp
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 $\frac{1}{k}$。

决策单调分治

考虑求解 $\max \sum_{i\in S} a_i + \max_{i\in S} b_i$ 或者 $\min \sum_{i\in S} a_i + \max_{i\in S} b_i$ 。

$|S| = k \in [1, n]$。

首先考虑如何计算固定的 k 的答案。将所有盘子按照 b 排序，枚举第 $x(k \le x \le n)$ 个盘子作为选中的 b 最大的盘子，那么剩下的 $k-1$ 个盘子显然是贪心选择前 $x-1$个盘子 中 a 最小的 $k-1$ 个。给定 k 和 x ，可以通过可持久线段树在 $O(\log n)$ 的时间内求出对应方案的值 $w(k,x)$。 令f(k)表示使k取到最优解的x。对于两个不同的决策 $x, y(x<y)$，若 $w(k,x) \ge w(k,y)$ ，那么增大 k 之后由于 y 的可选择范围严格包含了 x 的可选择范围，因此 y 新选的 a 值一定不大于 x 所选的，即 $w(k\prime,x) \ge w(k\prime,y)$对于 $k \le k\prime \le n$ 恒成立。由此可得 $f(1) \le f(2) \le f(3) \le \cdots \le f(n)$，最优决策具有单调性，可以分治求解，共需计算 $O(n \log n)$ 个 $w(k,x)$ 的值。

```cpp
std::sort(p.begin() + 1, p.end(), [&](const auto & x, const auto & y) {
    return x.second < y.second;
});
auto solve = [&](auto&& self, int l, int r, int lo, int hi) {
    if (l > r) {
        return ;
    }
    int mid = (l + r) >> 1;
    int k = std::max(mid, lo);
    i64& x = ans[mid];
    for (int i = std::max(mid, lo); i <= hi; i += 1) {
        if (chmin(x, p[i].first + p[i].second + sum(root[i - 1], 1, inf, mid - 
1))) {
            k = i;
        }
    }
    self(self, l, mid - 1, lo, k);
    self(self, mid + 1, r, k, hi);
};
solve(solve, 1, n, 1, n);
```

SG

```
void solve () {
    constexpr int W = 5;
    int sg[W][W][W];
    auto dfs = [&](auto && dfs, int r, int b, int m, int o) {
        if (sg[r][b][m] != -1) {
            return sg[r][b][m];
        }
        std::set<int>h;
        if (r >= 1) {
            h.insert(dfs(dfs, r - 1, b, m, o ^ 1));
        }
        if (r >= 2) {
            h.insert(dfs(dfs, r - 2, b, m, o ^ 1));
        }
        if (r >= 3) {
            h.insert(dfs(dfs, r - 3, b, m, o ^ 1));
        }
        if (b >= 1) {
            h.insert(dfs(dfs, r + 1, b - 1, m, o ^ 1));
        }
        if (b >= 1) {
            h.insert(dfs(dfs, r, b - 1, m, o ^ 1));
            if (r >= 1) {
                h.insert(dfs(dfs, r - 1, b - 1, m, o ^ 1));
            }
        }
        if (b >= 2) {
            h.insert(dfs(dfs, r + 1, b - 2, m, o ^ 1));
        }
        if (m >= 1) {
            h.insert(dfs(dfs, r + 1, b, m - 1, o ^ 1));
            h.insert(dfs(dfs, r, b + 1, m - 1, o ^ 1));
            h.insert(dfs(dfs, r + 1, b + 1, m - 1, o ^ 1));
        }
        sg[r][b][m] = 0;
        while (h.contains(sg[r][b][m])) {
            sg[r][b][m] += 1;
        }
        return sg[r][b][m];
    };
    memset(sg, -1, sizeof(sg));
    for (int r = 0; r <= 3; r += 1) {
        for (int b = 0; b <= 3; b += 1) {
            for (int m = 0; m <= 3; m += 1) {
                std::cout << std::format("r = {}, b = {}, m = {}, (r + 2b + 4m)
% 4 = {}, sg = {}\n", r, b, m, (r + 2 * b + 4 * m) % 4,dfs(dfs, r, b, m, 0));
            }
        }
    }
}
```

BETA

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

线性逆元

```
inv[1] = 1;
for (int i = 1; i <= n; i += 1) {
    inv[i] = (P - P / i)* inv[P % i] % P;
}
```

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。

```
if [ $# -lt 1 ]; then
    echo "Usage: ./run.sh filename (without .cpp)"
    exit 1
fi
NAME=$1
echo "Compiling..."
g++ "$NAME.cpp" -Wall -Wl,--stack=1073741824 -O2 -std=c++20 -lstdc++exp -D
LOCAL -o "$NAME"
if [ $? -ne 0 ]; then
    echo "Compilation failed."
    exit 1
fi
echo "Running..."
START=$(date +%s%3N)
if [ -f "$NAME.in" ]; then
    ./"$NAME" < "$NAME.in" > "$NAME.out"
else
    ./"$NAME"
fi
RUN_STATUS=$?
END=$(date +%s%3N)
DURATION_MS=$((END - START))
if [ $RUN_STATUS -ne 0 ]; then
    echo "Runtime error (exit code: $RUN_STATUS)"
else
    echo "Execution time: ${DURATION_MS} ms"
fi
```

```
rm "$NAME"
```

```cpp
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;
```

```cpp
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;
}
```

```cpp
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);
```

```cpp
        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);
```

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

```cpp
// 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) {
```

```cpp
        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);
};
```