ICPC NTHU SplayTreap

October 14, 2022

Contents

1 Setup

1.1 Template.h

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
using namespace std;
using namespace __gnu_pbds;
using ll = long long;
using ld = long double;
using pii = pair<int, int>;
using pll = pair<ll, ll>;
using vi = vector<int>;
using vl = vector<ll>;
#define SZ(a) ((int)a.size())
#define ALL(v) (v).begin(), (v).end()
#define RALL(v) (v).rbegin(), (v).rend()
```

```
#define PB push_back
#define PPB pop_back
#define EB emplace_back
#define F first
#define S second
template<class T> inline bool chmin(T& a, const T& b) { if(a > b) { a = b; return true; } return false; }
template<class T> inline bool chmax(T& a, const T& b) { if(a < b) { a = b; return true; } return false; }
const double EPS = 1e-9;
```

1.2 q.sh

source/.custom_bash_commands.sh

2 Data-structure

2.1 HashMap.h

```
struct splitmix64_hash {
    static ull splitmix64(ull x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    ull operator()(ull x) const {
        static const ull FIXED_RANDOM = RAND;
        return splitmix64(x + FIXED_RANDOM);
    }
};
template<class T, class U, class H = splitmix64_hash> using hash_map = gp_hash_table<T, U, H>;
template<class T, class H = splitmix64_hash> using hash_set = hash_map<T, null_type, H>;
```

2.2 OrderStatisticTree.h

```
template<class T, class Comp = less<T>> using ordered_set = tree<T, null_type, Comp, rb_tree_tag, tree_order_statistics_node_update>;
template<class T> using ordered_multiset = ordered_set<T, less_equal<T>>;
// Use `s.erase(s.find_by_order(s.order_of_key(x)))` when using `ordered_multiset`
```

2.3 Segtree.h

```
template<class S, S (*e)(), S (*op)(S, S)>
class segtree {
public:
    segtree() : segtree(0) {}
    segtree(int _n) : segtree(vector<S>(_n, e())) {}
    segtree(const vector<S>& a): n(int(a.size())) {
        log = 31 - __builtin_clz(2 * n - 1);
        size = 1 << log;
        st = vector<S>(size * 2, e());
        for(int i = 0; i < n; ++i) st[size + i] = a[i];
        for(int i = size - 1; i; --i) update(i);
    }
    void set(int p, S val) {
        assert(0 <= p && p < n);
        p += size, st[p] = val;
        for(int i = 1; i <= log; ++i) update(p >> i);
    }
    inline S get(int p) const {
        assert(0 <= p && p < n);
        return st[p + size];
    }
    inline S operator[](int p) const { return get(p); }
    S prod(int l, int r) const {
        assert(0 <= l && l <= r && r <= n);
        S sml = e(), smr = e();
        l += size, r += size;
        while(l < r) {
            if(l & 1) sml = op(sml, st[l++]);
            if(r & 1) smr = op(st[--r], smr);
            l >>= 1, r >>= 1;
        }
        return op(sml, smr);
    }
    inline S all_prod() const { return st[1]; }
    template<bool (*f)(S)> int max_right(int l) const {
        return max_right(l, [](S x) { return f(x); });
    }
    template<class F> int max_right(int l, F f) const {
        assert(0 <= l && l <= n);
        assert(f(e()));
        if(l == n) return n;
        l += size;
        S sm = e();
        do {
            while(!(l & 1)) l >>= 1;
            if(!f(op(sm, st[l]))) {
                while(l < size) {
                    l <<= 1;
                    if(f(op(sm, st[l]))) {
                        sm = op(sm, st[l++]);
                    }
                }
                return l - size;
            }
            sm = op(sm, st[l++]);
        } while((l & -l) != l);
        return n;
    }
    template<bool (*f)(S)> int min_left(int r) const {
        return min_left(r, [](S x) { return f(x); });
    }
```

```
    }
    template<class F> int min_left(int r, F f)
    const {
        assert(0 <= r && r <= n);
        assert(f(e()));
        if(r == 0) return 0;
        r += size;
        S sm = e();
        do {
            r--;
            while(r > 1 && (r & 1)) {
                r >>= 1;
            }
            if(!f(op(st[r], sm))) {
                while(r < size) {
                    r = r << 1 | 1;
                    if(f(op(st[r], sm))) {
                        sm = op(st[r--], sm);
                    }
                }
                return r + 1 - size;
            }
            sm = op(st[r], sm);
        } while((r & -r) != r);
        return 0;
    }
private:
    int n, size, log;
    vector<S> st;
    inline void update(int v) { st[v] = op(st[v * 2], st[v * 2 + 1]); }
};
```

2.4 LazySegtree.h

```
template<class S,
         S (*e)(),
         S (*op)(S, S),
         class F,
         F (*id)(),
         S (*mapping)(F, S),
         F (*composition)(F, F)>
class lazy_segtree {
public:
    lazy_segtree() : lazy_segtree(0) {}
    explicit lazy_segtree(int _n) :
    lazy_segtree(vector<S>(_n, e())) {}
    explicit lazy_segtree(const vector<S>& v) :
    n(int(v.size())) {
        log = 31 - __builtin_clz(2 * n - 1);
        size = 1 << log;
        d = vector<S>(size << 1, e());
        lz = vector<F>(size, id());
        for(int i = 0; i < n; i++) d[size + i] =
    v[i];
        for(int i = size - 1; i; --i) update(i);
    }
    void set(int p, S x) {
        assert(0 <= p && p < n);
        p += size;
        for(int i = log; i; --i) push(p >> i);
        d[p] = x;
        for(int i = 1; i <= log; ++i) update(p >>
    i);
    }
    S get(int p) {
        assert(0 <= p && p < n);
        p += size;
        for(int i = log; i; i--) push(p >> i);
        return d[p];
    }
    S operator[](int p) { return get(p); }
    S prod(int l, int r) {
        assert(0 <= l && l <= r && r <= n);
        if(l == r) return e();
        l += size, r += size;
        for(int i = log; i; i--) {
            if(((l >> i) << i) != l) {
                push(l >> i);
            }
            if(((r >> i) << i) != r) {
                push(r >> i);
            }
        }
        S sml = e(), smr = e();
        while(l < r) {
            if(l & 1) sml = op(sml, d[l++]);
            if(r & 1) smr = op(d[--r], smr);
            l >>= 1, r >>= 1;
        }
        return op(sml, smr);
    }
    S all_prod() const { return d[1]; }
    void apply(int p, F f) {
        assert(0 <= p && p < n);
        p += size;
        for(int i = log; i; i--) push(p >> i);
        d[p] = mapping(f, d[p]);
        for(int i = 1; i <= log; i++) update(p >>
    i);
    }
    void apply(int l, int r, F f) {
        assert(0 <= l && l <= r && r <= n);
        if(l == r) return;
        l += size, r += size;
        for(int i = log; i; i--) {
            if(((l >> i) << i) != l) {
                push(l >> i);
            }
            if(((r >> i) << i) != r) {
                push((r - 1) >> i);
            }
        }
        {
            int l2 = l, r2 = r;
            while(l < r) {
                if(l & 1) all_apply(l++, f);
                if(r & 1) all_apply(--r, f);
                l >>= 1;
                r >>= 1;
            }
            l = l2, r = r2;
        }
        for(int i = 1; i <= log; i++) {
            if(((l >> i) << i) != l) {
                update(l >> i);
            }
```

```
            if(((r >> i) << i) != r) {
                update((r - 1) >> i);
            }
        }
    }
    template<bool (*g)(S)> int max_right(int l) {
        return max_right(l, [](S x) { return g(x); });
    }
    template<class G> int max_right(int l, G g) {
        assert(0 <= l && l <= n);
        assert(g(e()));
        if(l == n) return n;
        l += size;
        for(int i = log; i; i--) push(l >> i);
        S sm = e();
        do {
            while(!(l & 1)) {
                l >>= 1;
            }
            if(!g(op(sm, d[l]))) {
                while(l < size) {
                    push(l);
                    l <<= 1;
                    if(g(op(sm, d[l]))) sm = op(sm, d[l++]);
                }
                return l - size;
            }
            sm = op(sm, d[l++]);
        } while((l & -l) != l);
        return n;
    }
    template<bool (*g)(S)> int min_left(int r) {
        return min_left(r, [](S x) { return g(x); });
    }
    template<class G> int min_left(int r, G g) {
        assert(0 <= r && r <= n);
        assert(g(e()));
        if(r == 0) return 0;
        r += size;
        for(int i = log; i >= 1; i--) push((r - 1) >> i);
        S sm = e();
        do {
            r--;
            while(r > 1 && (r & 1)) r >>= 1;
            if(!g(op(d[r], sm))) {
                while(r < size) {
                    push(r);
                    r = r << 1 | 1;
                    if(g(op(d[r], sm))) sm = op(d[r--], sm);
                }
                return r + 1 - size;
            }
            sm = op(d[r], sm);
        } while((r & -r) != r);
        return 0;
    }
private:
    int n, size, log;
    vector<S> d;
    vector<F> lz;
    inline void update(int k) { d[k] = op(d[k << 1], d[k << 1 | 1]); }
    void all_apply(int k, F f) {
        d[k] = mapping(f, d[k]);
        if(k < size) {
            lz[k] = composition(f, lz[k]);
        }
    }
    void push(int k) {
        all_apply(k << 1, lz[k]);
        all_apply(k << 1 | 1, lz[k]);
        lz[k] = id();
    }
};
```

2.5 SparseTable.h

```
template<class T, T (*op)(T, T)> struct sparse_table {
    int n;
    vector<vector<T>> mat;
    sparse_table() : n(0) {}
    sparse_table(const vector<T>& a) {
        n = static_cast<int>(a.size());
        int max_log = 32 - __builtin_clz(n);
        mat.resize(max_log);
        mat[0] = a;
        for(int j = 1; j < max_log; ++j) {
            mat[j].resize(n - (1 << j) + 1);
            for(int i = 0; i <= n - (1 << j); ++i) {
                mat[j][i] = op(mat[j - 1][i], mat[j - 1][i + (1 << (j - 1))]);
            }
        }
    }
    inline T prod(int from, int to) const {
        assert(0 <= from && from <= to && to <= n - 1);
        int lg = 31 - __builtin_clz(to - from + 1);
        return op(mat[lg][from], mat[lg][to - (1 << lg) + 1]);
    }
};
```

2.6 PersistentSegtree.h

```
// 1. Set the value a in array k to x.
// 2. Calculate the sum of values in range [a,b] in array k.
// 3. Create a copy of array k and add it to the end of the list.
struct Node {
    ll val;
    Node* l;
    Node* r;
    Node(ll x = 0) : val(x), l(NULL), r(NULL) {}
    Node(Node* ll, Node* rr) : l(ll), r(rr) {
        val = (l ? l->val : 0) + (r ? r->val : 0);
    }
};
```

```
Node* build(int l, int r) {
    if(l + 1 == r) {
        ll x;
        cin >> x;
        return new Node(x);
    }
    int m = (l + r) / 2;
    return new Node(build(l, m), build(m, r));
}
Node* update(Node* v, int p, ll x, int l, int r) {
    if(l + 1 == r) return new Node(x);
    int m = (l + r) / 2;
    if(p < m) return new Node(update(v->l, p, x, l, m), v->r);
    else return new Node(v->l, update(v->r, p, x, m, r));
}
ll query(Node* v, int x, int y, int l, int r) {
    if(r <= x || l >= y) return 0;
    if(x <= l && r <= y) return v->val;
    int m = (l + r) / 2;
    return query(v->l, x, y, l, m) + query(v->r, x, y, m, r);
}
int main() {
    int n, q; cin >> n >> q;
    vector<Node*> version{build(0, n)};
    while(q--) {
        int tc;
        cin >> tc;
        if(tc == 1) {
            int k, p, x; cin >> k >> p >> x;
            --k, --p;
            version[k] = update(version[k], p, x, 0, n);
        } else if(tc == 2) {
            int k, l, r; cin >> k >> l >> r;
            --k, --l;
            cout << query(version[k], l, r, 0, n) << "\n";
        } else if(tc == 3) {
            int k; cin >> k;
            --k;
            version.push_back(version[k]);
        } else {
            assert(false);
        }
    }
    return 0;
}
```

2.7 ConvexHullTrick.h

```
struct Line_t {
    mutable ll k, m, p;
    inline bool operator<(const Line_t& o) const { return k < o.k; }
    inline bool operator<(ll x) const { return p < x; }
};
// return maximum (with minimum use negative coefficient and constant)
struct CHT : multiset<Line_t, less<>> {
    // (for doubles, use INF = 1/.0, div(a,b) = a/b)
    static const ll INF = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b);
    }
    bool isect(iterator x, iterator y) {
        if(y == end()) {
            x->p = INF;
            return 0;
        }
        if(x->k == y->k) x->p = (x->m > y->m ? INF : -INF);
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add_line(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while(isect(y, z)) z = erase(z);
        if(x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while((y = x) != begin() && (--x)->p >= y->p) isect(x, erase(y));
    }
    ll get(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};
```

2.8 LiChao.h

```
template<class T> struct LiChaoTree {
    static constexpr T INF = numeric_limits<T>::max();
    struct Line {
        T a, b;
        Line(T a, T b) : a(a), b(b) {}
        T operator()(T x) const { return a * x + b; }
    };
    int n;
    vector<Line> fs;
    vector<T> xs;
    LiChaoTree(const vector<T>& xs_) : xs(xs_) {
        sort(xs.begin(), xs.end());
        xs.erase(unique(xs.begin(), xs.end()), xs.end());
        n = SZ(xs);
        fs.assign(2 * n, Line(T(0), INF));
    }
    int index(T x) const { return lower_bound(xs.begin(), xs.end(), x) - xs.begin(); }
    void add_line(T a, T b) { apply(a, b, 0, n); }
    void add_segment(T a, T b, T xl, T xr) {
        int l = index(xl), r = index(xr);
        apply(a, b, l, r);
    }
    inline T get(T x) const {
        int i = index(x);
        T res = INF;
```

```
        for(i += n; i; i >>= 1) chmin(res, fs[i](x));
        return res;
    }
    void apply(T a, T b, int l, int r) {
        Line g(a, b);
        for(l += n, r += n; l < r; l >>= 1, r >>= 1) {
            if(l & 1) push(g, l++);
            if(r & 1) push(g, --r);
        }
    }
    void push(Line g, int i) {
        int l = i, r = i + 1;
        while(l < n) l <<= 1, r <<= 1;
        while(l < r) {
            int c = (l + r) / 2;
            T xl = xs[l - n], xr = xs[r - 1 - n], xc = xs[c - n];
            Line& f = fs[i];
            if(f(xl) <= g(xl) && f(xr) <= g(xr)) return;
            if(f(xl) >= g(xl) && f(xr) >= g(xr)) {
                f = g;
                return;
            }
            if(f(xc) > g(xc)) swap(f, g);
            if(f(xl) > g(xl)) {
                i = 2 * i;
                r = c;
            } else {
                i = 2 * i + 1;
                l = c;
            }
        }
    }
};
```

2.9 Treap.h

```
template<class S,
         S (*e)(),
         S (*op)(S, S),
         class F,
         F (*id)(),
         S (*mapping)(F, S),
         F (*composition)(F, F)>
class Treap {
public:
    struct Node {
        S val, range;
        F tag;
        bool rev = false;
        int size = 1;
        int pri;
        Node* l = NULL;
        Node* r = NULL;
        Node() : Node(e()) {}
        Node(const S& s) : val(s), range(s), tag(id()), pri(rng()) {}
    };
    static int size(Node*& v) { return (v ? v->size : 0); }
    static Node* merge(Node* a, Node* b) {
        if(!a || !b) return (a ? a : b);
        push(a);
        push(b);
        if(a->pri > b->pri) {
            a->r = merge(a->r, b);
            pull(a);
            return a;
        } else {
            b->l = merge(a, b->l);
            pull(b);
            return b;
        }
    }
    static void split(Node* v, Node*& a, Node*& b, int k) {
        if(k == 0) {
            a = NULL;
            b = v;
            return;
        }
        push(v);
        if(size(v->l) >= k) {
            b = v;
            split(v->l, a, v->l, k);
            pull(b);
        } else {
            a = v;
            split(v->r, v->r, b, k - size(v->l) - 1);
            pull(a);
        }
    }
    static void print(Node* v) {
        if(!v) return;
        push(v);
        print(v->l);
        cout << v->val << " ";
        print(v->r);
    }
private:
    static void pull(Node*& v) {
        v->size = 1 + size(v->l) + size(v->r);
        v->range = v->val;
        if(v->l) v->range = op(v->l->range, v->range);
        if(v->r) v->range = op(v->range, v->r->range);
    }
    static void push(Node*& v) {
        if(v->rev) {
            swap(v->l, v->r);
            if(v->l) v->l->rev ^= 1;
            if(v->r) v->r->rev ^= 1;
            v->rev = false;
        }
        if(v->tag != id()) {
            v->val = mapping(v->tag, v->val);
            if(v->l) v->l->tag = composition(v->tag, v->l->tag);
            if(v->r) v->r->tag = composition(v->tag, v->r->tag);
            v->tag = id();
        }
    }
```

```
};
using TP = Treap<S, e, op, F, id, mapping, composition>;
```

2.10 Chtholly.h

```
struct ODT {
    struct S {
        int l, r;
        mutable int v;
        S(int L, int R = -1, int V = 0) : l(L), r(R), v(V) {}
        bool operator<(const S& s) const { return l < s.l; }
    };
    using IT = set<S>::iterator;
    set<S> seg;
    ODT() { seg.insert(S(0, maxn)); }
    IT split(int x) {
        IT it = --seg.upper_bound(S(x));
        if(it->l == x) return it;
        int l = it->l, r = it->r, v = it->v;
        seg.erase(it);
        seg.insert(S(l, x - 1, v));
        return seg.insert(S(x, r, v)).first;
    }
    void assign(int l, int r, int v) {
        IT itr = split(r + 1), it = split(l);
        seg.erase(it, itr);
        seg.insert(S(l, r, v));
    }
};
```

3 Graph

3.1 SCC.h

```
struct SCC {
    int n;
    vector<vector<int>> g, h;
    SCC() : SCC(0) {}
    SCC(int _n) : n(_n), g(_n), h(_n) {}
    void add_edge(int u, int v) {
        assert(0 <= u && u < n);
        assert(0 <= v && v < n);
        g[u].PB(v); h[v].PB(u);
    }
    vector<int> solve() {
        vector<int> id(n), top;
        top.reserve(n);
        function<void(int)> dfs1 = [&](int u) {
            id[u] = 1;
            for(auto v : g[u]) {
                if(id[v] == 0) dfs1(v);
            }
            top.PB(u);
        };
        for(int i = 0; i < n; ++i) {
            if(id[i] == 0) dfs1(i);
        }
        fill(id.begin(), id.end(), -1);
        function<void(int, int)> dfs2 = [&](int u, int x) {
            id[u] = x;
            for(auto v : h[u]) {
                if(id[v] == -1) dfs2(v);
            }
        };
        for(int i = n - 1, cnt = 0; i >= 0; --i) {
            int u = top[i];
            if(id[u] == -1) dfs2(u, cnt++);
        }
        return id;
    }
};
```

3.2 TwoSat.h

```
struct TwoSat {
    int n;
    SCC g;
    TwoSat() : TwoSat(0) {}
    TwoSat(int _n) : n(_n), g(_n * 2) {}
    void add_clause(int u, bool x, int v, bool y) {
        g.add_edge(2 * u + !x, 2 * v + y);
        g.add_edge(2 * v + !y, 2 * u + x);
    }
    pair<bool, vector<bool>> solve() {
        auto id = g.solve();
        vector<bool> ans(n);
        for(int i = 0; i < n; ++i) {
            if(id[2 * i] == id[2 * i + 1]) return {false, {}};
            ans[i] = (id[2 * i] < id[2 * i + 1]);
        }
        return {true, ans};
    }
};
```

3.3 LCA.h

```
struct LCA {
    LCA() : LCA(0) {}
    LCA(int _n) : n(_n), g(_n) {}
    static pii __lca_op(pii a, pii b) { return min(a, b); }
    void add_edge(int u, int v) {
        assert(0 <= u && u < n);
        assert(0 <= v && v < n);
        g[u].PB(v); g[v].PB(u);
    }
    void build(int root = 0) {
        assert(0 <= root && root < n);
        depth.assign(n, 0);
        parent.assign(n, -1);
        subtree_size.assign(n, 1);
        euler.reserve(2 * n - 1);
        first_occurrence.assign(n, 0);
        tour_list.reserve(n);
        tour_start.assign(n, 0);
```

```
        function<void(int, int, int)> dfs = [&](int u, int p, int d) {
            parent[u] = p;
            depth[u] = d;
            first_occurrence[u] = SZ(euler);
            euler.PB(u);
            pii heavy = {-1, -1};
            for(auto& v : g[u]) {
                if(v == p) continue;
                dfs(v, u, d + 1);
                subtree_size[u] += subtree_size[v];
                if(subtree_size[v] > heavy.F) heavy = {subtree_size[v], v};
                euler.PB(u);
            }
            sort(ALL(g[u]), [&](int a, int b) {
                return subtree_size[a] > subtree_size[b];
            });
        };
        dfs(root, -1, 0);
        heavy_root.assign(n, 0);
        function<void(int, bool)> dfs2 = [&](int u, bool is_heavy) {
            tour_start[u] = SZ(tour_list);
            tour_list.PB(u);
            heavy_root[u] = (is_heavy ? heavy_root[parent[u]] : u);
            bool heavy = true;
            for(auto& v : g[u]) {
                if(v == parent[u]) continue;
                dfs2(v, heavy);
                heavy = false;
            }
        };
        dfs2(root, false);
        vector<pii> route;
        route.reserve(SZ(euler));
        for(auto& u : euler) route.EB(depth[u], u);
        st = sparse_table<pii, __lca_op>(route);
    }
    inline int dist(int u, int v) const {
        return depth[u] + depth[v] - 2 * depth[lca(u, v)];
    }
    pair<int, array<int, 2>> get_diameter() const {
        pii u_max = {-1, -1};
        pii ux_max = {-1, -1};
        pair<int, array<int, 2>> uxv_max = {-1, {-1, -1}};
        for(int u : euler) {
            u_max = max(u_max, {depth[u], u});
            ux_max = max(ux_max, {u_max.F - 2 * depth[u], u_max.S});
            uxv_max = max(uxv_max, {ux_max.F + depth[u], {ux_max.S, u}});
        }
        return uxv_max;
    }
    inline int kth_ancestor(int u, int k) const {
        if(depth[u] < k) return -1;
        while(k > 0) {
            int root = heavy_root[u];
            if(depth[root] <= depth[u] - k) return tour_list[tour_start[u] - k];
            k -= depth[u] - depth[root] + 1;
            u = parent[root];
        }
        return u;
    }
    inline int kth_node_on_path(int a, int b, int k) const {
        int z = lca(a, b);
        int fi = depth[a] - depth[z], se = depth[b] - depth[z];
        assert(0 <= k && k <= fi + se);
        if(k < fi) return kth_ancestor(a, k);
        else return kth_ancestor(b, fi + se - k);
    }
    int lca(int u, int v) const {
        assert(0 <= u && u < n);
        assert(0 <= v && v < n);
        int l = first_occurrence[u], r = first_occurrence[v];
        return st.prod(min(l, r), max(l, r)).S;
    }
    int n;
    vector<vector<int>> g;
    vector<int> parent, depth, subtree_size;
    vector<int> euler, first_occurrence, tour_list, tour_start, heavy_root;
    sparse_table<pii, __lca_op> st;
};
```

3.4 HLD.h

```
struct HLD : LCA {
public:
    using LCA::add_edge;
    using LCA::build;
    using LCA::dist;
    using LCA::get_diameter;
    using LCA::kth_ancestor;
    using LCA::kth_node_on_path;
    using LCA::lca;
    HLD() : HLD(0) {}
    HLD(int _n) : LCA(_n) {}
    inline int get(int u) const { return tour_start[u]; }
    // return path_{[u,...,p)} where p is an ancestor of u
    vector<pii> path_up(int u, int p) const {
        vector<pii> seg;
        while(heavy_root[u] != heavy_root[p]) {
            seg.EB(get(heavy_root[u]), get(u) + 1);
            u = parent[heavy_root[u]];
        }
        // id_p is smaller than id_u but we don't want id_p
        seg.EB(get(p) + 1, get(u) + 1);
        return seg;
    }
    vector<pii> path(int u, int v) const {
        int z = lca(u, v);
        auto lhs = path_up(u, z);
        auto rhs = path_up(v, z);
        lhs.EB(get(z), get(z) + 1);
        lhs.insert(lhs.end(), ALL(rhs));
        return lhs;
```

```
    }
};
```

3.5 Dinic.h

```
template<class T> struct Dinic {
    struct Edge {
        int to;
        T cap;
        Edge(int _to, T _cap) : to(_to), cap(_cap) {}
    };
    static constexpr T INF = numeric_limits<T>::max() / 2;
    int n;
    vector<Edge> e;
    vector<vector<int>> g;
    vector<int> cur, h;
    Dinic() {}
    Dinic(int _n) : n(_n), g(_n) {}
    void add_edge(int u, int v, T c) {
        assert(0 <= u && u < n);
        assert(0 <= v && v < n);
        g[u].PB(SZ(e)); e.EB(v, c);
        g[v].PB(SZ(e)); e.EB(u, 0);
    }
    bool bfs(int s, int t) {
        h.assign(n, -1);
        queue<int> que;
        h[s] = 0;
        que.push(s);
        while(!que.empty()) {
            int u = que.front(); que.pop();
            for(int i : g[u]) {
                int v = e[i].to;
                T c = e[i].cap;
                if(c > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    if(v == t) return true;
                    que.push(v);
                }
            }
        }
        return false;
    }
    T dfs(int u, int t, T f) {
        if(u == t) return f;
        T r = f;
        for(int &i = cur[u]; i < SZ(g[u]); ++i) {
            int j = g[u][i];
            int v = e[j].to;
            T c = e[j].cap;
            if(c > 0 && h[v] == h[u] + 1) {
                T a = dfs(v, t, min(r, c));
                e[j].cap -= a;
                e[j ^ 1].cap += a;
                r -= a;
                if(r == 0) return f;
            }
        }
        return f - r;
    }
    T flow(int s, int t) {
        assert(0 <= s && s < n);
        assert(0 <= t && t < n);
        T ans = 0;
        while(bfs(s, t)) {
            cur.assign(n, 0);
            ans += dfs(s, t, INF);
        }
        return ans;
    }
};
```

3.6 MCMF.h

```
template<class Cap_t, class Cost_t> struct MCMF {
    struct Edge {
        int from;
        int to;
        Cap_t cap;
        Cost_t cost;
        Edge(int u, int v, Cap_t _cap, Cost_t _cost) : from(u), to(v), cap(_cap), cost(_cost) {}
    };
    static constexpr Cap_t EPS = static_cast<Cap_t>(1e-9);
    int n;
    vector<Edge> edges;
    vector<vector<int>> g;
    vector<Cost_t> d;
    vector<bool> in_queue;
    vector<int> previous_edge;
    MCMF(int _n) : n(_n), g(_n), d(_n), in_queue(_n), previous_edge(_n) {}
    void add_edge(int u, int v, Cap_t cap, Cost_t cost) {
        assert(0 <= u && u < n);
        assert(0 <= v && v < n);
        g[u].PB(SZ(edges));
        edges.EB(u, v, cap, cost);
        g[v].PB(SZ(edges));
        edges.EB(v, u, 0, -cost);
    }
    bool bfs(int s, int t) {
        bool found = false;
        fill(d.begin(), d.end(), numeric_limits<Cost_t>::max());
        d[s] = 0;
        in_queue[s] = true;
        queue<int> que;
        que.push(s);
        while(!que.empty()) {
            int u = que.front(); que.pop();
            if(u == t) found = true;
            in_queue[u] = false;
            for(auto& id : g[u]) {
                const Edge& e = edges[id];
                if(e.cap > EPS && d[u] + e.cost < d[e.to]) {
                    d[e.to] = d[u] + e.cost;
                    previous_edge[e.to] = id;
                    if(!in_queue[e.to]) {
                        que.push(e.to);
                        in_queue[e.to] = true;
```

```
                    }
                }
            }
        }
        return found;
    }
    pair<Cap_t, Cost_t> flow(int s, int t) {
        assert(0 <= s && s < n);
        assert(0 <= t && t < n);
        Cap_t cap = 0;
        Cost_t cost = 0;
        while(bfs(s, t)) {
            Cap_t send =
↪   numeric_limits<Cap_t>::max();
            int u = t;
            while(u != s) {
                const Edge& e =
↪   edges[previous_edge[u]];
                send = min(send, e.cap);
                u = e.from;
            }
            u = t;
            while(u != s) {
                Edge& e = edges[previous_edge[u]];
                e.cap -= send;
                Edge& b = edges[previous_edge[u] ^
↪   1];
                b.cap += send;
                u = e.from;
            }
            cap += send;
            cost += send * d[t];
        }
        return make_pair(cap, cost);
    }
};
```

3.7 BipartiteMatching.h

```
vector<int> v[Nx];
bitset<Nx> vis;
int mp[Nx],mq[Mx];
bool dfs(int x){
    vis[x]=1;
    for(int i : v[x]) if(!~mq[i] || !vis[mq[i]] &&
↪   dfs(mq[i])) return mq[ mp[x] = i ] = x, 1;
    return 0;
}
int matching(){
    int ans = 0;
    memset(mq, -1, sizeof(mq)), memset(mp, -1,
↪   sizeof(mp));
    for(int i = 0; i < n; ++i) {
        vis.reset();
        if(dfs(i)) ans += 1;
    }
    return ans;
}
```

3.8 ArticulationPoints.h

```
vector<int> ArticulationPoints(const
↪   vector<vector<int>>& g) {
    int n = SZ(g);
    vector<int> id(n, -1), low(n), cuts;
    function<void(int, int)> dfs = [&](int u, int
↪   p) {
        static int cnt = 0;
        id[u] = low[u] = cnt++;
        int child = 0;
        bool isCut = false;
        for(auto v : g[u]) {
            if(v == p) continue;
            if(id[v] != -1) low[u] = min(low[u],
↪   id[v]);
            else {
                dfs(v, u);
                low[u] = min(low[u], low[v]);
                if(low[v] >= id[u] && p != -1)
↪   isCut = true;
                child += 1;
            }
        }
        if(p == -1 && child > 1) isCut = true;
        if(isCut) cuts.PB(u);
    };
    for(int i = 0; i < n; ++i) {
        if(id[i] == -1) dfs(i, -1);
    }
    return cuts;
}
```

3.9 Bridges.h

```
vector<pii> findBridges(const vector<vector<int>>&
↪   g) {
    int n = (int) g.size();
    vector<int> id(n, -1), low(n);
    vector<pii> bridges;
    function<void(int, int)> dfs = [&](int u, int
↪   p) {
        static int cnt = 0;
        id[u] = low[u] = cnt++;
        for(auto v : g[u]) {
            if(v == p) continue;
            if(id[v] != -1) low[u] = min(low[u],
↪   id[v]);
            else {
                dfs(v, u);
                low[u] = min(low[u], low[v]);
                if(low[v] > id[u]) bridges.EB(u,
↪   v);
            }
        }
    };
    for(int i = 0; i < n; ++i) {
        if(id[i] == -1) dfs(i, -1);
    }
    return bridges;
}
```

3.10 Hungarian.h

```cpp
pair<ll, vector<pair<int, int>>> Hungarian(const vector<vector<ll>>& g) {
    const ll INF = LLONG_MAX;
    int n = SZ(g) + 1, m = SZ(g[0]) + 1;
    vector<vector<ll>> adj(n, vector<ll>(m));
    for(int i = 0; i < n - 1; ++i) {
        for(int j = 0; j < m - 1; ++j) {
            adj[i + 1][j + 1] = g[i][j];
        }
    }
    vector<ll> u(n), v(m);
    vector<int> match(m);
    for(int i = 1; i < n; i++) {
        int w = 0;
        match[w] = i;
        vector<ll> dist(m, INF);
        vector<int> pred(m, -1);
        vector<bool> vis(m);
        while(match[w]) {
            vis[w] = true;
            int cur = match[w], nw = 0;
            ll delta = INF;
            for(int j = 1; j < m; j++) {
                if(!vis[j]) {
                    ll edge = adj[cur][j] - u[cur] - v[j];
                    if(edge < dist[j]) {
                        dist[j] = edge;
                        pred[j] = w;
                    }
                    if(dist[j] < delta) {
                        delta = dist[j];
                        nw = j;
                    }
                }
            }
            for(int j = 0; j < m; ++j) {
                if(vis[j]) {
                    u[match[j]] += delta;
                    v[j] -= delta;
                } else dist[j] -= delta;
            }
            w = nw;
        }
        while(w) {
            int nw = pred[w];
            match[w] = match[nw];
            w = nw;
        }
    }
    vector<pii> res;
    for(int i = 1; i < n; ++i) res.EB(match[i] - 1, i - 1);
    return {-v[0], res};
}
```

3.11 FlowModels

- Maximum/Minimum flow with lower bound / Circulation problem
  1. Construct super source S and sink T.
  2. For each edge (x, y, l, u), connect $x \to y$ with capacity $u - l$.
  3. For each vertex v, denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
  4. If $in(v) > 0$, connect $S \to v$ with capacity $in(v)$, otherwise, connect $v \to T$ with capacity $-in(v)$.
     - To maximize, connect $t \to s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T. If $f \neq \sum_{v \in V, in(v)>0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
     - To minimize, let f be the maximum flow from S to T. Connect $t \to s$ with capacity ∞ and let the flow from S to T be f'. If $f + f' \neq \sum_{v \in V, in(v)>0} in(v)$, there's no solution. Otherwise, f' is the answer.
  5. The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.
- Construct minimum vertex cover from maximum matching M on bipartite graph (X, Y)
  1. Redirect every edge: $y \to x$ if $(x, y) \in M$, $x \to y$ otherwise.
  2. DFS from unmatched vertices in X.
  3. $x \in X$ is chosen iff x is unvisited.
  4. $y \in Y$ is chosen iff y is visited.
- Minimum cost cyclic flow
  1. Consruct super source S and sink T
  2. For each edge (x, y, c), connect $x \to y$ with $(cost, cap) = (c, 1)$ if $c > 0$, otherwise connect $y \to x$ with $(cost, cap) = (-c, 1)$
  3. For each edge with $c < 0$, sum these cost as K, then increase $d(y)$ by 1, decrease $d(x)$ by 1
  4. For each vertex v with $d(v) > 0$, connect $S \to v$ with $(cost, cap) = (0, d(v))$
  5. For each vertex v with $d(v) < 0$, connect $v \to T$ with $(cost, cap) = (0, -d(v))$
  6. Flow from S to T, the answer is the cost of the flow $C + K$
- Maximum density induced subgraph
  1. Binary search on answer, suppose we're checking answer T
  2. Construct a max flow model, let K be the sum of all weights
  3. Connect source $s \to v$, $v \in G$ with capacity K
  4. For each edge (u, v, w) in G, connect $u \to v$ and $v \to u$ with capacity w
  5. For $v \in G$, connect it with sink $v \to t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
  6. T is a valid answer if the maximum flow $f < KV$
- Minimum weight edge cover
  1. For each $v \in V$ create a copy v', and connect $u' \to v'$ with weight $w(u, v)$.
  2. Connect $v \to v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v.
  3. Find the minimum weight perfect matching on G'.
- Project selection problem
  1. If $p_v > 0$, create edge (s, v) with capacity p_v; otherwise, create edge (v, t) with capacity $-p_v$.
  2. Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v.
  3. The mincut is equivalent to the maximum profit of a subset of projects.

- 0/1 quadratic programming
$$\sum_x c_x x + \sum_y c_y \bar{y} + \sum_{xy} c_{xy} x\bar{y} + \sum_{xyx'y'} c_{xyx'y'}(x\bar{y} + x'\bar{y'})$$
can be minimized by the mincut of the following graph:
  1. Create edge (x, t) with capacity c_x and create edge (s, y) with capacity c_y.
  2. Create edge (x, y) with capacity c_{xy}.
  3. Create edge (x, y) and edge (x', y') with capacity $c_{xyx'y'}$.

4 String

4.1 RollingHash.h

```
template<class T> struct Rolling_Hash {
    Rolling_Hash() {}
    Rolling_Hash(int _A, string _s) : A(_A), n((int) _s.size()), s(_s), pref(n) {
        pref[0] = s[0];
        for(int i = 1; i < n; ++i) pref[i] = pref[i - 1] * A + s[i];
    }
    inline int size() const { return n; }
    inline T get(int l, int r) const {
        assert(0 <= l && l <= r && r < n);
        if(l == 0) return pref[r];
        return pref[r] - pref[l - 1] * T(pow_mod_constexpr(A, r - l + 1, T::mod()));
    }
    inline T id() const { return pref.back(); }
    int A, n;
    string s;
    vector<T> pref;
};
```

4.2 KMP.h

```
template<class T> vector<int> KMP(const vector<T>& a) {
    int n = SZ(a);
    vector<int> k(n);
    for(int i = 1; i < n; ++i) {
        int j = k[i - 1];
        while(j > 0 && a[i] != a[j]) j = k[j - 1];
        j += (a[i] == a[j]);
        k[i] = j;
    }
    return k;
}
```

4.3 DynamicKMP.h

```
template<int ALPHABET, int (*f)(char)>
struct DynamicKMP {
    vector<int> p;
    vector<array<int, ALPHABET>> dp;
    DynamicKMP() {}
    DynamicKMP(const string& s) {
        reserve(SZ(s));
        for(const char& c : s) push(c);
    }
    void push(char c) {
        int v = f(c);
        dp.EB();
        dp.back()[v] = SZ(dp);
        if(p.empty()) {
            p.PB(0);
            return;
        }
        int i = SZ(p);
        for(int j = 0; j < ALPHABET; ++j) {
            if(j == v) p.PB(dp[p[i - 1]][j]);
            else dp.back()[j] = dp[p[i - 1]][j];
        }
    }
    void pop() { p.PPB(); dp.PPB(); }
    int query() const { return p.back(); }
    vector<int> query_all() const { return p; }
    void reserve(int sz) { p.reserve(sz); dp.reserve(sz); }
};
```

4.4 Z.h

```
template<class T>
vector<int> z_algorithm(const vector<T>& a) {
    int n = SZ(a);
    vector<int> z(n);
    for(int i = 1, j = 0; i < n; ++i) {
        if(i <= j + z[j]) z[i] = min(z[i - j], j + z[j] - i);
        while(i + z[i] < n && a[i + z[i]] == a[z[i]]) z[i] += 1;
        if(i + z[i] > j + z[j]) j = i;
    }
    return z;
}
```

4.5 Manacher.h

```
template<class T>
vector<int> manacher_odd(const vector<T>& a) {
    vector<T> b(1, -87);
    b.insert(b.end(), ALL(a));
    b.PB(-69);
    int n = SZ(b);
    vector<int> z(n);
    z[0] = 1;
    for(int i = 1, l = -1, r = 1; i <= n; ++i) {
        if(i < r) z[i] = min(z[l + r - i], r - i);
        while(b[i - z[i]] == b[i + z[i]]) z[i]++;
        if(i + z[i] - 1 > r) {
            l = i - z[i] + 1;
            r = i + z[i] - 1;
        }
    }
    return vector<int>(1 + ALL(z) - 1);
}
```

4.6 SmallestRotation.h

```cpp
string SmallestRotation(string s) {
    int n = SZ(s), i = 0, j = 1;
    s += s;
    while(i < n && j < n) {
        int k = 0;
        while(k < n && s[i + k] == s[j + k]) ++k;
        if(s[i + k] <= s[j + k]) j += k + 1;
        else i += k + 1;
        j += (i == j);
    }
    return s.substr(i < n ? i : j, n);
}
```

4.7 SuffixArray.h

```cpp
vector<int> sa_is(const vector<int>& s, int upper) {
    int n = SZ(s);
    if(n == 0) return {};
    if(n == 1) return {0};
    if(n == 2) {
        if(s[0] < s[1]) return {0, 1};
        else return {1, 0};
    }
    vector<int> sa(n);
    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]);
    }
    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 vector<int>& lms) {
        fill(ALL(sa), -1);
        vector<int> buf(upper + 1);
        copy(ALL(sum_s), buf.begin());
        for(auto d : lms) {
            if(d == n) continue;
            sa[buf[s[d]]++] = d;
        }
        copy(ALL(sum_l), 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;
        }
        copy(ALL(sum_l), 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;
        }
    };
    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++;
    }
    vector<int> lms;
    lms.reserve(m);
    for(int i = 1; i < n; i++) {
        if(!ls[i - 1] && ls[i]) lms.PB(i);
    }
    induce(lms);
    if(m) {
        vector<int> sorted_lms;
        sorted_lms.reserve(m);
        for(int v : sa) {
            if(lms_map[v] != -1) sorted_lms.PB(v);
        }
        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(rec_s, rec_upper);
        for(int i = 0; i < m; i++) sorted_lms[i] = lms[rec_sa[i]];
        induce(sorted_lms);
    }
    return sa;
}
```

4.8 LCP.h

```cpp
template<class T>
vector<int> lcp_array(const vector<T>& s, const vector<int>& sa) {
    int n = SZ(s);
    assert(n >= 1);
    vector<int> rnk(n);
    for(int i = 0; i < n; i++) rnk[sa[i]] = i;
    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;
}
```

4.9 AhoCorasick.h

```
template<int ALPHABET, int (*f)(char)>
struct AhoCorasick {
    vector<array<int, ALPHABET>> trie, to;
    vector<int> fail, cnt;
    AhoCorasick() : AhoCorasick(vector<string>())
    ↪ {}
    AhoCorasick(const vector<string>& S) {
        newNode();
        for(const auto& s : S) insert(s);
    }
    int insert(const string& s) {
        int p = 0;
        for(const char& c : s) p = next(p, f(c));
        cnt[p] += 1;
        return p;
    }
    inline int next(int u, int v) {
        if(!trie[u][v]) trie[u][v] = newNode();
        return trie[u][v];
    }
    void build_failure() {
        queue<int> que;
        for(int i = 0; i < ALPHABET; ++i) {
            if(trie[0][i]) {
                to[0][i] = trie[0][i];
                que.push(trie[0][i]);
            }
        }
        while(!que.empty()) {
            int u = que.front(); que.pop();
            for(int i = 0; i < 26; ++i) {
                if(trie[u][i]) to[u][i] =
    ↪ trie[u][i];
                else to[u][i] = to[fail[u]][i];
            }
            for(int i = 0; i < 26; ++i) {
                if(trie[u][i]) {
                    int p = trie[u][i];
                    int k = fail[u];
                    while(k && !trie[k][i]) k =
    ↪ fail[k];
                    if(trie[k][i]) k = trie[k][i];
                    fail[p] = k;
                    cnt[p] += cnt[k];
                    que.push(p);
                }
            }
        }
    }
    inline int newNode() {
        int sz = (int) trie.size();
        trie.EB();
        to.EB();
        fill(ALL(trie.back()), 0);
        fill(ALL(to.back()), 0);
        fail.EB();
        cnt.EB();
        return sz;
    }
};
```

5 Math

5.1 ExtendGCD.h

```
// @return x, y s.t. ax + by = gcd(a, b)
ll ext_gcd(ll a, ll b, ll& x, ll& y) {
    if(b == 0) {
        x = 1; y = 0;
        return a;
    }
    ll x2, y2;
    ll c = a % b;
    if(c < 0) c += b;
    ll g = ext_gcd(b, c, x2, y2);
    x = y2;
    y = x2 - (a / b) * y2;
    return g;
}
```

5.2 InvGCD.h

```
pair<ll, ll> inv_gcd(ll a, ll b) {
    a %= b;
    if(a < 0) a += b;
    if(a == 0) return {b, 0};
    ll s = b, t = a;
    ll m0 = 0, m1 = 1;
    while(t) {
        ll u = s / t;
        s -= t * u;
        m0 -= m1 * u;
        swap(s, t);
        swap(m0, m1);
    }
    if(m0 < 0) m0 += b / s;
    return {s, m0};
}
```

5.3 Modint.h

```
template<int m>
struct modint {
    static constexpr int mod() { return m; }
    modint() : val(0) {}
    modint(long long v) {
        v %= mod();
        if(v < 0) v += mod();
```

```
        val = v;
    }
    const int& operator()() const { return val; }
    modint& operator+=(const modint& other) {
        val += other.val;
        if(val >= mod()) val -= mod();
        return *this;
    }
    modint& operator-=(const modint& other) {
        val -= other.val;
        if(val < 0) val += mod();
        return *this;
    }
    modint& operator*=(const modint& other) {
        val = 1LL * val * other.val % mod();
        return *this;
    }
    modint& operator/=(const modint& other) {
        auto eg = inv_gcd(other.val, mod());
        assert(eg.F == 1);
        return *this *= eg.S;
    }
    template<class T> modint& operator+=(const T&
    ↪ other) { return *this += modint(other); }
    template<class T> modint& operator-=(const T&
    ↪ other) { return *this -= modint(other); }
    template<class T> modint& operator*=(const T&
    ↪ other) { return *this *= modint(other); }
    template<class T> modint& operator/=(const T&
    ↪ other) { return *this /= modint(other); }
    modint operator+() const { return *this }
    modint operator-() const { return modint() -
    ↪ *this; }
    modint operator+(const modint& other) { return
    ↪ modint(*this) += other; }
    modint operator-(const modint& other) { return
    ↪ modint(*this) -= other; }
    modint operator*(const modint& other) { return
    ↪ modint(*this) *= other; }
    modint operator/(const modint& other) { return
    ↪ modint(*this) /= other; }
    int val;
};
template<int m, class T> modint<m> operator+(const
↪ T& lhs, const modint<m>& rhs) {
    return modint<m>(lhs) += rhs;
}
template<int m, class T> modint<m> operator-(const
↪ T& lhs, const modint<m>& rhs) {
    return modint<m>(lhs) -= rhs;
}
template<int m, class T> modint<m> operator*(const
↪ T& lhs, const modint<m>& rhs) {
    return modint<m>(lhs) *= rhs;
}
template<int m, class T> modint<m> operator/(const
↪ T& lhs, const modint<m>& rhs) {
    return modint<m>(lhs) /= rhs;
}
```

5.4 ModInverses.h

```
vector<int> mod_inverse(int m, int n = -1) {
    assert(n < m);
    if(n == -1) n = m - 1;
    vector<int> inv(n + 1);
    inv[0] = inv[1] = 1;
    for(int i = 2; i <= n; ++i) inv[i] = m - 1LL *
    ↪ (m / i) * inv[m % i] % m;
    return inv;
}
```

5.5 PowMod.h

```
constexpr long long pow_mod_constexpr(long long x,
↪ long long n, int m) {
    if(m == 1) return 0;
    unsigned int _m = (unsigned int)(m);
    unsigned long long r = 1;
    x %= m;
    if(x < 0) x += m;
    unsigned long long y = x;
    while(n) {
        if(n & 1) r = (r * y) % _m;
        y = (y * y) % _m;
        n >>= 1;
    }
    return r;
}
```

5.6 DiscreteLog.h

```
int DiscreteLog(int s, int x, int y, int m) {
    constexpr int K = 0;
    hash_map<int, int> p;
    int b = 1;
    for(int i = 0; i < K; ++i) {
        p[y] = i;
        y = 1LL * y * x % m;
        b = 1LL * b * x % m;
    }
    for(int i = 0; i < m + 10; i += K) {
        s = 1LL * s * b % m;
        if(p.find(s) != p.end()) return i + K -
    ↪ p[s];
    }
    return -1;
}
int DiscreteLog(int x, int y, int m) {
    if(m == 1) return 0;
    int s = 1;
    for(int i = 0; i < 100; ++i) {
        if(s == y) return i;
        s = 1LL * s * x % m;
    }
    if(s == y) return 100;
    int p = 100 + DiscreteLog(s, x, y, m);
    return (pow_mod(x, p, m) != y ? -1 : p);
}
```

5.7 CRT.h

```cpp
// @return
//     remainder, modulo
//             or
//     0, 0 if do not exist
pair<ll, ll> crt(const vector<ll>& r, const vector<ll>& m) {
    assert(SZ(r) == SZ(m));
    int n = SZ(r);
    // Contracts: 0 <= r0 < m0
    ll r0 = 0, m0 = 1;
    for(int i = 0; i < n; i++) {
        assert(1 <= m[i]);
        ll r1 = r[i] % m[i];
        if(r1 < 0) r1 += m[i];
        ll m1 = m[i];
        if(m0 < m1) {
            swap(r0, r1);
            swap(m0, m1);
        }
        if(m0 % m1 == 0) {
            if(r0 % m1 != r1) return {0, 0};
            continue;
        }
        ll g, im;
        tie(g, im) = inv_gcd(m0, m1);
        ll u1 = (m1 / g);
        if((r1 - r0) % g) return {0, 0};
        ll x = (r1 - r0) / g % u1 * im % u1;
        r0 += x * m0;
        m0 *= u1;
        if(r0 < 0) r0 += m0;
    }
    return {r0, m0};
}
```

5.8 MillerRabin.h

```cpp
constexpr bool is_prime_constexpr(int n) {
    if(n <= 1) return false;
    if(n == 2 || n == 7 || n == 61) return true;
    if(n % 2 == 0) return false;
    ll d = (n - 1) >> __builtin_ctz(n - 1);
    constexpr ll bases[3] = {2, 7, 61};
    for(ll a : bases) {
        ll t = d;
        ll y = pow_mod_constexpr(a, t, n);
        while(t != n - 1 && y != 1 && y != n - 1) {
            y = y * y % n;
            t <<= 1;
        }
        if(y != n - 1 && t % 2 == 0) return false;
    }
    return true;
}
template<int n> constexpr bool is_prime = is_prime_constexpr(n);
bool is_prime_ll(ull n) {
    static const vector<ull> SPRP = {
        2, 325, 9375, 28178, 450775, 9780504, 1795265022
    };
    if(n == 1 || n % 6 % 4 != 1) return (n | 1) == 3;
    ll t = __builtin_ctzll(n - 1), k = (n - 1) >> t;
    for(const ull &a : SPRP) {
        ull tmp = pow_mod(a, k, n);
        if(tmp <= 1 || tmp == n - 1) continue;
        for(int i = 0; i <= t; i++) {
            if(i == t) return false;
            tmp = __int128(tmp) * tmp % n;
            if(tmp == n - 1) break;
        }
    }
    return true;
}
```

5.9 PrimitiveRoot.h

```cpp
// m must be prime. return minimum primitive root
constexpr int primitive_root_constexpr(int m) {
    if(m == 2) return 1;
    if(m == 167772161) return 3;
    if(m == 469762049) return 3;
    if(m == 754974721) return 11;
    if(m == 998244353) return 3;
    int divs[20] = {};
    divs[0] = 2;
    int cnt = 1;
    int x = (m - 1) / 2;
    while(x % 2 == 0) x /= 2;
    for(int i = 3; (long long)(i)*i <= x; i += 2) {
        if(x % i == 0) {
            divs[cnt++] = i;
            while(x % i == 0) {
                x /= i;
            }
        }
    }
    if(x > 1) {
        divs[cnt++] = x;
    }
    for(int g = 2;; g++) {
        bool ok = true;
        for(int i = 0; i < cnt; i++) {
            if(pow_mod_constexpr(g, (m - 1) / divs[i], m) == 1) {
                ok = false;
                break;
            }
        }
        if(ok) return g;
    }
}
template<int m> constexpr int primitive_root = primitive_root_constexpr(m);
```

5.10 LinearSieve.h

```cpp
vector<bool> isprime;
vector<int> primes, phi, mobius;
```

```cpp
void linear_sieve(int n) {
    n += 1;
    isprime.resize(n);
    fill(2 + ALL(isprime), true);
    phi.resize(n); mobius.resize(n);
    phi[1] = mobius[1] = 1;
    for(int i = 2; i < n; ++i) {
        if(isprime[i]) {
            primes.PB(i);
            phi[i] = i - 1;
            mobius[i] = -1;
        }
        for(auto j : primes) {
            if(i * j >= n) break;
            isprime[i * j] = false;
            if(i % j == 0) {
                mobius[i * j] = 0;
                phi[i * j] = phi[i] * j;
                break;
            } else {
                mobius[i * j] = mobius[i] *
    mobius[j];
                phi[i * j] = phi[i] * phi[j];
            }
        }
    }
}
```

5.11 Factorizer.h

```cpp
template<class T>
vector<pair<T, int>> MergeFactors(const
    vector<pair<T, int>>& a, const vector<pair<T,
    int>>& b) {
    vector<pair<T, int>> c;
    int i = 0, j = 0;
    while(i < SZ(a) || j < SZ(b)) {
        if(i < SZ(a) && j < SZ(b) && a[i].F ==
    b[j].F) {
            c.EB(a[i].F, a[i].S + b[j].S);
            ++i, ++j;
            continue;
        }
        if(j == SZ(b) || (i < SZ(a) && a[i].F <
    b[j].F)) c.PB(a[i++]);
        else c.PB(b[j++]);
    }
    return c;
}
template<class T>
vector<pair<T, int>> RhoC(const T& n, const T& c) {
    if(n <= 1) return {};
    if(n % 2 == 0) return MergeFactors({{2, 1}},
    RhoC(n / 2, c));
    if(is_prime_constexpr(n)) return {{n, 1}};
    T x = 2, saved = 2, p = 1, lam = 1;
    while(true) {
        x = (x * x % n + c) % n;
        T g = __gcd(((x - saved) + n) % n, n);
        if(g != 1) return MergeFactors(RhoC(g, c +
    1), RhoC(n / g, c + 1));
        if(p == lam) {
            saved = x;
            p <<= 1;
            lam = 0;
        }
        lam += 1;
    }
    return {};
}
template<class T>
vector<pair<T, int>> Factorize(T n) {
    if(n <= 1) return {};
    return RhoC(n, T(1));
}
template<class T>
vector<T> BuildDivisorsFromFactors(const
    vector<pair<T, int>>& factors) {
    int total = 1;
    for(int i = 0; i < SZ(factors); ++i) total *=
    factors[i].second + 1;
    vector<T> divisors;
    divisors.reserve(total);
    divisors.PB(1);
    for(auto [p, cnt] : factors) {
        int sz = SZ(divisors);
        for(int i = 0; i < sz; ++i) {
            T cur = divisors[i];
            for(int j = 0; j < cnt; ++j) {
                cur *= p;
                divisors.PB(cur);
            }
        }
    }
    // sort(ALL(divisors));
    return divisors;
}
```

5.12 FloorSum.h

```cpp
// @param n < 2^32
// @param 1 <= m < 2^32
// @return sum_{i=0}^{n-1} \lfloor \frac{ai + b}{m}
    \rfloor \pmod{2^{64}}
ull floor_sum_unsigned(ull n, ull m, ull a, ull b)
    {
    ull ans = 0;
    while(true) {
        if(a >= m) {
            ans += n * (n - 1) / 2 * (a / m);
            a %= m;
        }
        if(b >= m) {
            ans += n * (b / m);
            b %= m;
        }
        ull y_max = a * n + b;
        if(y_max < m) break;
        n = (ull)(y_max / m);
        b = (ull)(y_max % m);
        swap(m, a);
    }
    return ans;
}
ll floor_sum(ll n, ll m, ll a, ll b) {
    assert(0 <= n && n < (1LL << 32));
```

```
    assert(1 <= m && m < (1LL << 32));
    ull ans = 0;
    if(a < 0) {
        ull a2 = (a % m + m) % m;
        ans -= 1ULL * n * (n - 1) / 2 * ((a2 - a) / m);
        a = a2;
    }
    if(b < 0) {
        ull b2 = (b % m + m) % m;
        ans -= 1ULL * n * ((b2 - b) / m);
        b = b2;
    }
    return ans + floor_sum_unsigned(n, m, a, b);
}
```

5.13 GaussJordan.h

```
int n;
double a[105][105];
void gauss_jorgan(){
    for(int i=0;i<n;++i){
        int mx=i;
        for(int j=i+1;j<n;++j)
    if(fabs(a[j][i])>fabs(a[mx][i])) mx=j;
        if(mx!=i) swap(a[i],a[mx]);
        if(a[i][i] < EPS) return cout<<"No
    Solution\n",void();
        for(int j=0;j<n;++j){
            if(i==j) continue;
            double df=a[j][i]/a[i][i];
            for(int k=i+1;k<=n;++k)
    a[j][k]-=a[i][k]*df;
        }
    }
    for(int i=0;i<n;++i)
    cout<<fixed<<setprecision(2)<<a[i][n]/a[i][i]<<"\n";
}
```

5.14 Combination.h

```
vector<mint> fact{1}, inv_fact{1};
void init_fact(int n) {
    while(SZ(fact) <= n) fact.PB(fact.back() *
    SZ(fact));
    int sz = SZ(inv_fact)
    if(sz >= n + 1) return;
    inv_fact.resize(n + 1);
    inv_fact[n] = 1 / fact.back();
    for(int i = n - 1; i >= sz; --i) {
        inv_fact[i] = inv_fact[i + 1] * (i + 1);
    }
}
mint binom(int n, int k) {
    if(k < 0 || k > n) return 0;
    init_fact(n);
    return fact[n] * inv_fact[k] * inv_fact[n - k];
}
mint permute(int n, int k) {
    if(k < 0 || k > n) return 0;
    init_fact(n);
    return fact[n] * inv_fact[n - k];
}
```

5.15 BitTransform.h

```
template<class T> void OrTransform(vector<T>& a) {
    const int n = SZ(a);
    assert((n & -n) == n);
    for(int i = 1; i < n; i <<= 1) {
        for(int j = 0; j < n; j += i << 1) {
            for(int k = 0; k < i; ++k) {
                a[i + j + k] += a[j + k];
            }
        }
    }
}
template<class T> void OrInvTransform(vector<T>& a)
    {
    const int n = SZ(a);
    assert((n & -n) == n);
    for(int i = 1; i < n; i <<= 1) {
        for(int j = 0; j < n; j += i << 1) {
            for(int k = 0; k < i; ++k) {
                a[i + j + k] -= a[j + k];
            }
        }
    }
}
template<class T> void AndTransform(vector<T>& a) {
    const int n = SZ(a);
    assert((n & -n) == n);
    for(int i = 1; i < n; i <<= 1) {
        for(int j = 0; j < n; j += i << 1) {
            for(int k = 0; k < i; ++k) {
                a[j + k] += a[i + j + k];
            }
        }
    }
}
template<class T> void AndInvTransform(vector<T>&
    a) {
    const int n = SZ(a);
    assert((n & -n) == n);
    for(int i = 1; i < n; i <<= 1) {
        for(int j = 0; j < n; j += i << 1) {
            for(int k = 0; k < i; ++k) {
                a[j + k] -= a[i + j + k];
            }
        }
    }
}
template<class T> void XorTransform(vector<T>& a) {
    const int n = SZ(a);
    assert((n & -n) == n);
    for(int i = 1; i < n; i <<= 1) {
        for(int j = 0; j < n; j += i << 1) {
            for(int k = 0; k < i; ++k) {
                T x = move(a[j + k]), y = move(a[i
    + j + k]);
                a[j + k] = x + y;
                a[i + j + k] = x - y;
            }
        }
```

```
    }
}
template<class T> void XorInvTransform(vector<T>& a) {
    XorTransform(a);
    T inv2 = T(1) / T((int) a.size());
    for(auto& x : a) {
        x *= inv2;
    }
}
// Compute c[k] = sum(a[i] * b[j]) for (i or j) = k.
// Complexity: O(n log n)
template<class T> vector<T> OrConvolution(vector<T> a, vector<T> b) {
    const int n = SZ(a);
    assert(n == SZ(b));
    OrTransform(a); OrTransform(b);
    for(int i = 0; i < n; ++i) a[i] *= b[i];
    OrInvTransform(a);
    return a;
}
// Compute c[k] = sum(a[i] * b[j]) for (i and j) = k.
// Complexity: O(n log n)
template<class T> vector<T>
    AndConvolution(vector<T> a, vector<T> b) {
    const int n = SZ(a);
    assert(n == SZ(b));
    AndTransform(a); AndTransform(b);
    for(int i = 0; i < n; ++i) a[i] *= b[i];
    AndInvTransform(a);
    return a;
}
// Compute c[k] = sum(a[i] * b[j]) for (i xor j) = k.
// Complexity: O(n log n)
template<class T> vector<T>
    XorConvolution(vector<T> a, vector<T> b) {
    const int n = SZ(a);
    assert(n == SZ(b));
    XorTransform(a); XorTransform(b);
    for (int i = 0; i < n; ++i) a[i] *= b[i];
    XorInvTransform(a);
    return a;
}
template<class T> vector<T>
    SubsetSumConvolution(const vector<T>& f, const vector<T>& g) {
    const int n = SZ(f);
    assert(n == SZ(g));
    assert((n & -n) == n);
    const int N = __lg(n);
    vector<vector<T>> fhat(N + 1, vector<T>(n));
    vector<vector<T>> ghat(N + 1, vector<T>(n));
    for(int mask = 0; mask < n; ++mask) {
        fhat[__builtin_popcount(mask)][mask] = f[mask];
        ghat[__builtin_popcount(mask)][mask] = g[mask];
    }
    for(int i = 0; i <= N; ++i)
    OrTransform(fhat[i]), OrTransform(ghat[i]);
    vector<vector<T>> h(N + 1, vector<T>(n));
    for(int mask = 0; mask < n; ++mask) {
        for(int i = 0; i <= N; ++i) {
            for(int j = 0; j <= i; ++j) {
                h[i][mask] += fhat[j][mask] * ghat[i - j][mask];
            }
        }
    }
    for(int i = 0; i <= N; ++i)
    OrInvTransform(h[i]);
    vector<T> result(n);
    for(int mask = 0; mask < n; ++mask) {
        result[mask] = h[__builtin_popcount(mask)][mask];
    }
    return result;
}
```

5.16 FFT.h

```
void FFT(vector<cd>& a, bool inv) {
    int n = SZ(a);
    for(int i = 1, j = 0; i < n; ++i) {
        int bit = n >> 1;
        for(; j & bit; bit >>= 1) j ^= bit;
        j ^= bit;
        if(i < j) swap(a[i], a[j]);
    }
    for(int len = 2; len <= n; len <<= 1) {
        const double ang = 2 * PI / len * (inv ? -1 : +1);
        cd rot(cos(ang), sin(ang));
        for(int i = 0; i < n; i += len) {
            cd w(1);
            for(int j = 0; j < len / 2; ++j) {
                cd u = a[i + j], v = a[i + j + len / 2] * w;
                a[i + j] = u + v;
                a[i + j + len / 2] = u - v;
                w *= rot;
            }
        }
    }
    if(inv) {
        for(auto& x : a) x /= n;
    }
}
```

5.17 Poly.h

```
vector<int> __bit_reorder;
template<class T>
class Poly {
public:
    static constexpr int R = primitive_root<T::mod()>;
    Poly() {}
    Poly(int n) : coeff(n) {}
    Poly(const vector<T>& a) : coeff(a) {}
    Poly(const initializer_list<T>& a) : coeff(a) {}
```

```
    static constexpr int mod() { return (int) T::mod(); }
    inline int size() const { return SZ(coeff); }
    void resize(int n) { coeff.resize(n); }
    T at(int idx) const {
        if(idx < 0 || idx >= size()) return 0;
        return coeff[idx];
    }
    T& operator[](int idx) { return coeff[idx]; }
    Poly mulxk(int k) const {
        auto b = coeff;
        b.insert(b.begin(), k, T(0));
        return Poly(b);
    }
    Poly modxk(int k) const {
        k = min(k, size());
        return Poly(vector<T>(coeff.begin(), coeff.begin() + k));
    }
    Poly divxk(int k) const {
        if(size() <= k) return Poly<T>();
        return Poly(vector<T>(coeff.begin() + k, coeff.end()));
    }
    friend Poly operator+(const Poly& a, const Poly& b) {
        vector<T> c(max(SZ(a), SZ(b)));
        for(int i = 0; i < SZ(c); ++i) c[i] = a.at(i) + b.at(i);
        return Poly(c);
    }
    friend Poly operator-(const Poly& a, const Poly& b) {
        vector<T> c(max(SZ(a), SZ(b)));
        for(int i = 0; i < SZ(c); ++i) res[i] = a.at(i) - b.at(i);
        return Poly(c);
    }
    static void ensure_base(int n) {
        if(SZ(__bit_reorder) != n) {
            int k = __builtin_ctz(n) - 1;
            __bit_reorder.resize(n);
            for(int i = 0; i < n; ++i) {
                __bit_reorder[i] = __bit_reorder[i >> 1] >> 1 | (i & 1) << k;
            }
        }
        if(SZ(roots) < n) {
            int k = __builtin_ctz(SZ(roots));
            roots.resize(n);
            while((1 << k) < n) {
                T e = pow_mod_constexpr(R, (T::mod() - 1) >> (k + 1), T::mod());
                for(int i = 1 << (k - 1); i < (1 << k); ++i) {
                    roots[2 * i] = roots[i];
                    roots[2 * i + 1] = roots[i] * e;
                }
                k += 1;
            }
        }
    }
    static void dft(vector<T>& a) {
        const int n = SZ(a);
        assert((n & -n) == n);
        ensure_base(n);
        for(int i = 0; i < n; ++i) {
            if(__bit_reorder[i] < i) swap(a[i], a[__bit_reorder[i]]);
        }
        for(int k = 1; k < n; k *= 2) {
            for(int i = 0; i < n; i += 2 * k) {
                for(int j = 0; j < k; ++j) {
                    T u = a[i + j];
                    T v = a[i + j + k] * roots[k + j];
                    a[i + j] = u + v;
                    a[i + j + k] = u - v;
                }
            }
        }
    }
    static void idft(vector<T>& a) {
        const int n = SZ(a);
        reverse(1 + ALL(a));
        dft(a);
        T inv = (1 - T::mod()) / n;
        for(int i = 0; i < n; ++i) a[i] *= inv;
    }
    friend Poly operator*(Poly a, Poly b) {
        if(SZ(a) == 0 || SZ(b) == 0) return Poly();
        if(min(SZ(a), SZ(b)) < 250) {
            vector<T> c(SZ(a) + SZ(b) - 1);
            for(int i = 0; i < SZ(a); ++i) {
                for(int j = 0; j < SZ(b); ++j) {
                    c[i + j] += a[i] * b[j];
                }
            }
            return Poly(c);
        }
        int tot = SZ(a) + SZ(b) - 1;
        int sz = 1;
        while(sz < tot) sz <<= 1;
        a.coeff.resize(sz); b.coeff.resize(sz);
        dft(a.coeff); dft(b.coeff);
        for(int i = 0; i < sz; ++i) a.coeff[i] = a[i] * b[i];
        idft(a.coeff);
        a.resize(tot);
        return a;
    }
    friend Poly operator*(T a, Poly b) {
        for(int i = 0; i < SZ(b); ++i) b[i] *= a;
        return b;
    }
    friend Poly operator*(Poly a, T b) {
        for(int i = 0; i < SZ(a); ++i) a[i] *= b;
        return a;
    }
    Poly& operator+=(Poly b) { return *this = *this + b; }
    Poly& operator-=(Poly b) { return *this = *this - b; }
    Poly& operator*=(Poly b) { return *this = *this * b; }
    Poly deriv() const {
        if(coeff.empty()) return Poly<T>();
        vector<T> res(size() - 1);
```

```
        for(int i = 0; i < size() - 1; ++i) res[i] = (i + 1) * coeff[i + 1];
        return Poly(res);
    }
    Poly integr() const {
        vector<T> res(size() + 1);
        for(int i = 0; i < size(); ++i) res[i + 1] = coeff[i] / T(i + 1);
        return Poly(res);
    }
    Poly inv(int m) const {
        Poly x{T(1) / coeff[0]};
        int k = 1;
        while(k < m) {
            k *= 2;
            x = (x * (Poly{T(2)} - modxk(k) * x)).modxk(k);
        }
        return x.modxk(m);
    }
    Poly log(int m) const { return (deriv() * inv(m)).integr().modxk(m); }
    Poly exp(int m) const {
        Poly x{T(1)};
        int k = 1;
        while(k < m) {
            k *= 2;
            x = (x * (Poly{T(1)} - x.log(k) + modxk(k))).modxk(k);
        }
        return x.modxk(m);
    }
    Poly pow(int k, int m) const {
        if(k == 0) {
            vector<T> a(m);
            a[0] = 1;
            return Poly(a);
        }
        int i = 0;
        while(i < size() && coeff[i]() == 0) i++;
        if(i == size() || 1LL * i * k >= m) return Poly(vector<T>(m));
        T v = coeff[i];
        auto f = divxk(i) * (1 / v);
        return (f.log(m - i * k) * T(k)).exp(m - i * k).mulxk(i * k) * power(v, k);
    }
    Poly sqrt(int m) const {
        Poly<T> x{1};
        int k = 1;
        while(k < m) {
            k *= 2;
            x = (x + (modxk(k) * x.inv(k)).modxk(k)) * T((mod() + 1) / 2);
        }
        return x.modxk(m);
    }
    Poly mulT(Poly b) const {
        if(b.size() == 0) return Poly<T>();
        int n = SZ(b);
        reverse(ALL(b.coeff));
        return ((*this) * b).divxk(n - 1);
    }
    vector<T> eval(vector<T> x) const {
        if(size() == 0) return vector<T>(SZ(x), 0);
        const int n = max(SZ(x), size());
        vector<Poly<T>> q(4 * n);
        vector<T> ans(x.size());
        x.resize(n);
        function<void(int, int, int)> build = [&](int p, int l, int r) {
            if(r - l == 1) q[p] = Poly{1, -x[l]};
            else {
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m, r);
                q[p] = q[2 * p] * q[2 * p + 1];
            }
        };
        build(1, 0, n);
        function<void(int, int, int, const Poly&)> work = [&](int p, int l, int r, const Poly& num) {
            if(r - l == 1) {
                if(l < SZ(ans)) ans[l] = num[0];
            } else {
                int m = (l + r) / 2;
                work(2 * p, l, m, num.mulT(q[2 * p + 1]).modxk(m - l));
                work(2 * p + 1, m, r, num.mulT(q[2 * p]).modxk(r - m));
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
    }
private:
    vector<T> coeff;
    static vector<T> roots;
};
template<class T> vector<T> Poly<T>::roots{0, 1};
```

5.18 XorBasis.h

```
template<int LOG> struct XorBasis {
    bool zero = false;
    int cnt = 0;
    ll p[LOG] = {};
    vector<ll> d;
    void insert(ll x) {
        for(int i = LOG - 1; i >= 0; --i) {
            if(x >> i & 1) {
                if(!p[i]) {
                    p[i] = x;
                    cnt += 1;
                    return;
                } else x ^= p[i];
            }
        }
        zero = true;
    }
    ll get_max() {
        ll ans = 0;
        for(int i = LOG - 1; i >= 0; --i) {
            if((ans ^ p[i]) > ans) ans ^= p[i];
        }
        return ans;
    }
```

```
    ll get_min() {
        if(zero) return 0;
        for(int i = 0; i < LOG; ++i) {
            if(p[i]) return p[i];
        }
    }
    bool include(ll x) {
        for(int i = LOG - 1; i >= 0; --i) {
            if(x >> i & 1) x ^= p[i];
        }
        return x == 0;
    }
    void update() {
        d.clear();
        for(int j = 0; j < LOG; ++j) {
            for(int i = j - 1; i >= 0; --i) {
                if(p[j] >> i & 1) p[j] ^= p[i];
            }
        }
        for(int i = 0; i < LOG; ++i) {
            if(p[i]) d.PB(p[i]);
        }
    }
    ll get_kth(ll k) {
        if(k == 1 && zero) return 0;
        if(zero) k -= 1;
        if(k >= (1LL << cnt)) return -1;
        update();
        ll ans = 0;
        for(int i = 0; i < SZ(d); ++i) {
            if(k >> i & 1) ans ^= d[i];
        }
        return ans;
    }
};
```

5.19 Theorem

- Cramer's rule

  $ax+by = ecx+dy = f \Rightarrow x = ed-bfad-bcy = af-ecad-bc$

- Kirchhoff's Theorem Denote L be a $n \times n$ matrix as the Laplacian matrix of graph G, where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i, j) in G.
  - The number of undirected spanning in G is $\det(\tilde{L}_{11})$.
  - The number of directed spanning tree rooted at r in G is $\det(\tilde{L}_{rr})$.

- Tutte's Matrix Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniformly at random) if $i < j$ and $(i, j) \in E$, otherwise $d_{ij} = -d_{ji}$. $\frac{rank(D)}{2}$ is the maximum matching on G.

- Cayley's Formula
  - Given a degree sequence $d_1, d_2, \ldots, d_n$ for each *labeled* vertices, there are $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}$ spanning trees.
  - Let $T_{n,k}$ be the number of *labeled* forests on n vertices with k components, such that vertex $1, 2, \ldots, k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.

- Erdős–Gallai theorem A sequence of nonnegative integers $d_1 \geq \cdots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + \cdots + d_n$ is even and $\sum_{i-1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$ holds for every $1 \leq k \leq n$.

- Gale–Ryser theorem A pair of sequences of nonnegative integers $a_1 \geq \cdots \geq a_n$ and $b_1, \ldots, b_n$ is bigraphic if and only if $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ and $\sum_{i=1}^{k} a_i \leq \sum_{i=1}^{n} \min(b_i, k)$ holds for every $1 \leq k \leq n$.

- Fulkerson–Chen–Anstee theorem A sequence $(a_1, b_1), \ldots, (a_n, b_n)$ of nonnegative integer pairs with $a_1 \geq \cdots \geq a_n$ is digraphic if and only if $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ and $\sum_{i=1}^{k} a_i \leq \sum_{i=1}^{k} \min(b_i, k-1) + \sum_{i=k+1}^{n} \min(b_i, k)$ holds for every $1 \leq k \leq n$.

- Möbius inversion formula
  - $f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu(d) f(\frac{n}{d})$
  - $f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu(\frac{d}{n}) f(d)$

- Spherical cap
  - A portion of a sphere cut off by a plane.
  - r: sphere radius, a: radius of the base of the cap, h: height of the cap, θ: $\arcsin(a/r)$.
  - Volume $= \pi h^2(3r-h)/3 = \pi h(3a^2+h^2)/6 = \pi r^3(2+\cos\theta)(1-\cos\theta)^2/3$.
  - Area $= 2\pi rh = \pi(a^2+h^2) = 2\pi r^2(1-\cos\theta)$.

5.20 Numbers

- Bernoulli numbers $B_0 - 1, B_1^{\pm} = \pm\frac{1}{2}, B_2 = \frac{1}{6}, B_3 = 0$ $\sum_{j=0}^{m} m+1 j B_j = 0$, EGF is $B(x) = \frac{x}{e^x-1} = \sum_{n=0}^{\infty} B_n \frac{x^n}{n!}$. $S_m(n) = \sum_{k=1}^{n} k^m = \frac{1}{m+1} \sum_{k=0}^{m} m+1 k B_k^{+} n^{m+1-k}$

- Stirling numbers of the second kind Partitions of n distinct elements into exactly k groups. $S(n,k) = S(n-1,k-1) + kS(n-1,k), S(n,1) = S(n,n) = 1$ $S(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^{k-i}\binom{k}{i} i^n \; x^n = \sum_{i=0}^{n} S(n,i)(x)_i$

- Pentagonal number theorem $\prod_{n=1}^{\infty}(1-x^n) = 1 + \sum_{k=1}^{\infty}(-1)^k \left(x^{k(3k+1)/2} + x^{k(3k-1)/2}\right)$

- Catalan numbers $C_n^{(k)} = \frac{1}{(k-1)n+1} knn$ $C^{(k)}(x) = 1 + x[C^{(k)}(x)]^k$

- Eulerian numbers Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$. $E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$ $E(n,0) = E(n,n-1) = 1$ $E(n,k) = \sum_{j=0}^{k}(-1)^j n+1 j(k+1-j)^n$

5.21 GeneratingFunctions

- Ordinary Generating Function $A(x) = \sum_{i \geq 0} a_i x^i$
  - $A(rx) \Rightarrow r^n a_n$
  - $A(x) + B(x) \Rightarrow a_n + b_n$
  - $A(x)B(x) \Rightarrow \sum_{i=0}^{n} a_i b_{n-i}$
  - $A(x)^k \Rightarrow \sum_{i_1+i_2+\cdots+i_k=n} a_{i_1} a_{i_2} \ldots a_{i_k}$
  - $xA(x)' \Rightarrow na_n$
  - $\frac{A(x)}{1-x} \Rightarrow \sum_{i=0}^{n} a_i$
- Exponential Generating Function $A(x) = \sum_{i \geq 0} \frac{a_i}{i!} x_i$
  - $A(x) + B(x) \Rightarrow a_n + b_n$
  - $A^{(k)}(x) \Rightarrow a_{n+k}$
  - $A(x)B(x) \Rightarrow \sum_{i=0} nia_i b_{n-i}$
  - $A(x)^k \Rightarrow \sum_{i_1+i_2+\cdots+i_k=n} ni_1, i_2, \ldots, i_k a_{i_1} a_{i_2} \ldots a_{i_k}$
  - $xA(x) \Rightarrow na_n$
- Special Generating Function
  - $(1+x)^n = \sum_{i \geq 0} nix^i$
  - $\frac{1}{(1-x)^n} = \sum_{i \geq 0} in-1x^i$

6 Geometry

6.1 Point.h

```cpp
template<class T> struct Point {
    T x, y;
    Point() : x(0), y(0) {}
    Point(const T& a, const T& b) : x(a), y(b) {}
    Point(const pair<T, T>& p) : x(p.F), y(p.S) {}
    inline Point& operator+=(const Point& rhs) {
        x += rhs.x, y += rhs.y; return *this;
    }
    inline Point& operator-=(const Point& rhs) {
        x -= rhs.x, y -= rhs.y; return *this;
    }
    inline Point& operator*=(const T& rhs) {
        x *= rhs, y *= rhs; return *this;
    }
    inline Point& operator/=(const T& rhs) {
        x /= rhs, y /= rhs; return *this;
    }
    template<class U>
    inline Point& operator+=(const Point<U>& rhs) {
        return *this += Point<T>(rhs);
    }
    template<class U>
    inline Point& operator-=(const Point<U>& rhs) {
        return *this -= Point<T>(rhs);
    }
    inline Point operator+() const { return *this; }
    inline Point operator-() const {
        return Point(-x, -y);
    }
    inline Point operator+(const Point& rhs) {
        return Point(*this) += rhs;
    }
    inline Point operator-(const Point& rhs) {
        return Point(*this) -= rhs;
    }
    inline Point operator*(const T& rhs) {
        return Point(*this) *= rhs;
    }
    inline Point operator/(const T& rhs) {
        return Point(*this) /= rhs;
    }
    inline bool operator==(const Point& rhs) {
        return x == rhs.x && y == rhs.y;
    }
    inline bool operator!=(const Point& rhs) {
        return !(*this == rhs);
    }
    inline T dist2() const { return x * x + y * y; }
    inline ld dist() const { return sqrt(dist2()); }
    inline Point unit() const { return *this / this->dist(); }
    inline ld angle() const { return atan2(y, x); }
    inline friend T dot(const Point& lhs, const Point& rhs) {
        return lhs.x * rhs.x + lhs.y * rhs.y;
    }
    inline friend T cross(const Point& lhs, const Point& rhs) {
        return lhs.x * rhs.y - lhs.y * rhs.x;
    }
    inline friend Point dot_cross(const Point& lhs, const Point& rhs) {
        return Point(dot(lhs, rhs), cross(lhs, rhs));
    }
};
```

6.2 LineSeg.h

```cpp
int sign(const double& a) { return fabs(a) < EPS ? 0 : a > 0 ? 1 : -1; }
template<class T>
int ori(const Point<T>& a, const Point<T>& b, const Point<T>& c) {
    return sign(cross(b - a, c - a));
}
template<class T>
bool collinearity(const Point<T>& a, const Point<T>& b, const Point<T>& c) {
    return sign(cross(a - c, b - c)) == 0;
}
template<class T>
bool btw(const Point<T>& a, const Point<T>& b, const Point<T>& c) {
    if(!collinearity(a, b, c)) return 0;
    return sign(dot(a - c, b - c)) <= 0;
}
template<class T>
bool seg_intersect(const Point<T>& a, const Point<T>& b, const Point<T>& c, const Point<T>& d) {
    int abc = ori(a, b, c), abd = ori(a, b, d);
    int cda = ori(c, d, a), cdb = ori(c, d, b);
    if(abc == 0 && abd == 0) return btw(a, b, c) || btw(a, b, d) || btw(c, d, a) || btw(c, d, b);
    return abc * abd <= 0 && cda * cdb <= 0;
}
template<class T>
```

```
Point<T> intersect(const Point<T>& a, const Point<T>& b, const Point<T>& c, const Point<T>& d) {
    T a123 = cross(b - a, c - a);
    T a124 = cross(b - a, d - a);
    return (d * a123 - c * a124) / (a123 - a124);
}
```

6.3 ConvexHull.h

```
// @return the points of the convex hull in clock-wise order
template<class T>
vector<Point<T>> ConvexHull(vector<Point<T>> points) {
    const int n = SZ(points);
    sort(ALL(points), [](const Point<T>& a, const Point<T>& b) {
        if(a.x == b.x) return a.y < b.y;
        return a.x < b.x;
    });
    auto build = [&]() {
        vector<Point<T>> upper;
        upper.PB(points[0]);
        upper.PB(points[1]);
        for(int i = 2; i < n; ++i) {
            while(SZ(upper) >= 2) {
                if(cross(upper.end()[-1] - upper.end()[-2], points[i] - upper.end()[-1]) > 0)
                    upper.PPB();
                else break;
            }
            upper.PB(points[i]);
        }
        return upper;
    };
    vector<Point<T>> upper = build();
    reverse(ALL(points));
    vector<Point<T>> lower = build();
    lower.PPB();
    upper.insert(upper.end(), 1 + ALL(lower));
    return upper;
}
```

6.4 HalfPlaneIntersection.h

```
struct Halfplane {
    Point p, pq;
    ld angle;
    Halfplane() {}
    Halfplane(const Point& a, const Point& b) : p(a), pq(b - a) {
        angle = atan2l(pq.y, pq.x);
    }
    bool out(const Point& r) { return cross(pq, r - p) < -EPS; }
    bool operator<(const Halfplane& e) const { return angle < e.angle; }
    friend Point inter(const Halfplane& s, const Halfplane& t) {
        ld alpha = cross((t.p - s.p), t.pq) / cross(s.pq, t.pq);
        return s.p + (s.pq * alpha);
    }
};
vector<Point> hp_intersect(vector<Halfplane>& H) {
    Point box[4] = {
        Point(inf, inf), Point(-inf, inf),
        Point(-inf, -inf), Point(inf, -inf)
    };
    for(int i = 0; i < 4; ++i) H.EB(box[i], box[(i + 1) % 4]);
    sort(H.begin(), H.end());
    deque<Halfplane> dq;
    int len = 0;
    for(int i = 0; i < SZ(H); i++) {
        while(len > 1 && H[i].out(inter(dq[len - 1], dq[len - 2]))) {
            dq.PPB(); --len;
        }
        while(len > 1 && H[i].out(inter(dq[0], dq[1]))) {
            dq.pop_front(); --len;
        }
        if(len > 0 && fabsl(cross(H[i].pq, dq[len-1].pq)) < EPS) {
            if(dot(H[i].pq, dq[len - 1].pq) < 0.0) return {};
            if(H[i].out(dq[len - 1].p)) {
                dq.PPB(); --len;
            } else continue;
        }
        dq.PB(H[i]);
        ++len;
    }
    while(len > 2 && dq[0].out(inter(dq[len - 1], dq[len - 2]))) {
        dq.PPB(); --len;
    }
    while(len > 2 && dq[len - 1].out(inter(dq[0], dq[1]))) {
        dq.pop_front(); --len;
    }
    if(len < 3) return {};
    vector<Point> ret(len);
    for(int i = 0; i + 1 < len; ++i) ret[i] = inter(dq[i], dq[i+1]);
    ret.back() = inter(dq[len-1], dq[0]);
    return ret;
}
```

7 Misc

7.1 TenarySearch.h

```
// return the maximum of f(x) in [l, r]
double ternary_search(double l, double r) {
    while(r - l > EPS) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1), f2 = f(m2);
        if(f1 < f2) l = m1;
```

```
        else r = m2;
    }
    return f(l);
}
// return the maximum of f(x) in (l,r]
int ternary_search(int l, int r) {
    while(r - l > 1) {
        int mid = (l + r) / 2;
        if(f(m) > f(m + 1)) r = m;
        else l = m;
    }
    return r;
}
```

7.2 Aliens.h

```
// find minimum
int Aliens(int l, int r, int k, const
↪ function<pii(int)>& f) {
    while(l < r) {
        int m = l + (r - l) / 2;
        auto [score, op] = f(m);
        if(op == k) return score - m * k;
        if(op < k) r = m;
        else l = m + 1;
    }
    return f(l).first - l * k;
}
```

7.3 Debug.h

```
#ifdef LOCAL
    #define eprintf(...) { fprintf(stderr,
↪ __VA_ARGS__); fflush(stderr); }
#else
    #define eprintf(...) 42
#endif
```

7.4 Timer.h

```
const clock_t startTime = clock();
inline double getCurrentTime() {
    return (double) (clock() - startTime) /
↪ CLOCKS_PER_SEC;
}
```

7.5 ReadChar.h

```
inline char gc() {
    static const int SZ = 1 << 20;
    static int cnt = 1 << 21;
    static char buf[SZ];
    static char *ptr = buf, *end = buf;
    if(ptr == end) {
        if(cnt < SZ) return EOF;
        cnt = fread(buf, 1, SZ, stdin);
        ptr = buf;
        end = buf + cnt;
    }
    return *(ptr++);
}
```