Team Girigiri Library

xuzijian629, knshnb, ogingin

Compiled on July 10, 2019

Contents

1 Segment Tree

1.1 Base

```
// T0: 元の配列のモノイド
// T1: T0にする作用素モノイド
template<class T0, class T1>
class SegmentTree {
    // k番目のノードにのlazyを搬
    void eval(int k, int len) {
        // 定倍高速化
        if (lazy[k] == u1) return;
        // len個分のlazy[k]を評
        node[k] = g(node[k], p(lazy[k], len));
        if (k < N - 1) {
            // 最下段でなければ下のlazyに搬
            lazy[2 * k + 1] = f1(lazy[2 * k + 1], lazy[k]);
            lazy[2 * k + 2] = f1(lazy[2 * k + 2], lazy[k]);
        }
        lazy[k] = u1;
    }

    // k番目のノード[l, r)について、[a, b)の範にxを作用
    void update(int a, int b, T1 x, int k, int l, int r) {
        eval(k, r - l);
        if (b <= l || r <= a) return;
        if (a <= l && r <= b) {
            lazy[k] = f1(lazy[k], x);
            eval(k, r - l);
        } else {
            update(a, b, x, 2 * k + 1, l, (l + r) / 2);
            update(a, b, x, 2 * k + 2, (l + r) / 2, r);
            node[k] = f0(node[2 * k + 1], node[2 * k + 2]);
        }
    }

    // k番目のノード[l, r)について、[a, b)のクエリを求める
    T0 query(int a, int b, int k, int l, int r) {
        if (r <= a || b <= l) return u0;
        eval(k, r - l);
        if (a <= l && r <= b) return node[k];
        T0 vl = query(a, b, 2 * k + 1, l, (l + r) / 2);
        T0 vr = query(a, b, 2 * k + 2, (l + r) / 2, r);
        return f0(vl, vr);
    }

public:
    int sz; // 元の配列のサイズ
    int N;
    vector<T0> node;
    vector<T1> lazy;
    // T0上の演算、位元
    using F0 = function<T0(T0, T0)>;
    F0 f0;
    T0 u0;
    // T1上の演算、位元
    using F1 = function<T1(T1, T1)>;
    F1 f1;
```

```
    T1 u1;
    // 作用
    using G = function<T0(T0, T1)>;
    G g;
    // 多のt1(T1)にするf1の合成
    using P = function<T1(T1, int)>;
    P p;

    SegmentTree(const vector<T0> &a, F0 f0, T0 u0, F1 f1, T1 u1, G g, P p)
            : sz(a.size()), f0(f0), u0(u0), f1(f1), u1(u1), g(g), p(p) {
        for (N = 1; N < sz; N *= 2);
        node.resize(2 * N - 1);
        lazy.resize(2 * N - 1, u1);
        for (int i = 0; i < sz; i++) node[N - 1 + i] = a[i];
        for (int i = N - 2; i >= 0; i--) node[i] = f0(node[2 * i + 1], node[2 * i + 2]);
    }

    // [a, b)にxを作用
    void update(int a, int b, T1 x) {
        assert(0 <= a && a < b && b <= sz);
        update(a, b, x, 0, 0, N);
    }

    void update(int a, T1 x) {
        update(a, a + 1, x);
    }

    // [a, b)
    T0 query(int a, int b) {
        return query(a, b, 0, 0, N);
    }

    T0 query(int a) {
        return query(a, a + 1);
    }
};
```

1.2 Min & Update

```
// Min & Update
SegmentTree<int, int> seg(
        vector<int>(n, (1LL << 31) - 1),
        [](int x, int y) { return min(x, y); }, 1e18,
        [](int x, int y) { return y == 1e18 ? x : y; }, 1e18,
        [](int x, int y) { return y == 1e18 ? x : y; },
        [](int y, int len) { return y; }
);
```

1.3 Sum & Add

```
// Sum & Add
SegmentTree<int, int> seg(
        vector<int>(n, 0),
        plus<>(), 0,
        plus<>(), 0,
        plus<>(),
        multiplies<>()
);
```

1.4 Min & Add

```
// Min & Add
SegmentTree<int, int> seg(
        vector<int>(n, 0),
        [](int x, int y) { return min(x, y); }, 1e18,
        plus<>(), 0,
        plus<>(),
        [](int y, int len) { return y; }
);
```

1.5 Sum & Update

```
// Sum & Update
constexpr int u1 = 1e18;
SegmentTree<int, int> seg(
        vector<int>(n),
        plus<>(), 0,
        [](int x, int y) { return y == u1 ? x : y; }, u1,
        [](int x, int y) { return y == u1 ? x : y; },
        [](int y, int len) { return y == u1 ? u1 : y * len; }
);
```

1.6 Sum & Affine

```
// Sum & Affine
using T = int;
using Affine = pair<T, T>;
SegmentTree<Affine, Affine> seg(
        vector<Affine>(n),
        [](Affine x, Affine y) { return Affine(x.first + y.first, x.second + y.second); },
        Affine(0, 0),
        [](Affine x, Affine y) { return Affine(x.first * y.first, x.second * y.first +
        y.second); }, Affine(1, 0),
        [](Affine x, Affine y) { return Affine(x.first * y.first, x.second * y.first +
        y.second); },
        [](Affine y, int len) { return Affine(y.first, y.second * len); }
    );
//    seg.update(i, j, {a, b}); // [i, j)にax + bを作用
//    seg.update(i, j, {0, a}); // update
//    seg.update(i, j, {1, a}); // 加算
//    seg.update(i, j, {a, 0}); // 倍
```

1.7 Or & Update

```
// range or & range update
constexpr int u1 = 1ll << 30;
SegmentTree<int, int> seg(vector<int>(n), [](int x, int y) { return x | y; }, 0,
                                          [](int x, int y) { return y == u1 ? x : y; }, u1,
                                          [](int x, int y) { return y == u1 ? x : y; },
                                          [](int y, int len) { return y; });
```

1.8 Persistent Segment Tree

```
// 一点更新、問みみ
class PersistentSegTree {
    struct Node {
        int data;
        Node *l, *r;

        Node(const int &data) : data(data) {}
    };

    int sz;
```

```
    function<int(int, int)> f;
    int unit;

    PersistentSegTree(function<int(int, int)> f, int unit) : f(move(f)), unit(unit) {}

    Node *build(vector<int> &v) {
        sz = int(v.size());
        return build(0, int(v.size()), v);
    }

    Node *merge(Node *l, Node *r) {
        auto t = new Node(f(l->data, r->data));
        t->l = l;
        t->r = r;
        return t;
    }

    Node *build(int l, int r, vector<int> &v) {
        if (l + 1 >= r) return new Node(v[l]);
        return merge(build(l, (l + r) >> 1, v), build((l + r) >> 1, r, v));
    }

    Node *update(int a, int &x, Node *k, int l, int r) {
        if (r <= a || a + 1 <= l) {
            return k;
        } else if (a <= l && r <= a + 1) {
            return new Node(x);
        } else {
            return merge(update(a, x, k->l, l, (l + r) >> 1), update(a, x, k->r, (l + r) >>
            1, r));
        }
    }

    Node *update(Node *t, int k, int &x) {
        return update(k, x, t, 0, sz);
    }

    int query(int a, int b, Node *k, int l, int r) {
        if (r <= a || b <= l || !k) {
            return unit;
        } else if (a <= l && r <= b) {
            return k->data;
        } else {
            return f(query(a, b, k->l, l, (l + r) >> 1), query(a, b, k->r, (l + r) >> 1,
            r));
        }
    }

    int query(Node *t, int a, int b) {
        return query(a, b, t, 0, sz);
    }
};
```

2 Graph

2.1 Longest Path of DAG

```
int longest_path(const vector<vector<int>> &adj) {
    int n = adj.size();
    vector<vector<int>> inv(n);
    for (int a = 0; a < n; a++) {
        for (auto b : adj[a]) {
            inv[b].push_back(a);
        }
    }
    vector<int> len(n, -1);
    function<void(int)> set_len = [&](int v) {
        if (len[v] != -1) return;
        if (inv[v].empty()) {
            len[v] = 0;
            return;
        }
        int nax = 0;
        for (int u : inv[v]) {
            set_len(u);
            nax = max(nax, len[u]);
        }
        len[v] = nax + 1;
    };

    for (int i = 0; i < n; i++) {
        set_len(i);
    }

    return *max_element(len.begin(), len.end());
}
```

2.2 Topological Sort & Cycle Detection

```
class DAG {
    int n;
    vector<vector<int>> adj;
    vector<int> visited;

    void dfs(int v) {
        if (visited[v] == 2) {
            is_acyclic = false;
            return;
        } else if (!visited[v]) {
            visited[v] = 2;
            for (int s : adj[v]) {
                dfs(s);
            }
            visited[v] = 1;
            sorted.push_back(v);
        }
    }

public:
    vector<int> sorted;
    DAG(int n) : n(n), adj(n), visited(n) {}
    bool is_acyclic = true;
```

```
    void add_arc(int a, int b) {
        assert(0 <= a && a < n && 0 <= b && b < n);
        adj[a].push_back(b);
    }

    void tsort() {
        for (int i = 0; i < n; i++) {
            if (!visited[i]) dfs(i);
        }
        reverse(sorted.begin(), sorted.end());
    }
};
```

2.3 Strongly Connected Components Decomposition

```
class SCCDecomp {
    int n;
    vector<vector<int>> adj;

public:
    vector<vector<int>> scc;

    SCCDecomp(int n) : n(n), adj(n) {}

    void addArc(int a, int b) {
        adj[a].push_back(b);
    }

    int run() {
        vector<int> num(n), low(n);
        stack<int> S;
        vector<int> inS(n);
        int t = 0;
        function<void(int)> visit = [&](int v) {
            low[v] = num[v] = ++t;
            S.push(v);
            inS[v] = 1;
            for (int s: adj[v]) {
                if (!num[s]) {
                    vector<int> sit(s);
                    low[v] = min(low[v], low[s]);
                } else if (inS[s]) {
                    low[v] = min(low[v], num[s]);
                }
            }
            if (low[v] == num[v]) {
                scc.emplace_back();
                while (1) {
                    int w = S.top();
                    S.pop();
                    inS[w] = false;
                    scc.back().push_back(w);
                    if (v == w) {
                        break;
                    }
                }
            }
        };

        for (int i = 0; i < n; i++) {
            if (num[i] == 0) {
                vector<int> sit(i);
            }
        }

        return scc.size();
    }
};
```

2.4 Bellman Form

```
struct edge {
    int a, b;
    double w;
};

class BellmanFord {
    int n;
    vector<edge> edges;
    vector<double> d;

public:
    BellmanFord(int n) : n(n), edges(n), d(n, numeric_limits<double>::infinity()) {}

    void add_edge(int a, int b, double w) {
        assert(0 <= a && a < n && 0 <= b && b < n);
        edges.push_back({a, b, w});
        edges.push_back({b, a, w});
    }

    void add_arc(int a, int b, double w) {
        assert(0 <= a && a < n && 0 <= b && b < n);
        edges.push_back({a, b, w});
    }

    // return true if has negative loop
    bool run(int a) {
        d[a] = 0;
        for (int i = 0; i < n; i++) {
            bool change = false;
            for (edge e: edges) {
                if (d[e.a] != numeric_limits<double>::infinity()) {
                    if (d[e.b] > d[e.a] + e.w) {
                        d[e.b] = d[e.a] + e.w;
                        change = true;
                    }
                }
            }
            if (!change) {
                return false;
            }
        }
        return true;
    }
```

```
    double dist(int a) {
        return d[a];
    }
};
```

2.5 Erdos-Gallai Graph Existence

```
// ask knshnb for template!
// Erdos-Gallai theorem: (O(n))
// https://en.wikipedia.org/wiki/Erdős–Gallai_theorem
bool is_graphic(const VI& d) {
  int n = d.size();
  if (accumulate(ALL(d), 0LL) % 2) return false;
  VI acc(n + 1);
  REP (i, n) {
    acc[i + 1] = acc[i] + d[i];
  }
  int l = n - 1; // d[l] >= i + 1をたす最大のl
  REP (i, n) {
    int lhs = acc[i + 1];
    while (l >= i + 1 && d[l] < i + 1) l--;
    // [i + 1, l]: i + 1, [l + 1, n - 1]: acc
    int rhs = i * (i + 1) + (i + 1) * (l - i) + (acc[n] - acc[l + 1]);
    if (lhs > rhs) return false;
  }
  return true;
}
```

3 Tree

3.1 Heavy Light Decomposition

```
struct HLD {
    int n;
    vector<vector<int>> adj;
    vector<int> sz, in, out, head, rev, par, depth;

    HLD(int n) : n(n), adj(n), sz(n), in(n), out(n), head(n), rev(n), par(n), depth(n) {}

    void add_edge(int a, int b) {
        assert(0 <= a && a < n && 0 <= b && b < n);
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

    void dfs_sz(int v, int p, int d) {
        par[v] = p;
        sz[v] = 1;
        depth[v] = d;
        if (!adj[v].empty() && adj[v][0] == p) swap(adj[v][0], adj[v].back());
        for (int &s : adj[v]) {
            if (s == p) continue;
            dfs_sz(s, v, d + 1);
            sz[v] += sz[s];
            if (sz[adj[v][0]] < sz[s]) swap(adj[v][0], s);
        }
    }

    void dfs_hld(int v, int p, int &times) {
        in[v] = times++;
        rev[in[v]] = v;
        for (int &s : adj[v]) {
            if (s == p) continue;
            head[s] = adj[v][0] == s ? head[v] : s;
            dfs_hld(s, v, times);
        }
        out[v] = times;
    }

    void build() {
        dfs_sz(0, -1, 0);
        int t = 0;
        dfs_hld(0, -1, t);
    }

    int la(int v, int k) {
        while (1) {
            int u = head[v];
            if (in[v] - k >= in[u]) return rev[in[v] - k];
            k -= in[v] - in[u] + 1;
            v = par[u];
        }
    }

    int lca(int u, int v) {
        for (;; v = par[head[v]]) {
            if (in[u] > in[v]) swap(u, v);
            if (head[u] == head[v]) return u;
        }
    }

    template<typename Q, typename F, typename T>
    // qは閉間にさせること!!!!!!!!
    T query(int u, int v, const Q &q, const F &f, const T &e, bool edge = false) {
        T l = e, r = e;
        for (;; v = par[head[v]]) {
            if (in[u] > in[v]) swap(u, v), swap(l, r);
            if (head[u] == head[v]) break;
            l = f(q(in[head[v]], in[v]), l);
        }
        return f(f(q(in[u] + edge, in[v]), l), r);
    }

    template<typename Q>
    // qは閉間にさせること!!!!!!!!
    void update(int u, int v, const Q &q, bool edge = false) {
        for (;; v = par[head[v]]) {
            if (in[u] > in[v]) swap(u, v);
            if (head[u] == head[v]) break;
            q(in[head[v]], in[v]);
        }
        q(in[u] + edge, in[v]);
    }
};
```

3.2 Euler Tour

See Heavy Light Decomposition

3.3 Centroid Decomposition

```
class CentroidDecomposition {
    int n;
    vector<vector<int>> adj;
    vector<int> sub;
    vector<bool> visited;
    int build_dfs(int v, int p) {
        sub[v] = 1;
        for (int s : adj[v]) {
            if (s != p && !visited[s]) {
                sub[v] += build_dfs(s, v);
            }
        }
        return sub[v];
    }
    int search_centroid(int v, int p, int mid) {
        for (int s : adj[v]) {
            if (s != p && !visited[s]) {
                if (sub[s] > mid) return search_centroid(s, v, mid);
            }
        }
        return v;
    }
    void path_dfs(int v, int p, int centroid) {
        path[v].push_back(centroid);
        for (int s : adj[v]) {
            if (s != p && !visited[s]) {
                path_dfs(s, v, centroid);
            }
        }
    }
    int build(vector<vector<int>> &t, int v) {
        int centroid = search_centroid(v, -1, build_dfs(v, -1) / 2);
        visited[centroid] = true;
        path_dfs(centroid, -1, centroid);
        for (int s : adj[centroid]) {
            if (!visited[s]) {
                t[centroid].emplace_back(build(t, s));
            }
        }
        visited[centroid] = false;
        return centroid;
    }

public:
    // 分解した木において各頂点から重心までのpath(それぞれO(logN)の長さになる)
    vector<vector<int>> path;
    vector<vector<int>> decomp;

    CentroidDecomposition(int n) : n(n), adj(n), sub(n), visited(n), path(n), decomp(n) {}
    void add_edge(int a, int b) {
        assert(0 <= a && a < n && 0 <= b && b < n);
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    int build() {
        int root = build(decomp, 0);
        for (int i = 0; i < n; i++) {
            reverse(path[i].begin(), path[i].end());
        }
        return root;
    }
};
```

3.4 Tree Hash

```
// https://snuke.hatenablog.com/entry/2017/02/03/054210
class TreeHash {
    int n, mod;
    vector<vector<pair<int, int>>> adj;
    vector<int> label;
    vector<int> Hash, Hashp;
    map<pair<int, int>, int> es;
    LCA lca;
    StrongHash hash;

    int modpow(int a, long long n, int mod) {
        if (n == 0) return 1;
        if (n & 1) return 1ll * a * modpow(a, n - 1, mod) % mod;
        int t = modpow(a, n / 2, mod);
        return 1ll * t * t % mod;
    }

    void dfs(int v, int p) {
        long long ret = hash(label[v]);
        for (auto &s : adj[v]) {
            if (s.first != p) {
                dfs(s.first, v);
                // h * hのところは適
                ret += hash(s.second) + 1ll * Hash[s.first] * Hash[s.first] % mod;
            }
        }
        Hash[v] = ret % mod;
    }

    void dfs2(int v, int p) {
        for (auto &s : adj[v]) {
            if (s.first != p) {
                long long t = Hash[v];
                t -= hash(s.second) + 1ll * Hash[s.first] * Hash[s.first] % mod;
                t += Hashp[v];
                t %= mod;
                if (t < 0) t += mod;
                Hashp[s.first] = (hash(s.second) + t * t) % mod;
                dfs2(s.first, v);
            }
        }
    }

public:
    TreeHash(int n, int mod) : n(n), mod(mod), adj(n), label(n, 1), lca(n), Hash(n),
    Hashp(n) {}

    void add_edge(int a, int b, int w = 1) {
```

```
        assert(0 <= a && a < n && 0 <= b && b < n);
        adj[a].emplace_back(b, w);
        adj[b].emplace_back(a, w);
        lca.add_edge(a, b);
        es[{min(a, b), max(a, b)}] = w;
    }

    void set_label(int v, int x) {
        assert(0 <= v && v < n);
        label[v] = x;
    }

    void build() {
        dfs(0, -1);
        dfs2(0, -1);
        lca.build();
    }

    // rootを根として見たときの頂点v以下の部分木のhash
    int get(int root, int v) {
        if (root == v) return Hash[v] + Hashp[v];
        if (lca.lca(root, v) == v) {
            int s = lca.kth_node(v, root, 1);
            int w = es[{min(s, v), max(s, v)}];
            long long t = Hash[v];
            t -= hash(w) + 1ll * Hash[s] * Hash[s] % mod;
            t += Hashp[v];
            t %= mod;
            if (t < 0) t += mod;
            return t;
        } else {
            return Hash[v];
        }
    }
};
```

3.5 Maximum Independent Set Forest

```
struct MaximumIndependentSetForest {
    struct Edge {
        int to, type;
    }; // type 1: real edge, type 0: dummy edge to make connected
    vector<vector<Edge>> G;

    MaximumIndependentSetForest(const vector<pair<int, int>> &edges) {
        map<int, int> mp;
        for (auto &e : edges) {
            mp[e.first], mp[e.second];
        }
        int N = 0;
        for (auto &p : mp) {
            p.second = N++;
        }
        UnionFind uf(N);
        G = vector<vector<Edge>>(N);
        for (auto &e : edges) {
            int a = mp[e.first], b = mp[e.second];
            uf.unite(a, b);
            G[a].push_back({b, 1});
            G[b].push_back({a, 1});
        }
        for (int i = 1; i < N; i++) {
            if (!uf.same(0, i)) {
                G[0].push_back({i, 0});
                G[i].push_back({0, 0});
                uf.unite(0, i);
            }
        }
    }

    pair<int, int> dfs(int v, int p) {
        pair<int, int> ret = {1, 0};
        for (Edge &e : G[v]) {
            if (e.to != p) {
                pair<int, int> tmp = dfs(e.to, v);
                ret.second += max(tmp.first, tmp.second);
                if (e.type) {
                    ret.first += tmp.second;
                } else {
                    ret.first += max(tmp.first, tmp.second);
                }
            }
        }
        return ret;
    }

    int solve() {
        if (G.empty()) return 0;
        pair<int, int> ans = dfs(0, -1);
        return max(ans.first, ans.second);
    }
};
```

4 String

4.1 Rolling Hash

```
struct RollingHash {
    vector<int> hash;
    vector<int> pows;
    int p, m;

    RollingHash(string s, int p = 2, int m = 1000000007) : hash(s.size() + 1), pows(s.size()
    + 1), p(p),
                                                                    m(m) {
        hash[0] = 1;
        pows[0] = 1;
        for (int i = 0; i < s.size(); i++) {
            hash[i + 1] = (hash[i] * p % m + s[i]) % m;
            pows[i + 1] = pows[i] * p % m;
        }
    }

    // [l, r)
    int encode(int l, int r) {
        return ((hash[r] - hash[l] * pows[r - l] % m) % m + m) % m;
```

```
    }

    bool eq(int l0, int r0, int l1, int r1) {
        return encode(l0, r0) == encode(l1, r1);
    }
};
```

4.2 Z Algorithm

```
vector<int> z_algorithm(const string &s) {
    int n = s.size();
    vector<int> A(n);
    A[0] = n;
    int i = 1, j = 0;
    while (i < n) {
        while (i + j < n && s[j] == s[i + j]) {
            j++;
        }
        A[i] = j;
        if (j == 0) {
            i++;
            continue;
        }
        int k = 1;
        while (i + k < n && k + A[k] < j) {
            A[i + k] = A[k];
            k++;
        }
        i += k;
        j -= k;
    }
    return A;
}
```

4.3 Manacher

```
vector<int> manacher(const string &s) {
    int n = s.size();
    vector<int> ret(n);
    for (int i = 0, j = 0; i < n;) {
        while (i - j >= 0 && i + j < n && s[i - j] == s[i + j]) j++;
        ret[i] = j;
        int k = 1;
        while (i - k >= 0 && i + k < n && k + ret[i - k] < j) {
            ret[i + k] = ret[i - k];
            k++;
        }
        i += k;
        j -= k;
    }
    return ret;
}
```

4.4 Edit Distance

```
int edit_distance(const string &s, const string &t) {
    vector<vector<int>> dp(s.size() + 1, vector<int>(t.size() + 1));
    for (int i = 0; i <= s.size(); i++) {
        for (int j = 0; j <= t.size(); j++) {
            if (i == 0) {
                dp[i][j] = j;
            } else if (j == 0) {
                dp[i][j] = i;
            } else {
                dp[i][j] = INT_MAX;
            }
        }
    }

    for (int i = 1; i <= s.size(); i++) {
        for (int j = 1; j <= t.size(); j++) {
            if (s[i - 1] == t[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1];
            } else {
                dp[i][j] = min({dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]}) + 1;
            }
        }
    }
    return dp[s.size()][t.size()];
}
```

4.5 Longest Common Substring

```
string LCS(const string &s, const string &t) {
    int n = s.size(), m = t.size();
    vector<vector<int>> lcs(n + 1, vector<int>(m + 1));
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (s[i - 1] == t[j - 1]) {
                lcs[i][j] = max(lcs[i][j], lcs[i - 1][j - 1] + 1);
            } else {
                lcs[i][j] = max({lcs[i][j], lcs[i - 1][j], lcs[i][j - 1]});
            }
        }
    }
    string ret;
    for (int i = n, j = m; i > 0 && j > 0;) {
        if (lcs[i][j] == lcs[i - 1][j]) {
            i--;
            continue;
        }
        if (lcs[i][j] == lcs[i][j - 1]) {
            j--;
            continue;
        }
        assert(s[i - 1] == t[j - 1]);
        ret += s[i - 1];
        i--, j--;
    }
    reverse(ret.begin(), ret.end());
    return ret;
}
```

4.6 Suffix Array

```
// SA-ISによるSuffix Arrayの。構築O(N)
class SuffixArray {
    vector<int> sa_is(const vector<int> &str, const int k) {
        const int n = str.size();
        vector<bool> is_S(n);
```

```
    is_S[n - 1] = true;
    vector<bool> is_LMS(n);
    vector<int> LMSs;
    for (int i = n - 2; i >= 0; i--) {
        is_S[i] = str[i] < str[i + 1] || (str[i] == str[i + 1] && is_S[i + 1]);
    }
    REP(i, n)
    {
        if (is_S[i] & (i == 0 || !is_S[i - 1])) {
            is_LMS[i] = true;
            LMSs.push_back(i);
        }
    }
    vector<int> pseudo_sa = induced_sort(str, LMSs, is_S, k);
    vector<int> orderedLMSs(LMSs.size());
    int index = 0;
    for (int x: pseudo_sa) {
        if (is_LMS[x]) { orderedLMSs[index++] = x; }
    }
    pseudo_sa[orderedLMSs[0]] = 0;
    int rank = 0;
    if (orderedLMSs.size() > 1) { pseudo_sa[orderedLMSs[1]] = ++rank; }
    REPI(i, 1, orderedLMSs.size() - 1)
    {
        bool is_diff = false;
        REP(j, n)
        {
            int p = orderedLMSs[i] + j;
            int q = orderedLMSs[i + 1] + j;
            if (str[p] != str[q] || is_LMS[p] != is_LMS[q]) {
                is_diff = true;
                break;
            }
            if (j > 0 && is_LMS[p]) { break; }
        }
        pseudo_sa[orderedLMSs[i + 1]] = is_diff ? ++rank : rank;
    }
    vector<int> new_str(LMSs.size());
    index = 0;
    REP(i, n)
    {
        if (is_LMS[i]) { new_str[index++] = pseudo_sa[i]; }
    }
    vector<int> LMS_sa;
    if (rank + 1 == LMSs.size()) {
        LMS_sa = orderedLMSs;
    } else {
        LMS_sa = sa_is(new_str, rank + 1);
        for (int &x: LMS_sa) { x = LMSs[x]; }
    }
    return induced_sort(str, LMS_sa, is_S, k);
}

vector<int> induced_sort(const vector<int> &str, const vector<int> &LMSs, const
vector<bool> &is_S, const int k) {
    int n = str.size();
    vector<int> buckets(n);
    vector<int> chars(k + 1);
    for (int c: str) { chars[c + 1]++; }
    REP(i, k)
    { chars[i + 1] += chars[i]; }
    vector<int> count(k);
    for (int i = LMSs.size() - 1; i >= 0; i--) {
        int c = str[LMSs[i]];
        buckets[chars[c + 1] - 1 - count[c]++] = LMSs[i];
    }
    count = vector<int>(k);
    REP(i, n)
    {
        if (buckets[i] == 0 || is_S[buckets[i] - 1]) { continue; }
        int c = str[buckets[i] - 1];
        buckets[chars[c] + count[c]++] = buckets[i] - 1;
    }
    count = vector<int>(k);
    for (int i = n - 1; i >= 0; i--) {
        if (buckets[i] == 0 || !is_S[buckets[i] - 1]) { continue; }
        int c = str[buckets[i] - 1];
        buckets[chars[c + 1] - 1 - count[c]++] = buckets[i] - 1;
    }
    return buckets;
}

public:
    string S;
    int N;
    vector<int> sa; // sa[i]: suffixが書順i番目となる開始位置のindex
    SuffixArray(string str_in) : S(str_in), N(str_in.size()) {
        str_in += "$";
        vector<int> str(N + 1);
        REP(i, N + 1)
        { str[i] = str_in[i] - '$'; }
        sa = sa_is(str, 128);
        sa.erase(sa.begin());
    }

    int operator[](int index) {
        return sa[index];
    }

    // sizeがTと等しく初めてT以上になるようなSの部分文字列(sa)
    vector<int>::iterator lower_bound(string T) {
        int l = -1, r = N;
        while (r - l > 1) {
            int mid = (l + r) / 2;
            if (S.compare(sa[mid], T.size(), T) < 0) {
                l = mid;
            } else {
                r = mid;
            }
        }
        return sa.begin() + r;
    }
```

```
    // sizeがTと等しく初めてTより大きくなるようなSの部分文字列(sa)
    vector<int>::iterator upper_bound(string T) {
        int l = -1, r = N;
        while (r - l > 1) {
            int mid = (l + r) / 2;
            if (S.compare(sa[mid], T.size(), T) <= 0) {
                l = mid;
            } else {
                r = mid;
            }
        }
        return sa.begin() + r;
    }

    // S2が部分文字列として何回出現するか
    int count(string S2) {
        return upper_bound(S2) - lower_bound(S2);
    }
};
```

4.7 Segment Tree (For Longest Common Prefix)

```
template<class T = int>
class SegTree {
    using VT = vector<T>;
    int orig_n;

    // k番目のノードの[l, r)について[a, b)を求める
    T query(int a, int b, int k, int l, int r) {
        if (r <= a || b <= l) { return UNIT; }
        if (a <= l && r <= b) { return dat[k]; }
        T vl = query(a, b, k * 2 + 1, l, (l + r) / 2);
        T vr = query(a, b, k * 2 + 2, (l + r) / 2, r);
        return f(vl, vr);
    }

public:
    int N;
    VT dat;
    function<T(T, T)> f;
    int UNIT;

    SegTree(int n, function<T(T, T)> f_, const T unit) {
        orig_n = n;
        f = f_;
        UNIT = unit;
        for (N = 1; N < n; N *= 2);
        dat = VT(2 * N - 1, UNIT);
    }

    SegTree(VT a = {}, function<T(T, T)> f_ = [](int a, int b) { return min(a, b); }, T unit
    = 1e15) {
        orig_n = a.size();
        f = f_;
        UNIT = unit;
        for (N = 1; N < a.size(); N *= 2);
        dat = VT(2 * N - 1);
        REP(i, a.size()) {
            dat[N - 1 + i] = a[i];
        }
        for (int k = N - 2; k >= 0; k--) {
            dat[k] = f(dat[2 * k + 1], dat[2 * k + 2]);
        }
    }

    // k番目をaに
    void update(int k, int a) {
        k += N - 1;
        dat[k] = a;
        while (k > 0) {
            k = (k - 1) / 2;
            dat[k] = f(dat[2 * k + 1], dat[2 * k + 2]);
        }
    }

    // [a, b)でのクエリ
    T query(int a, int b) {
        assert(0 <= a && a < b && b <= orig_n);
        return query(a, b, 0, 0, N);
    }
};
```

4.8 Longest Common Prefix

5 Range Query

5.1 Range Count Query by Comparision

```
class RangeCount {
    const int ST_SIZE = (1 << 20) - 1;
    int n;
    vector<int> data;
    vector<vector<int>> segtree;

    void init(int k, int l, int r) {
        if (r - l == 1) {
            segtree[k].push_back(data[l]);
        } else {
            int lch = k * 2 + 1;
            int rch = k * 2 + 2;
            init(lch, l, (l + r) / 2);
            init(rch, (l + r) / 2, r);
            segtree[k].resize(r - l);

            merge(segtree[lch].begin(), segtree[lch].end(), segtree[rch].begin(),
            segtree[rch].end(),
                  segtree[k].begin());
        }
    }

    // number of x in [i, j)
    int query(int i, int j, int x, int k, int l, int r) {
        if (j <= l || r <= i) {
            return 0;
        }
```

```
        if (i <= l && r <= j) {
            return upper_bound(segtree[k].begin(), segtree[k].end(), x) -
            segtree[k].begin();
        }

        int lc = query(i, j, x, k * 2 + 1, l, (l + r) / 2);
        int rc = query(i, j, x, k * 2 + 2, (l + r) / 2, r);
        return lc + rc;
    }

public:
    RangeCount(const vector<int> &v) {
        n = v.size();
        data = vector<int>(v);
        segtree = vector<vector<int>>(ST_SIZE);
        init(0, 0, n);
    }

    int exact(int i, int j, int x) {
        return query(i, j, x, 0, 0, n) - query(i, j, x - 1, 0, 0, n);
    }

    int le(int i, int j, int x) {
        return query(i, j, x, 0, 0, n);
    }

    int lt(int i, int j, int x) {
        return query(i, j, x - 1, 0, 0, n);
    }

    int ge(int i, int j, int x) {
        return query(i, j, 1e9, 0, 0, n) - query(i, j, x - 1, 0, 0, n);
    }

    int gt(int i, int j, int x) {
        return query(i, j, 1e9, 0, 0, n) - query(i, j, x, 0, 0, n);
    }
};
```

5.2 Range Number of Distinct Elements Query

```
constexpr int MAX_N = 202020;
constexpr int MAX_ROOT = 8000000;

// https://github.com/anh1l1ator/Spoj/blob/master/Desktop/Codes/DQUERY_Online.cpp
struct Node {
    int cnt, L, R;

    Node() {
        cnt = 0;
        L = R = -1;
    }

    Node(int x, int y, int z) {
        L = x, R = y, cnt = z;
    }
} tree[MAX_ROOT];

class DistinctElements {
    int gc = 0, N;
    vector<int> rt;

    int build(int L, int R) {
        gc++;
        if (L == R) return gc;
        int x = gc;
        tree[x] = Node(build(L, (L + R) / 2), build((L + R) / 2 + 1, R), 0);
        return x;
    }

    int update(int L, int R, int root, int idx, int val) {
        assert(root < MAX_ROOT);
        if (L > idx || R < idx) {
            return root;
        }
        gc++;
        if (L == idx && R == idx) {
            tree[gc] = Node(-1, -1, tree[root].cnt + val);
            return gc;
        }
        int x = gc;
        tree[x] = Node(update(L, (L + R) / 2, tree[root].L, idx, val),
                       update((L + R) / 2 + 1, R, tree[root].R, idx, val), tree[root].cnt +
                       val);
        return x;
    }

    int query(int L, int R, int root, int ql, int qr) {
        if (qr < L || ql > R) return 0;
        if (ql <= L && R <= qr) {
            return tree[root].cnt;
        }
        return query(L, (L + R) / 2, tree[root].L, ql, qr) + query((L + R) / 2 + 1, R,
        tree[root].R, ql, qr);
    }

    int query1idx(int l, int r) {
        return query(1, N, rt[r], l, r);
    }

public:
    DistinctElements(const vector<int> &as) {
        N = as.size();
        rt = vector<int>(MAX_N);
        vector<int> G(1 << 20, -1);
        rt[0] = build(1, N);
        for (int i = 1; i <= N; i++) {
            int p = rt[i - 1];
            if (G[as[i - 1]] != -1) {
                p = update(1, N, p, G[as[i - 1]], -1);
            }
            rt[i] = update(1, N, p, i, 1);
            G[as[i - 1]] = i;
```

```
        }
    }

    int query(int l, int r) {
        return query1idx(l + 1, r);
    }
};
```

5.3 Pair Query

```
// (type, value)の配列にし、[l, r)になるtype kのvalueについての累積。とりあえず累積和
struct PairQuery {
    vector<vector<pair<int, int>>> acc;
    PairQuery(const vector<pair<int, int>> &as, int MAX_TYPE) : acc(MAX_TYPE,
    vector<pair<int, int>>(1, {-1, 0})) {
        int n = as.size();
        for (int i = 0; i < n; i++) {
            int t = as[i].first, v = as[i].second;
            if (t == -1) continue;
            acc[t].emplace_back(i, acc[t].back().second + v);
        }
    }
    // [l, r)
    int query(int l, int r, int k) {
        auto R = lower_bound(acc[k].begin(), acc[k].end(), make_pair(r, 0));
        auto L = lower_bound(acc[k].begin(), acc[k].end(), make_pair(l, 0));
        --R;
        --L;
        return R->second - L->second;
    }
};
```

5.4 Mo Base

```
struct Mo {
    int block;
    vector<int> order, le, ri;
    vector<bool> visited;
    function<void(int)> add, del;
    function<int()> get;
    int nl, nr;
    Mo() {}
    Mo(int n, function<void(int)> add, function<void(int)> del, function<int()> get) :
    block((int) sqrt(n)), nl(0), nr(0), visited(n), add(move(add)), del(move(del)),
    get(move(get)) {};

    // [l, r)
    void add_query(int l, int r) {
        le.push_back(l);
        ri.push_back(r);
    }

    void shift(int idx) {
        visited[idx].flip();
        if (visited[idx]) add(idx);
        else del(idx);
    }

    vector<int> answer() {
        int qcnt = le.size();
        order.resize(qcnt);
        vector<int> ret(qcnt);
        iota(order.begin(), order.end(), 0);
        sort(order.begin(), order.end(), [&](int a, int b) {
            return le[a] / block != le[b] / block ? le[a] < le[b] : ri[a] < ri[b];
        });
        for (int i = 0; i < qcnt; i++) {
            int idx = order[i];
            while (nl > le[idx]) shift(--nl);
            while (nr < ri[idx]) shift(nr++);
            while (nl < le[idx]) shift(nl++);
            while (nr > ri[idx]) shift(--nr);
            ret[order[i]] = get();
        }
        return ret;
    }
};
```

5.5 Range Mode Query Offline

```
struct OfflineRangeModeQuery {
    int n;
    int MAX;
    vector<int> as;
    // cnt[a]: aの現在のカウント
    // cntrev[k]: カウントがkの個が何種類あるか
    // best: 現在のmodeのカウント
    vector<int> cnt, cntrev;
    int best = 0;
    vector<array<int, 4>> queries;
    Mo mo;
    OfflineRangeModeQuery(const vector<int> &as, int MAX) : n(as.size()), as(as), MAX(MAX),
    cnt(MAX + 1), cntrev(n + 1) {
        auto add = [&](int idx) {
            cntrev[cnt[as[idx]]]--;
            cnt[as[idx]]++;
            cntrev[cnt[as[idx]]]++;
            if (cnt[as[idx]] > best) best++;
        };
        auto del = [&](int idx) {
            cntrev[cnt[as[idx]]]--;
            if (cnt[as[idx]] == best && cntrev[best] == 0) best--;
            cnt[as[idx]]--;
            cntrev[cnt[as[idx]]]++;
        };
        auto ans = [&]() {
            return best;
        };
        mo = Mo(n, add, del, ans);
    }

    // [l, r]閉間!!!!!!!!!! 0-indexed
    void add_query(int l, int r) {
        mo.add_query(l, r + 1);
    }
    vector<int> answer() {
        return mo.answer();
```

```
    }
};
```

5.6 Tree Mo

```
// vertex query
// edgeクエリは書いていないが、add, delを2引にして、add(from, to), del(from, to)のときに、(from, to)の重みを操作すればよさそう
// edgeクエリのときはLCAがいらない
struct TreeMo {
    int n, block;
    vector<int> order, le, ri, lcas, in, vs;
    vector<bool> visited;
    function<void(int)> add, del;
    function<int()> get;
    int nl, nr;
    vector<vector<int>> adj;
    LCA lca;
    bool built = false;

    TreeMo() {}

    TreeMo(int n, function<void(int)> add, function<void(int)> del, function<int()> get) :
    n(n), adj(n), block((int) sqrt(2 * n - 1)), nl(0), nr(0), visited(n), in(n),
    add(move(add)), del(move(del)), get(move(get)) {}

    void dfs(int v, int p, int d) {
        in[v] = vs.size();
        vs.push_back(v);
        for (int s : adj[v]) {
            if (s != p) {
                dfs(s, v, d + 1);
                vs.push_back(s);
            }
        }
    }

    void add_edge(int a, int b) {
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

    void build_tree() {
        lca = LCA(adj);
        dfs(0, -1, 0);
        built = true;
    }

    void add_query(int u, int v) {
        assert(built && 0 <= u && u < n && 0 <= v && v < n);
        if (in[u] > in[v]) swap(u, v);
        le.push_back(in[u] + 1);
        ri.push_back(in[v] + 1);
        lcas.push_back(lca.lca(u, v));
    }

    void shift(int v) {
        visited[v].flip();
        if (visited[v]) add(v);
        else del(v);
    }

    vector<int> answer() {
        int qcnt = le.size();
        order.resize(qcnt);
        vector<int> ret(qcnt);
        iota(order.begin(), order.end(), 0);
        sort(order.begin(), order.end(), [&](int a, int b) {
            return le[a] / block != le[b] / block ? le[a] < le[b] : ri[a] < ri[b];
        });
        for (int i = 0; i < qcnt; i++) {
            if (i > 0) shift(lcas[order[i - 1]]);
            int idx = order[i];
            while (nl > le[idx]) shift(vs[--nl]);
            while (nr < ri[idx]) shift(vs[nr++]);
            while (nl < le[idx]) shift(vs[nl++]);
            while (nr > ri[idx]) shift(vs[--nr]);
            shift(lcas[idx]);
            ret[idx] = get();
        }
        return ret;
    }
};
```

6 Data Structure

6.1 Union Find Tree

```
struct UnionFind {
    int n, cnt;
    vector<int> par, rank, sz;

    UnionFind(int n) : n(n), cnt(n), par(n), rank(n), sz(n, 1) {
        iota(par.begin(), par.end(), 0);
    }

    int find(int x) {
        if (x == par[x]) return x;
        return par[x] = find(par[x]);
    }

    bool same(int x, int y) {
        return find(x) == find(y);
    }

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

    void unite(int x, int y) {
        x = find(x), y = find(y);
        if (x == y) return;
        if (rank[x] < rank[y]) {
            par[x] = y;
            sz[y] += sz[x];
        } else {
```

```
            par[y] = x;
            sz[x] += sz[y];
            if (rank[x] == rank[y]) {
                rank[x]++;
            }
        }
        cnt--;
    }
};
```

6.2 Parially Parsistent Union Find Tree

```
class PartiallyParsistentUnionFind {
    vector<int> rank, par, time;
    const int INF = 1e9;
public:
    PartiallyParsistentUnionFind(int n) {
        rank.resize(n);
        par.resize(n, -1);
        time.resize(n, INF);
    }

    int find(int t, int x) {
        if (time[x] > t) return x;
        return find(t, par[x]);
    }

    bool unite(int t, int x, int y) {
        x = find(t, x);
        y = find(t, y);
        if (x == y) return false;
        if (rank[x] > rank[y]) {
            par[y] = x;
            time[y] = t;
        } else {
            par[x] = y;
            time[x] = t;
            if (rank[x] == rank[y]) {
                rank[y]++;
            }
        }
        return true;
    }

    bool same(int t, int x, int y) {
        return find(t, x) == find(t, y);
    }
};
```

6.3 Potential Union Find Tree

```
int par[101010];
int diff[101010];

int find(int x) {
    if (par[x] == x) return x;
    int r = find(par[x]);
    diff[x] += diff[par[x]];
    return par[x] = r;
}

int weight(int x) {
    find(x);
    return diff[x];
}

void unite(int x, int y, int w) {
    w += weight(x);
    w -= weight(y);
    x = find(x);
    y = find(y);
    par[y] = x;
    diff[y] = w;
}
```

6.4 Union Find Tree with Undo

```
struct UnionFindUndo {
    vector<int> data;
    stack<pair<int, int>> history;

    UnionFindUndo(int sz) {
        data.assign(sz, -1);
    }

    bool unite(int x, int y) {
        x = find(x), y = find(y);
        history.emplace(x, data[x]);
        history.emplace(y, data[y]);
        if (x == y) return (false);
        if (data[x] > data[y]) swap(x, y);
        data[x] += data[y];
        data[y] = x;
        return (true);
    }

    int find(int k) {
        if (data[k] < 0) return (k);
        return (find(data[k]));
    }

    int size(int k) {
        return (-data[find(k)]);
    }

    void undo() {
        data[history.top().first] = history.top().second;
        history.pop();
        data[history.top().first] = history.top().second;
        history.pop();
    }

    void snapshot() {
        while (history.size()) history.pop();
    }

    void rollback() {
```

```
        while (history.size()) undo();
    }
};
```

6.5 Sparse Table

```
template<typename T>
struct SparseTable {
    vector<vector<T>> st;
    vector<int> lookup;

    SparseTable() {}

    SparseTable(const vector<T> &v) {
        int b = 0;
        while ((1 << b) <= v.size()) b++;
        st.assign(b, vector<T>(1 << b));
        for (int i = 0; i < v.size(); i++) {
            st[0][i] = v[i];
        }
        for (int i = 1; i < b; i++) {
            for (int j = 0; j + (1 << i) <= 1 << b; j++) {
                st[i][j] = min(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
            }
        }
        lookup.resize(v.size() + 1);
        for (int i = 2; i < lookup.size(); i++) {
            lookup[i] = lookup[i >> 1] + 1;
        }
    }

    T query(int l, int r) {
        int b = lookup[r - l];
        return min(st[b][l], st[b][r - (1 << b)]);
    }
};
```

6.6 Binary Trie

```
template<typename U = unsigned, int B = 32>
class lazy_binary_trie {
    struct node {
        int cnt;
        U lazy;
        node *ch[2];
        node() : cnt(0), lazy(0), ch{ nullptr, nullptr } {}
    };
    void push(node* t, int b) {
        if ((t->lazy >> (U)b) & (U)1) swap(t->ch[0], t->ch[1]);
        if (t->ch[0]) t->ch[0]->lazy ^= t->lazy;
        if (t->ch[1]) t->ch[1]->lazy ^= t->lazy;
        t->lazy = 0;
    }
    node* add(node* t, U val, int b = B - 1) {
        if (!t) t = new node;
        t->cnt += 1;
        if (b < 0) return t;
        push(t, b);
        bool f = (val >> (U)b) & (U)1;
        t->ch[f] = add(t->ch[f], val, b - 1);
        return t;
    }
    node* sub(node* t, U val, int b = B - 1) {
        assert(t);
        t->cnt -= 1;
        if (t->cnt == 0) return nullptr;
        if (b < 0) return t;
        push(t, b);
        bool f = (val >> (U)b) & (U)1;
        t->ch[f] = sub(t->ch[f], val, b - 1);
        return t;
    }
    U get_min(node* t, U val, int b = B - 1) {
        assert(t);
        if (b < 0) return 0;
        push(t, b);
        bool f = (val >> (U)b) & (U)1; f ^= !t->ch[f];
        return get_min(t->ch[f], val, b - 1) | ((U)f << (U)b);
    }
    U get(node* t, int k, int b = B - 1) {
        if (b < 0) return 0;
        push(t, b);
        int m = t->ch[0] ? t->ch[0]->cnt : 0;
        return k < m ? get(t->ch[0], k, b - 1) : get(t->ch[1], k - m, b - 1) | ((U)1 <<
        (U)b);
    }
    int count_lower(node* t, U val, int b = B - 1) {
        if (!t || b < 0) return 0;
        push(t, b);
        bool f = (val >> (U)b) & (U)1;
        return (f && t->ch[0] ? t->ch[0]->cnt : 0) + count_lower(t->ch[f], val, b - 1);
    }
    node *root;
public:
    lazy_binary_trie() : root(nullptr) {}
    int size() const {
        return root ? root->cnt : 0;
    }
    bool empty() const {
        return !root;
    }
    void insert(U val) {
        root = add(root, val);
    }
    void erase(U val) {
        root = sub(root, val);
    }
    void xor_all(U val) {
        if (root) root->lazy ^= val;
    }
    U max_element(U bias = 0) {
        return get_min(root, ~bias);
    }
    U min_element(U bias = 0) {
        return get_min(root, bias);
```

```
    }
    int lower_bound(U val) { // return id
        return count_lower(root, val);
    }
    int upper_bound(U val) { // return id
        return count_lower(root, val + 1);
    }
    U operator[](int k) {
        assert(0 <= k && k < size());
        return get(root, k);
    }
    int count(U val) {
        if (!root) return 0;
        node *t = root;
        for (int i = B - 1; i >= 0; i--) {
            push(t, i);
            t = t->ch[(val >> (U)i) & (U)1];
            if (!t) return 0;
        }
        return t->cnt;
    }
};
```

6.7 Implicit Treap Base

```
template<class Monoid, class OperatorMonoid>
class ImplicitTreap {
    random_device rnd;

    struct Node {
        T value, acc, lazy;
        int priority, cnt;
        bool rev;
        Node *l, *r;

        Node(T value, int priority) : value(value), acc(Monoid::id()),
        lazy(OperatorMonoid::id()), priority(priority),
                                      cnt(1), rev(false), l(nullptr), r(nullptr) {}
    } *root = nullptr;

    using Tree = Node *;

    int cnt(Tree t) {
        return t ? t->cnt : 0;
    }

    T acc(Tree t) {
        return t ? t->acc : Monoid::id();
    }

    void update_cnt(Tree t) {
        if (t) {
            t->cnt = 1 + cnt(t->l) + cnt(t->r);
        }
    }

    void update_acc(Tree t) {
        if (t) {
            t->acc = Monoid::op(acc(t->l), Monoid::op(t->value, acc(t->r)));
        }
    }

    void pushup(Tree t) {
        update_cnt(t), update_acc(t);
    }

    void pushdown(Tree t) {
        if (t && t->rev) {
            t->rev = false;
            swap(t->l, t->r);
            if (t->l) t->l->rev ^= 1;
            if (t->r) t->r->rev ^= 1;
        }
        if (t && t->lazy != OperatorMonoid::id()) {
            if (t->l) {
                t->l->lazy = OperatorMonoid::op(t->l->lazy, t->lazy);
                t->l->acc = Modifier::op(t->l->acc, t->lazy, cnt(t->l));
            }
            if (t->r) {
                t->r->lazy = OperatorMonoid::op(t->r->lazy, t->lazy);
                t->r->acc = Modifier::op(t->r->acc, t->lazy, cnt(t->r));
            }
            t->value = Modifier::op(t->value, t->lazy, 1);
            t->lazy = OperatorMonoid::id();
        }
        pushup(t);
    }

    void split(Tree t, int key, Tree &l, Tree &r) {
        if (!t) {
            l = r = nullptr;
            return;
        }
        pushdown(t);
        int implicit_key = cnt(t->l) + 1;
        if (key < implicit_key) {
            split(t->l, key, l, t->l), r = t;
        } else {
            split(t->r, key - implicit_key, t->r, r), l = t;
        }
        pushup(t);
    }

    void insert(Tree &t, int key, Tree item) {
        Tree t1, t2;
        split(t, key, t1, t2);
        merge(t1, t1, item);
        merge(t, t1, t2);
    }

    void merge(Tree &t, Tree l, Tree r) {
        pushdown(l);
        pushdown(r);
        if (!l || !r) {
```

```
            t = l ? l : r;
        } else if (l->priority > r->priority) {
            merge(l->r, l->r, r), t = l;
        } else {
            merge(r->l, l, r->l), t = r;
        }
        pushup(t);
    }

    void erase(Tree &t, int key) {
        Tree t1, t2, t3;
        split(t, key + 1, t1, t2);
        split(t1, key, t1, t3);
        merge(t, t1, t2);
    }

    void update(Tree t, int l, int r, T x) {
        if (l >= r) return;
        Tree t1, t2, t3;
        split(t, l, t1, t2);
        split(t2, r - l, t2, t3);
        t2->lazy = OperatorMonoid::op(t2->lazy, x);
        t2->acc = Modifier::op(t2->acc, x, cnt(t2));
        merge(t2, t2, t3);
        merge(t, t1, t2);
    }

    T query(Tree t, int l, int r) {
        if (l == r) return Monoid::id();
        Tree t1, t2, t3;
        split(t, l, t1, t2);
        split(t2, r - l, t2, t3);
        T ret = t2->acc;
        merge(t2, t2, t3);
        merge(t, t1, t2);
        return ret;
    }

    // [l, r)の中で左から何番目か
    int find(Tree t, T x, int offset, bool left = true) {
        if (Monoid::op(t->acc, x) == x) {
            return -1;
        } else {
            if (left) {
                if (t->l && Monoid::op(t->l->acc, x) != x) {
                    return find(t->l, x, offset, left);
                } else {
                    return (Monoid::op(t->value, x) != x) ? offset + cnt(t->l) : find(t->r,
                    x, offset + cnt(t->l) + 1,
                                                                                    left);
                }
            } else {
                if (t->r && Monoid::op(t->r->acc, x) != x) {
                    return find(t->r, x, offset + cnt(t->l) + 1, left);
                } else {
                    return (Monoid::op(t->value, x) != x) ? offset + cnt(t->l) : find(t->l,
                    x, offset, left);
                }
            }
        }
    }

    void reverse(Tree t, int l, int r) {
        if (l > r) return;
        Tree t1, t2, t3;
        split(t, l, t1, t2);
        split(t2, r - l, t2, t3);
        t2->rev ^= 1;
        merge(t2, t2, t3);
        merge(t, t1, t2);
    }

    // [l, r)の先頭がmになるようにシフトさせる。std::rotateと同じ仕
    void rotate(Tree t, int l, int m, int r) {
        reverse(t, l, r);
        reverse(t, l, l + r - m);
        reverse(t, l + r - m, r);
    }

    void dump(Tree t) {
        if (!t) return;
        pushdown(t);
        dump(t->l);
        cout << t->value << " ";
        dump(t->r);
    }

public:
    ImplicitTreap() {}

    ImplicitTreap(vector<T> as) {
        ::reverse(as.begin(), as.end());
        for (T a : as) {
            insert(0, a);
        }
    }

    int size() {
        return cnt(root);
    }

    void insert(int pos, T x) {
        insert(root, pos, new Node(x, rnd.random()));
    }

    void update(int l, int r, T x) {
        update(root, l, r, x);
    }

    T query(int l, int r) {
        return query(root, l, r);
```

```
    }

    // 二分探索。[l, r)のkでMonoid::op(tr[k], x) != xとなる最左/最右のもの。存在しない場合は-1
    // たとえばMinMonoidの場合、x未の最左/最右の要素の位置を返す
    int binary_search(int l, int r, T x, bool left = true) {
        if (l >= r) return -1;
        Tree t1, t2, t3;
        split(root, l, t1, t2);
        split(t2, r - l, t2, t3);
        int ret = find(t2, x, l, left);
        merge(t2, t2, t3);
        merge(root, t1, t2);
        return ret;
    }

    void erase(int pos) {
        erase(root, pos);
    }

    void reverse(int l, int r) {
        reverse(root, l, r);
    }

    void rotate(int l, int m, int r) {
        rotate(root, l, m, r);
    }

    void dump() {
        dump(root);
        cout << endl;
    }

    T operator[](int pos) {
        return query(pos, pos + 1);
    }
};
```

6.8 Implicit Treap Monoids

```
struct SumMonoid {
    static constexpr T id() {
        return 0;
    }

    static T op(T a, T b) {
        return a + b;
    }
};

struct MinMonoid {
    static constexpr T id() {
        return 2e18;
    }

    static T op(T a, T b) {
        return min(a, b);
    }
};

// 本はSumMonoid用
struct UpdateMonoid {
    static constexpr T id() {
        return 2e18;
    }

    static T op(T a, T b) {
        return b;
    }
};

struct Modifier {
    // lazyの結果によってaccがどうわるか。szは部分木のサイズ
    static T op(T a, T b, int sz) {
        return b == UpdateMonoid::id() ? a : b * sz;
    }
};
```

7 Flow

7.1 Dinic

```
// O(V^2E)
struct Dinic {
    struct edge {
        int to, cap, rev;
    };

    int n;
    vector<vector<edge>> G;
    vector<int> level;
    vector<int> iter;

    void bfs(int s) {
        level.assign(n, -1);
        queue<int> que;
        que.push(s);
        level[s] = 0;
        while (que.size()) {
            int v = que.front();
            que.pop();
            for (int i = 0; i < G[v].size(); i++) {
                edge &e = G[v][i];
                if (e.cap > 0 && level[e.to] < 0) {
                    level[e.to] = level[v] + 1;
                    que.push(e.to);
                }
            }
        }
    }

    int dfs(int v, int t, int f) {
        if (v == t) return f;
        for (int i = iter[v]; i < G[v].size(); i++) {
            edge &e = G[v][i];
            if (e.cap > 0 && level[v] < level[e.to]) {
                int d = dfs(e.to, t, min(f, e.cap));
```

```
                if (d > 0) {
                    e.cap -= d;
                    G[e.to][e.rev].cap += d;
                    return d;
                }
            }
        }
        return 0;
    }

    Dinic(int n) : n(n), G(n), level(n), iter(n) {}

    void add_edge(int from, int to, int cap) {
        G[from].push_back({to, cap, (int) G[to].size()});
        G[to].push_back({from, 0, (int) G[from].size() - 1});
    }

    int max_flow(int s, int t) {
        int flow = 0;
        while (1) {
            bfs(s);
            if (level[t] < 0) return flow;
            iter.assign(n, 0);
            int f;
            while ((f = dfs(s, t, 1e9)) > 0) {
                flow += f;
            }
        }
    }
};
```

7.2 Ford Fulkerson

```
// O(FE)
class FordFulkerson {
    struct edge {
        int to, cap, rev;
    };

    int n;
    vector<vector<edge>> G;
    vector<bool> used;

    int dfs(int v, int t, int f) {
        if (v == t) return f;
        used[v] = true;

        for (int i = 0; i < G[v].size(); i++) {
            edge &e = G[v][i];
            if (!used[e.to] && e.cap > 0) {
                int d = dfs(e.to, t, min(f, e.cap));
                if (d > 0) {
                    e.cap -= d;
                    G[e.to][e.rev].cap += d;
                    return d;
                }
            }
        }
        return 0;
    }

public:
    FordFulkerson(int n) : n(n), G(n), used(n) {}

    void addEdge(int from, int to, int cap) {
        G[from].push_back({to, cap, int(G[to].size())});
        G[to].push_back({from, 0, int(G[from].size()) - 1});
    }

    int max_flow(int s, int t) {
        int flow = 0;
        while (1) {
            used.assign(n, 0);
            int f = dfs(s, t, 1e9);
            if (f == 0) return flow;
            flow += f;
        }
    }
};
```

8 Matching

8.1 Hungarian Bipartite Weighted Matching

```
// 二部グラフの最大重みマッチング
class Hungarian {
    int n, p, q;
    vector<vector<int>> mat;
    vector<int> fx, fy, x, y;
    const int INF = 1e9;

public:
    Hungarian(const vector<vector<int>> &mat) : n(mat.size()), fx(n, INF), fy(n), x(n, -1),
    y(n, -1), mat(mat) {}

    int run() {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                fx[i] = max(fx[i], mat[i][j]);
            }
        }

        for (int i = 0; i < n;) {
            vector<int> t(n, -1), s(n + 1, i);
            for (p = q = 0; p <= q && x[i] < 0; p++) {
                for (int k = s[p], j = 0; j < n && x[i] < 0; j++) {
                    if (fx[k] + fy[j] == mat[k][j] && t[j] < 0) {
                        s[++q] = y[j];
                        t[j] = k;
                        if (s[q] < 0) {
                            for (p = j; p >= 0; j = p) {
                                y[j] = k = t[j];
                                p = x[k];
                                x[k] = j;
                            }
                        }
```

```
                }
            }
        }
        if (x[i] < 0) {
            int d = INF;
            for (int k = 0; k <= q; k++) {
                for (int j = 0; j < n; j++) {
                    if (t[j] < 0) {
                        d = min(d, fx[s[k]] + fy[j] - mat[s[k]][j]);
                    }
                }
            }
            for (int j = 0; j < n; j++) {
                fy[j] += (t[j] < 0 ? 0 : d);
            }
            for (int k = 0; k <= q; k++) {
                fx[s[k]] -= d;
            }
        } else {
            i++;
        }
    }
    int ret = 0;
    for (int i = 0; i < n; i++) {
        ret += mat[i][x[i]];
    }
    return ret;
}

int match_y(int k) {
    return x[k];
}

int match_x(int k) {
    return y[k];
}
};
```

8.2 Blossom General Matching

```
class Blossom {
    // 1-based vertex index
    vector<int> vis, par, orig, match, aux;
    int t = 0, N;
    vector<vector<int>> conn;
    queue<int> Q;

    void augment(int u, int v) {
        int pv = v, nv;
        do {
            pv = par[v], nv = match[pv];
            match[v] = pv, match[pv] = v;
            v = nv;
        } while (u != pv);
    }
    int lca(int v, int w) {
        ++t;
        while (1) {
            if (v) {
                if (aux[v] == t) {
                    return v;
                }
                aux[v] = t;
                v = orig[par[match[v]]];
            }
            swap(v, w);
        }
    }

    void blossom(int v, int w, int a) {
        while (orig[v] != a) {
            par[v] = w;
            w = match[v];
            if (vis[w] == 1) {
                Q.push(w), vis[w] = 0;
            }
            orig[v] = orig[w] = a;
            v = par[w];
        }
    }

    bool bfs(int u) {
        vis.assign(N + 1, -1);
        iota(orig.begin(), orig.end(), 0);
        Q = queue<int>();
        Q.push(u);
        vis[u] = 0;
        while (Q.size()) {
            int v = Q.front();
            Q.pop();
            for (int x : conn[v]) {
                if (vis[x] == -1) {
                    par[x] = v;
                    vis[x] = 1;
                    if (!match[x]) {
                        return augment(u, x), true;
                    }
                    Q.push(match[x]);
                    vis[match[x]] = 0;
                } else if (vis[x] == 0 && orig[v] != orig[x]) {
                    int a = lca(orig[v], orig[x]);
                    blossom(x, v, a);
                    blossom(v, x, a);
                }
            }
        }
        return false;
    }

public:
    Blossom(int n) : vis(n + 1), par(n + 1), orig(n + 1), match(n + 1), aux(n + 1), N(n),
    conn(n + 1) {}

    void addEdge(int u, int v) {
```

```
        assert(1 <= u && u <= N && 1 <= v && v <= N);
        conn[u].push_back(v);
        conn[v].push_back(u);
    }

    int Match() {
        int ans = 0;
        vector<int> V(N - 1);
        iota(V.begin(), V.end(), 1);
        shuffle(V.begin(), V.end(),
        mt19937(chrono::steady_clock::now().time_since_epoch().count()));
        for (int x : V) {
            if (!match[x]) {
                for (int y : conn[x]) {
                    if (!match[y]) {
                        match[x] = y;
                        match[y] = x;
                        ++ans;
                        break;
                    }
                }
            }
        }
        for (int i = 1; i <= N; i++) {
            if (!match[i] && bfs(i)) ++ans;
        }
        return ans;
    }
};
```

9 Math

9.1 Sieve

```
struct Sieve {
    vector<int> smallest_factor;
    vector<int> smallest_power;
    vector<int> moebius;
    vector<int> totient;
    vector<bool> prime;
    vector<int> primes;
    Sieve(int m) : smallest_factor(m + 1), smallest_power(m + 1), moebius(m + 1), totient(m
    + 1), prime(m + 1, true) {
        moebius[1] = totient[1] = 1;
        prime[0] = prime[1] = false;

        for (int i = 2; i <= m; i++) {
            if (prime[i]) {
                smallest_factor[i] = i;
                smallest_power[i] = 1;
                moebius[i] = -1;
                totient[i] = i - 1;
                primes.push_back(i);
            }

            for (int p : primes) {
                if (p > smallest_factor[i] || i * p > m) break;
                prime[i * p] = false;
                smallest_factor[i * p] = p;
                smallest_power[i * p] = smallest_factor[i] == p ? smallest_power[i] + 1 : 1;
                moebius[i * p] = smallest_factor[i] == p ? 0 : -moebius[i];
                totient[i * p] = smallest_factor[i] == p ? p * totient[i] : (p - 1) *
                totient[i];
            }
        }
    }
};
```

9.2 Extended Euclid

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

9.3 Miller Rabin Prime Check

```
mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());

struct MillerRabin {
    int modpow(int a, int n, int mod) {
        if (n == 0) return 1;
        if (n % 2 == 0) {
            int t = modpow(a, n / 2, mod);
            return t * t % mod;
        }
        return a * modpow(a, n - 1, mod) % mod;
    }

    int modinv(int n, int mod) {
        return modpow(n, mod - 2, mod);
    }

    bool is_prime(int n, int k = 50) {
        if (n == 2) return true;
        if (n < 2 || n % 2 == 0) return false;
        int d = n - 1;
        while (d % 2 == 0) {
            d /= 2;
        }
        for (int i = 0; i < k; i++) {
            int a = rnd() % (n - 2) + 1;
            int t = d;
            int y = modpow(a, t, n);
            while (t != n - 1 && y != 1 && y != n - 1) {
                y = modpow(y, 2, n);
                t *= 2;
            }
            if (y != n - 1 && t % 2 == 0) {
                return false;
            }
        }
```

```
        return true;
    }
};
```

9.4 Combination nonprime

```
struct Combination {
    int n;
    vector<vector<int>> dp;
    Combination(int n) : n(n) {}

    void build() {
        dp = vector<vector<int>>(n + 1, vector<int>(n + 1));
        for (int i = 0; i <= n; i++) {
            dp[i][0] = 1;
            dp[i][i] = 1;
        }
        for (int i = 2; i <= n; i++) {
            for (int j = 1; j < i; j++) {
                dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j];
                dp[i][j] %= MOD;
            }
        }
    }

    int built_ncr(int n, int r) {
        return dp[n][r];
    }

    // avoid MLE
    int ncr(int n, int r) {
        if (n < 2) return 1;
        vector<int> cur(2, 1);
        for (int i = 2; i <= n; i++) {
            vector<int> nex(n + 1, 1);
            for (int j = 1; j < i; j++) {
                nex[j] = cur[j - 1] + cur[j];
                nex[j] %= MOD;
            }
            cur = move(nex);
        }
        return cur[r];
    }
};
```

9.5 Chinese Remainder Theorem

```
pair<int, int> chrem(const vector<int> &ps, const vector<int> &rs) {
    using Long = __int128_t;
    int P = 1;
    for (int p: ps) {
        P *= p;
    }
    Long ret = 0;
    for (int i = 0; i < ps.size(); i++) {
        int p = P / ps[i];
        ret += Long(1) * rs[i] * modinv(p, ps[i]) * p;
    }
    return make_pair(ret % P, P);
}
```

9.6 Moebius Transform

```
// before: ss[i]: 集合族iの共通部分の大きさ
// after : ss[i]: 集合族iに含まれていて、集合族~iに含まれない部分の大きさ
void moebius(vector<int> &ss) {
    int N = ss.size();
    int n = 0;
    while (N > 1) {
        n++;
        N >>= 1;
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < (1 << n); j++) {
            if (!(j & (1 << i))) {
                ss[j] -= ss[j | (1 << i)];
            }
        }
    }
}

int intsqrt(int n) {
    int l = 0, r = n + 1;
    while (l < r - 1) {
        int m = (l + r) / 2;
        if (__int128_t(m) * m <= n) {
            l = m;
        } else {
            r = m;
        }
    }
    return l;
}
```

9.7 baby-step Giant step

```
// 必要にじて #define int long long
int modpow(int a, int n, int mod) {
    if (n == 0) return 1;
    if (n % 2 == 0) {
        int t = modpow(a, n / 2, mod);
        return t * t % mod;
    }
    return a * modpow(a, n - 1, mod) % mod;
}

int modinv(int a, int mod) {
    return modpow(a, mod - 2, mod);
}

int modlog(int b, int y, int mod) {
    // find minimam x such that modpow(b, x, mod) == y
    b %= mod;
    y %= mod;
    assert(b);
    int l = -1, r = mod;
```

```
    while (l < r - 1) {
        int m = (l + r) / 2;
        if (m * m >= mod) r = m;
        else l = m;
    }
    int sqrtM = r;
    unordered_map<int, int> bpow;
    int p = 1;
    for (int i = 0; i < sqrtM; i++) {
        if (!bpow.count(p)) {
            bpow[p] = i;
        }
        p *= b;
        p %= mod;
    }

    int B = modpow(modinv(b, mod), sqrtM, mod);
    p = y;
    for (int i = 0; i < sqrtM; i++) {
        if (bpow.count(p)) {
            return i * sqrtM + bpow[p];
        }
        p *= B;
        p %= mod;
    }

    return -1;
}
```

9.8 Xor Gauss-Jordan Elimination

```
vector<long long> rbd(vector<long long> mat) {
    int n = mat.size();
    int rk = 0;
    for (int i = 62; i >= 0; i--) {
        bool exist = false;
        for (int j = rk; j < n; j++) {
            if (mat[j] >> i & 1) {
                exist = true;
                swap(mat[rk], mat[j]);
                break;
            }
        }
        if (exist) {
            for (int j = 0; j < n; j++) {
                if (j != rk && mat[j] >> i & 1) {
                    mat[j] ^= mat[rk];
                }
            }
            rk++;
        }
    }
    return mat;
}
```

9.9 Modulo Integer

```
template<typename T>
T pow(T a, long long n, T e = 1) {
    T ret = e;
    while (n) {
        if (n & 1) ret *= a;
        a *= a;
        n >>= 1;
    }
    return ret;
}

template<int mod>
struct ModInt {
    int x;
    ModInt() : x(0) {}
    ModInt(long long x_) { if ((x = x_ % mod + mod) >= mod) x -= mod; }
    ModInt& operator+=(ModInt rhs) { if ((x += rhs.x) >= mod) x -= mod; return *this; }
    ModInt& operator-=(ModInt rhs) { if ((x -= rhs.x) < 0) x += mod; return *this; }
    ModInt& operator*=(ModInt rhs) { x = (unsigned long long) x * rhs.x % mod; return *this;
    }
    ModInt& operator/=(ModInt rhs) { x = (unsigned long long) x * rhs.inv().x % mod; return
    *this; }

    ModInt operator-() const { return -x < 0 ? mod - x : -x; }
    ModInt operator+(ModInt rhs) const { return ModInt(*this) += rhs; }
    ModInt operator-(ModInt rhs) const { return ModInt(*this) -= rhs; }
    ModInt operator*(ModInt rhs) const { return ModInt(*this) *= rhs; }
    ModInt operator/(ModInt rhs) const { return ModInt(*this) /= rhs; }
    bool operator==(ModInt rhs) const { return x == rhs.x; }
    bool operator!=(ModInt rhs) const { return x != rhs.x; }
    ModInt inv() const { return pow(*this, mod - 2); }

    friend ostream& operator<<(ostream& s, ModInt<mod> a) { s << a.x; return s; }
    friend istream& operator>>(istream& s, ModInt<mod>& a) { s >> a.x; return s; }
};
```

9.10 Matrix

```
template<typename T>
class mat : public vector<vector<T>> {
private:
    int r, c;
public:
    int row() const {
        return r;
    }

    int column() const {
        return c;
    }

    mat(int n, int m, T val = 0) {
        r = n;
        c = m;
        for (int i = 0; i < n; i++) {
            this->push_back(vector<T>(m, val));
        }
    }

    mat operator+(const mat &rhs) {
```

```
    assert(r == rhs.r && c == rhs.c);
    mat<T> ret(r, c);
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            ret[i][j] = (*this)[i][j] + rhs[i][j];
        }
    }
    return ret;
}

mat operator+(const T val) {
    mat<T> ret(r, c);
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            ret[i][j] = (*this)[i][j] + val;
        }
    }
    return ret;
}

mat operator-(const mat &rhs) {
    assert(r == rhs.r && c == rhs.c);
    mat<T> ret(r, c);
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            ret[i][j] = (*this)[i][j] - rhs[i][j];
        }
    }
    return ret;
}

mat operator-(const T val) {
    mat<T> ret(r, c);
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            ret[i][j] = (*this)[i][j] - val;
        }
    }
    return ret;
}

vector<T> operator*(const vector<T> &rhs) {
    assert(c == rhs.size());
    vector<T> vec(r, 0);
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            vec[i] += (*this)[i][j] * rhs[j];
        }
    }
    return vec;
}

mat operator*(const mat &rhs) {
    assert(c == rhs.r);
    mat<T> ret(r, rhs.c);
    for (int i = 0; i < r; i++) {
        for (int k = 0; k < c; k++) {
            for (int j = 0; j < rhs.c; j++) {
                ret[i][j] += (*this)[i][k] * rhs[k][j];
            }
        }
    }
    return ret;
}

mat operator-() {
    mat<T> ret(r, c);
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            ret[i][j] = -(*this)[i][j];
        }
    }
    return ret;
}

int rank() {
    int res = 0;
    mat<double> B(r, c);
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            B[i][j] = (*this)[i][j];
        }
    }
    for (int i = 0; i < c; i++) {
        if (res == r) return res;
        int pivot = res;
        for (int j = res + 1; j < r; j++) {
            if (abs(B[j][i]) > abs(B[pivot][i])) {
                pivot = j;
            }
        }
        if (abs(B[pivot][i]) < EPS) continue;
        swap(B[pivot], B[res]);
        for (int j = i + 1; j < c; j++) {
            B[res][j] /= B[res][i];
        }
        for (int j = res + 1; j < r; j++) {
            for (int k = i + 1; k < c; k++) {
                B[j][k] -= B[res][k] * B[j][i];
            }
        }
        res++;
    }
    return res;
}

T det() {
    assert(r == c);
    T ans = 1;
    mat B(r, r);
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < r; j++) {
```

```
                B[i][j] = (*this)[i][j];
            }
        }
        for (int i = 0; i < r; i++) {
            for (int j = i + 1; j < r; j++) {
                for (; B[j][i] != 0; ans = -ans) {
                    T tm = B[i][i] / B[j][i];
                    for (int k = i; k < r; k++) {
                        T t = B[i][k] - tm * B[j][k];
                        B[i][k] = B[j][k];
                        B[j][k] = t;
                    }
                }
            }
            ans *= B[i][i];
        }
        return ans;
    }

    mat inverse() {
        assert(r == c);
        mat B(r, 2 * r);
        for (int i = 0; i < r; i++) {
            for (int j = 0; j < r; j++) {
                B[i][j] = (*this)[i][j];
            }
        }
        for (int i = 0; i < r; i++) {
            B[i][r + i] = 1;
        }
        for (int i = 0; i < r; i++) {
            int pivot = i;
            for (int j = i; j < r; j++) {
                if (abs(B[j][i]) > abs(B[pivot][i])) {
                    pivot = j;
                }
            }

            // assert regular
            assert(abs(B[pivot][i]) > EPS);
            swap(B[i], B[pivot]);
            for (int j = i + 1; j <= 2 * r; j++) {
                B[i][j] /= B[i][i];
            }
            for (int j = 0; j < r; j++) {
                if (i != j) {
                    for (int k = i + 1; k <= 2 * r; k++) {
                        B[j][k] -= B[j][i] * B[i][k];
                    }
                }
            }
        }
        mat res(r, r);
        for (int i = 0; i < r; i++) {
            for (int j = 0; j < r; j++) {
                res[i][j] = B[i][r + j];
            }
        }
        return res;
    }

    void print() {
        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c - 1; j++) {
                cout << (*this)[i][j] << '\t';
            }
            cout << (*this)[i][c - 1] << endl;
        }
    }
};

template<typename T>
vector<T> eq_solve(const mat<T> &A, const vector<T> &b) {
    assert(A.row() == A.column());
    int n = A.row();
    mat<T> B(n, n + 1);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            B[i][j] = A[i][j];
        }
    }
    for (int i = 0; i < n; i++) {
        B[i][n] = b[i];
    }
    for (int i = 0; i < n; i++) {
        int pivot = i;
        for (int j = i; j < n; j++) {
            if (abs(B[j][i]) > abs(B[pivot][i])) {
                pivot = j;
            }
        }
        // assert having a unique root
        assert(abs(B[pivot][i]) > EPS);
        swap(B[i], B[pivot]);
        for (int j = i + 1; j <= n; j++) {
            B[i][j] /= B[i][i];
        }
        for (int j = 0; j < n; j++) {
            if (i != j) {
                for (int k = i + 1; k <= n; k++) {
                    B[j][k] -= B[j][i] * B[i][k];
                }
            }
        }
    }
    vector<T> ret(n);
    for (int i = 0; i < n; i++) {
        ret[i] = B[i][n];
    }
    return ret;
}
```

10 Misc

10.1 pbds

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

using Tree = tree<int, null_type, less<>, rb_tree_tag, tree_order_statistics_node_update>;
```

10.2 Slide Min

```
struct SlideMin {
    // data[i]: mininum over [max(0, i + 1 - width), i + 1)
    int n;
    vector<int> data;

    SlideMin(const vector<int> &as, int width, bool minimum = true) : n(as.size()),
    data(as.size()) {
        using ii = pair<int, int>;
        deque<ii> deq;
        auto comp = [&](ii &e, int v) {
            return minimum ? e.first >= v : e.first <= v;
        };
        for (int i = 0; i < n; i++) {
            if (deq.size() && deq.front().second <= i - width) {
                deq.pop_front();
            }
            while (deq.size()) {
                if (comp(deq.back(), as[i])) deq.pop_back();
                else break;
            }
            deq.push_back(ii(as[i], i));
            data[i] = deq.front().first;
        }
    }
};
```

10.3 Inversion Number

```
long long inv_number(vector<int> &as) {
    int cnt = 0;
    int n = as.size();
    if (n > 1) {
        vector<int> bs(as.begin(), as.begin() + n / 2);
        vector<int> cs(as.begin() + n / 2, as.end());
        cnt += inv_number(bs);
        cnt += inv_number(cs);
        for (int i = 0, j = 0, k = 0; i < n; i++) {
            if (k == cs.size()) {
                as[i] = bs[j++];
            } else if (j == bs.size()) {
                as[i] = cs[k++];
            } else if (bs[j] <= cs[k]) {
                as[i] = bs[j++];
            } else {
                as[i] = cs[k++];
                cnt += n / 2 - j;
            }
        }
    }
    return cnt;
}
```

10.4 Largest Rectangle

```
int largest_rectangle(vector<int> &hist) {
    hist.push_back(0);
    stack<pair<int, int>> st;
    int ret = 0;
    for (int i = 0; i < hist.size(); i++) {
        if (st.empty() || st.top().first < hist[i]) {
            st.emplace(hist[i], i);
        } else {
            int s = 0;
            while (!st.empty() && st.top().first >= hist[i]) {
                ret = max(ret, st.top().first * (i - st.top().second));
                s = st.top().second;
                st.pop();
            }
            st.emplace(hist[i], s);
        }
    }

    return ret;
}
```

11 Famous DP

11.1 Convex Hull Trick

```
template<typename T = double, bool is_min = true>
class ConvexHullTrick {
    using P = pair<T, T>;
    deque<P> L;

    T getY(const P &a, const T &x) {
        return a.first * x + a.second;
    }

    bool check(const P &a, const P &b, const P &c) {
        return (b.first - a.first) * (c.second - b.second) >= (b.second - a.second) *
        (c.first - b.first);
    }

    bool empty() const {
        return L.empty();
    }

public:
    void add_line(T a, T b) {
        if (!is_min) {
            a *= -1;
            b *= -1;
        }
        P line(a, b);
        if (!L.empty() && L.back().first == a) {
            line.second = min(line.second, L.back().second);
            L.pop_back();
        }
        while (L.size() >= 2 && check(L[L.size() - 2], L[L.size() - 1], line)) {
```

```
            L.pop_back();
        }
        L.emplace_back(line);
    }

    T query(T x) {
        assert(!empty());
        int l = -1, r = L.size() - 1;
        while (l < r - 1) {
            int m = (l + r) / 2;
            if (getY(L[m], x) >= getY(L[m + 1], x)) {
                l = m;
            } else {
                r = m;
            }
        }
        return (!is_min ? -1 : 1) * getY(L[r], x);
    }

    T query_monotone(T x) {
        assert(!empty());
        while (L.size() >= 2 && getY(L[0], x) >= getY(L[1], x)) {
            L.pop_front();
        }
        return (!is_min ? -1 : 1) * getY(L[0], x);
    }
};
```

11.2 Doubling

```
struct Doubling {
    int n;
    int size; // MSB + 1
    vector<vector<int>> next; // next[k][i]: iから(1<<k)回でどこまで進めるか

    // edge[i]: 1回でiからどこまで進めるか
    Doubling(vector<int> &edge) : n(edge.size()), size(64 - __builtin_clzll(edge.size())) {
        next.resize(size, vector<int>(n + 1, n));
        for (int i = 0; i < n; i++) next[0][i] = edge[i];
        for (int k = 0; k < size - 1; k++) {
            for (int i = 0; i < n; i++) {
                next[k + 1][i] = next[k][next[k][i]];
            }
        }
    }

    // i番目のx個先
    int get(int i, int x) {
        int ret = i;
        for (int bit = 0; bit < size; bit++) {
            if (!(x >> bit & 1)) continue;
            ret = next[bit][ret];
        }
        return ret;
    }

    // iからはじめて何回進めば初めてj以上になるか
    // j以上になりえないときはnを返す
    int lower_bound(int i, int j) {
        int cur = i, acc = 0;
        for (int wid = size - 1; wid >= 0; wid--) {
            if (next[wid][cur] < j) {
                acc += 1LL << wid;
                cur = next[wid][cur];
            }
        }
        return min(n, acc + 1);
    }
};
```

11.3 Partition Count

```
int partition_count(int n, int m, int mod) {
    // divide n (undistinguished) items into m (undistinguished) groups, groups can have 0
    items
    vector<vector<int>> dp(m + 1, vector<int>(n + 1));
    dp[0][0] = 1;
    for (int i = 1; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (j - i >= 0) {
                dp[i][j] = (dp[i - 1][j] + dp[i][j - i]) % mod;
            } else {
                dp[i][j] = dp[i - 1][j];
            }
        }
    }
    return dp[m][n];
}
```

12 Geometry

12.1 Base

```
using val_t = Rational;
using Point = complex<val_t>;
using Polygon = vector<Point>;
const val_t EPS = 0;
struct Line : public pair<Point, Point> {
    Line(const Point &a, const Point &b) : pair<Point, Point>(a, b) {}
};
struct Segment : public pair<Point, Point> {
    Segment(const Point &a, const Point &b) : pair<Point, Point>(a, b) {}
};
val_t dot(const Point &a, const Point &b) {
    return real(conj(a) * b);
}
val_t cross(const Point &a, const Point &b) {
    return imag(conj(a) * b);
}
val_t X(const Point &a) {
    return real(a);
}
val_t Y(const Point &a) {
    return imag(a);
}
val_t norm2(const Point &a) {
    return X(a) * X(a) + Y(a) * Y(a);
}
```

```
int sign(val_t x) {
    if (x < -EPS) return -1;
    if (x > EPS) return +1;
    return 0;
}
int ccw(const Point &a, Point b, Point c) {
    b -= a, c -= a;
    if (cross(b, c) > EPS) return +1; // a->b->c is ccw
    if (cross(b, c) < -EPS) return -1; // a->b->c is cw
    if (dot(b, c) < 0) return 0; // b--a--c
    if (norm2(b) < norm2(c)) return +2; // a--b--c
    return -2; // b--c--a
}
bool is_crossing(const Segment &a, const Segment &b) {
    return ccw(a.first, a.second, b.first) * ccw(a.first, a.second, b.second) <= 0
        && ccw(b.first, b.second, a.first) * ccw(b.first, b.second, a.second) <= 0;
}
Point intersection(const Line &a, const Line &b) {
    return a.first + (a.second - a.first) * cross(b.second - b.first, a.first - b.first) /
    cross(b.second - b.first, a.first - a.second);
}
val_t dist2(const Point &a, const Point &b) {
    return norm2(a - b);
}
val_t dist2(const Line &line, const Point &p) {
    val_t t = cross(p - line.first, line.second - line.first);
    return t * t / norm2(line.second - line.first);
}
val_t dist2(const Point &p, const Line &line) {
    return dist2(line, p);
}
val_t dist2(const Segment &segment, const Point &p) {
    if (sign(dot(segment.first - segment.second, p - segment.second)) *
    sign(dot(segment.second - segment.first, p - segment.first)) >= 0) {
        return dist2(Line(segment.first, segment.second), p);
    }
    return min(norm2(p - segment.first), norm2(p - segment.second));
}
val_t dist2(const Point &p, const Segment &segment) {
    return dist2(segment, p);
}
val_t dist2(const Segment &a, const Segment &b) {
    if (is_crossing(a, b)) return val_t(0);
    return min({dist2(a, b.first), dist2(a, b.second), dist2(b, a.first), dist2(b,
    a.second)});
}

bool operator<(const Point &a, const Point &b) {
    return X(a) != X(b) ? X(a) < X(b) : Y(a) < Y(b);
}
```

12.2 Convex Hull

```
// assume all points are not on a same line
Polygon convex_hull(vector<Point> ps, bool ignore_on_edge = false) {
    int n = ps.size(), k = 0;
    sort(ps.begin(), ps.end());
    Polygon ret(2 * n);
    for (int i = 0; i < n; ret[k++] = ps[i++]) {
        if (!ignore_on_edge) {
            while (k >= 2 && ccw(ret[k - 2], ret[k - 1], ps[i]) <= 0) k--;
        } else {
            while (k >= 2 && ccw(ret[k - 2], ret[k - 1], ps[i]) != 1) k--;
        }
    }
    for (int i = n - 2, t = k + 1; i >= 0; ret[k++] = ps[i--]) {
        if (!ignore_on_edge) {
            while (k >= t && ccw(ret[k - 2], ret[k - 1], ps[i]) <= 0) k--;
        } else {
            while (k >= t && ccw(ret[k - 2], ret[k - 1], ps[i]) != 1) k--;
        }
    }
    ret.resize(k - 1);
    return ret;
}
```

13 ei1333

13.1 Template

```
template<typename T>
struct edge {
    int src, to;
    T cost;

    edge(int to, T cost) : src(-1), to(to), cost(cost) {}

    edge(int src, int to, T cost) : src(src), to(to), cost(cost) {}

    edge &operator=(const int &x) {
        to = x;
        return *this;
    }

    operator int() const { return to; }
};

template<typename T>
using Edges = vector<edge<T> >;
template<typename T>
using WeightedGraph = vector<Edges<T> >;
using UnWeightedGraph = vector<vector<int> >;
template<typename T>
using Matrix = vector<vector<T>>;
```

13.2 Euler Path

```
// s は始点
template<typename T>
vector<edge<T> > eulerian_path(Edges<T> es, int s, bool directed) {
    int V = 0;
    for (auto &e : es) V = max(V, max(e.to, e.src) + 1);
    vector<vector<pair<edge<T>, int> > > g(V);
    for (auto &e : es) {
        int sz_to = (int) g[e.to].size();
        g[e.src].emplace_back(e, sz_to);
        if (!directed) {
            int sz_src = (int) g[e.src].size() - 1;
```

```
            swap(e.src, e.to);
            g[e.src].emplace_back(e, sz_src);
        }
    }
    vector<edge<T> > ord;
    stack<pair<int, edge<T> > > st;
    st.emplace(s, edge<T>(-1, -1));
    while (st.size()) {
        int idx = st.top().first;
        if (g[idx].empty()) {
            ord.emplace_back(st.top().second);
            st.pop();
        } else {
            auto e = g[idx].back();
            g[idx].pop_back();
            if (e.second == -1) continue;
            if (!directed) g[e.first.to][e.second].second = -1;
            st.emplace(e.first.to, e.first);
        }
    }
    ord.pop_back();
    reverse(begin(ord), end(ord));
    if (ord.size() != es.size()) return {};
    return ord;
}
```

13.3 Hopcroft Karp Bipartite Matching

```
// O(EV^0.5)
struct HopcroftKarp {
    vector<vector<int>> graph;
    vector<int> dist, match;
    vector<bool> used, vv;

    HopcroftKarp(int n, int m) : graph(n), match(m, -1), used(n) {}

    void add_edge(int u, int v) {
        graph[u].push_back(v);
    }

    void bfs() {
        dist.assign(graph.size(), -1);
        queue<int> que;
        for (int i = 0; i < graph.size(); i++) {
            if (!used[i]) {
                que.emplace(i);
                dist[i] = 0;
            }
        }

        while (!que.empty()) {
            int a = que.front();
            que.pop();
            for (auto &b : graph[a]) {
                int c = match[b];
                if (c >= 0 && dist[c] == -1) {
                    dist[c] = dist[a] + 1;
                    que.emplace(c);
                }
            }
        }
    }

    bool dfs(int a) {
        vv[a] = true;
        for (auto &b : graph[a]) {
            int c = match[b];
            if (c < 0 || (!vv[c] && dist[c] == dist[a] + 1 && dfs(c))) {
                match[b] = a;
                used[a] = true;
                return (true);
            }
        }
        return (false);
    }

    int bipartite_matching() {
        int ret = 0;
        while (true) {
            bfs();
            vv.assign(graph.size(), false);
            int flow = 0;
            for (int i = 0; i < graph.size(); i++) {
                if (!used[i] && dfs(i)) ++flow;
            }
            if (flow == 0) return (ret);
            ret += flow;
        }
    }

    void output() {
        for (int i = 0; i < match.size(); i++) {
            if (~match[i]) {
                cout << match[i] << "-" << i << endl;
            }
        }
    }
};
```

13.4 Chromatic Number

```
// 隣接行列を渡すと彩色を返す
// O(n2^n)
int chromatic_number(const Matrix<bool> &g) {
    int N = (int) g.size();
    vector<int> es(N);
    for (int i = 0; i < g.size(); i++) {
        for (int j = 0; j < g.size(); j++) {
            es[i] |= g[i][j] << j;
        }
    }
    int ret = N;
    for (int d : {7, 11, 21}) {
        int mod = 1e9 + d;
        vector<int> ind(1 << N), aux(1 << N, 1);
        ind[0] = 1;
```

```
        for (int S = 1; S < 1 << N; S++) {
            int u = __builtin_ctz(S);
            ind[S] = ind[S ^ (1 << u)] + ind[(S ^ (1 << u)) & ~es[u]];
        }
        for (int i = 1; i < ret; i++) {
            int64_t all = 0;
            for (int j = 0; j < 1 << N; j++) {
                int S = j ^(j >> 1);
                aux[S] = (1LL * aux[S] * ind[S]) % mod;
                all += j & 1 ? aux[S] : mod - aux[S];
            }
            if (all % mod) ret = i;
        }
    }
    return ret;
}
```

13.5 Maximum Independent Set

```
template<typename T>
vector<int> maximum_independent_set(const Matrix <T> &g, int trial = 1000000) {
    int N = (int) g.size();
    vector<uint64_t> bit(N);

    assert(N <= 64);
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (i != j) {
                assert(g[i][j] == g[j][i]);
                if (g[i][j]) bit[i] |= uint64_t(1) << j;
            }
        }
    }

    vector<int> ord(N);
    iota(begin(ord), end(ord), 0);
    mt19937 mt(chrono::steady_clock::now().time_since_epoch().count());
    int ret = 0;
    uint64_t ver;
    for (int i = 0; i < trial; i++) {
        shuffle(begin(ord), end(ord), mt);
        uint64_t used = 0;
        int add = 0;
        for (int j : ord) {
            if (used & bit[j]) continue;
            used |= uint64_t(1) << j;
            ++add;
        }
        if (ret < add) {
            ret = add;
            ver = used;
        }
    }
    vector<int> ans;
    for (int i = 0; i < N; i++) {
        if ((ver >> i) & 1) ans.emplace_back(i);
    }
    return ans;
}
```

13.6 LowLink

```
// build() でグラフgにする LowLink を構築する。構築後，articulation には節点，bridge
には橋が追加される。非連結でもOK。
template<typename G>
struct LowLink {
    const G &g;
    vector<int> used, ord, low;
    vector<int> articulation;
    vector<pair<int, int> > bridge;

    LowLink(const G &g) : g(g) {}

    int dfs(int idx, int k, int par) {
        used[idx] = true;
        ord[idx] = k++;
        low[idx] = ord[idx];
        bool is_articulation = false;
        int cnt = 0;
        for (auto &to : g[idx]) {
            if (!used[to]) {
                ++cnt;
                k = dfs(to, k, idx);
                low[idx] = min(low[idx], low[to]);
                is_articulation |= ~par && low[to] >= ord[idx];
                if (ord[idx] < low[to]) bridge.emplace_back(minmax(idx, (int) to));
            } else if (to != par) {
                low[idx] = min(low[idx], ord[to]);
            }
        }
        is_articulation |= par == -1 && cnt > 1;
        if (is_articulation) articulation.push_back(idx);
        return k;
    }

    virtual void build() {
        used.assign(g.size(), 0);
        ord.assign(g.size(), 0);
        low.assign(g.size(), 0);
        int k = 0;
        for (int i = 0; i < g.size(); i++) {
            if (!used[i]) k = dfs(i, k, -1);
        }
    }
};
```

13.7 Fast Fourier Transform

```
// multiply(a, b) で a と b をみんだ配列を返す
namespace FastFourierTransform {
    using real = double;

    struct C {
        real x, y;

        C() : x(0), y(0) {}
```

```
    C(real x, real y) : x(x), y(y) {}

    inline C operator+(const C &c) const { return C(x + c.x, y + c.y); }

    inline C operator-(const C &c) const { return C(x - c.x, y - c.y); }

    inline C operator*(const C &c) const { return C(x * c.x - y * c.y, x * c.y + y *
    c.x); }

    inline C conj() const { return C(x, -y); }
};

const real PI = acosl(-1);
int base = 1;
vector<C> rts = {{0, 0},
                 {1, 0}};
vector<int> rev = {0, 1};


void ensure_base(int nbase) {
    if (nbase <= base) return;
    rev.resize(1 << nbase);
    rts.resize(1 << nbase);
    for (int i = 0; i < (1 << nbase); i++) {
        rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
    }
    while (base < nbase) {
        real angle = PI * 2.0 / (1 << (base + 1));
        for (int i = 1 << (base - 1); i < (1 << base); i++) {
            rts[i << 1] = rts[i];
            real angle_i = angle * (2 * i + 1 - (1 << base));
            rts[(i << 1) + 1] = C(cos(angle_i), sin(angle_i));
        }
        ++base;
    }
}

void fft(vector<C> &a, int n) {
    assert((n & (n - 1)) == 0);
    int zeros = __builtin_ctz(n);
    ensure_base(zeros);
    int shift = base - zeros;
    for (int i = 0; i < n; i++) {
        if (i < (rev[i] >> shift)) {
            swap(a[i], a[rev[i] >> shift]);
        }
    }
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                C z = a[i + j + k] * rts[j + k];
                a[i + j + k] = a[i + j] - z;
                a[i + j] = a[i + j] + z;
            }
        }
    }
}

vector<int64_t> multiply(const vector<int> &a, const vector<int> &b) {
    int need = (int) a.size() + (int) b.size() - 1;
    int nbase = 1;
    while ((1 << nbase) < need) nbase++;
    ensure_base(nbase);
    int sz = 1 << nbase;
    vector<C> fa(sz);
    for (int i = 0; i < sz; i++) {
        int x = (i < (int) a.size() ? a[i] : 0);
        int y = (i < (int) b.size() ? b[i] : 0);
        fa[i] = C(x, y);
    }
    fft(fa, sz);
    C r(0, -0.25 / (sz >> 1)), s(0, 1), t(0.5, 0);
    for (int i = 0; i <= (sz >> 1); i++) {
        int j = (sz - i) & (sz - 1);
        C z = (fa[j] * fa[j] - (fa[i] * fa[i]).conj()) * r;
        fa[j] = (fa[i] * fa[i] - (fa[j] * fa[j]).conj()) * r;
        fa[i] = z;
    }
    for (int i = 0; i < (sz >> 1); i++) {
        C A0 = (fa[i] + fa[i + (sz >> 1)]) * t;
        C A1 = (fa[i] - fa[i + (sz >> 1)]) * t * rts[(sz >> 1) + i];
        fa[i] = A0 + A1 * s;
    }
    fft(fa, sz >> 1);
    vector<int64_t> ret(need);
    for (int i = 0; i < need; i++) {
        ret[i] = llround(i & 1 ? fa[i >> 1].y : fa[i >> 1].x);
    }
    return ret;
}
};
```

13.8 Dinic (For Limited Flow)

```
template<typename flow_t>
struct Dinic {
    const flow_t INF;

    struct edge {
        int to;
        flow_t cap;
        int rev;
        bool isrev;
    };

    vector<vector<edge> > graph;
    vector<int> min_cost, iter;

    Dinic(int V) : INF(numeric_limits<flow_t>::max()), graph(V) {}

    void add_edge(int from, int to, flow_t cap) {
        graph[from].emplace_back((edge) {to, cap, (int) graph[to].size(), false});
        graph[to].emplace_back((edge) {from, 0, (int) graph[from].size() - 1, true});
    }
```

```
    bool bfs(int s, int t) {
        min_cost.assign(graph.size(), -1);
        queue<int> que;
        min_cost[s] = 0;
        que.push(s);
        while (!que.empty() && min_cost[t] == -1) {
            int p = que.front();
            que.pop();
            for (auto &e : graph[p]) {
                if (e.cap > 0 && min_cost[e.to] == -1) {
                    min_cost[e.to] = min_cost[p] + 1;
                    que.push(e.to);
                }
            }
        }
        return min_cost[t] != -1;
    }

    flow_t dfs(int idx, const int t, flow_t flow) {
        if (idx == t) return flow;
        for (int &i = iter[idx]; i < graph[idx].size(); i++) {
            edge &e = graph[idx][i];
            if (e.cap > 0 && min_cost[idx] < min_cost[e.to]) {
                flow_t d = dfs(e.to, t, min(flow, e.cap));
                if (d > 0) {
                    e.cap -= d;
                    graph[e.to][e.rev].cap += d;
                    return d;
                }
            }
        }
        return 0;
    }

    flow_t max_flow(int s, int t) {
        flow_t flow = 0;
        while (bfs(s, t)) {
            iter.assign(graph.size(), 0);
            flow_t f = 0;
            while ((f = dfs(s, t, INF)) > 0) flow += f;
        }
        return flow;
    }

    void output() {
        for (int i = 0; i < graph.size(); i++) {
            for (auto &e : graph[i]) {
                if (e.isrev) continue;
                auto &rev_e = graph[e.to][e.rev];
                cout << i << "->" << e.to << " (flow: " << rev_e.cap << "/" << e.cap +
                rev_e.cap << ")" << endl;
            }
        }
    }
};
```

13.9 Limited Flow

```
// 頂点で初期化。流せない場合は -1

template<typename flow_t, template<typename> class F>
struct MaxFlowLowerBound {
    F<flow_t> flow;
    int X, Y;
    flow_t low_sum;

    MaxFlowLowerBound(int V) : flow(V + 2), X(V), Y(V + 1), low_sum(0) {}

    void add_edge(int from, int to, flow_t low, flow_t high) {
        flow.add_edge(from, to, high - low);
        flow.add_edge(X, to, low);
        flow.add_edge(from, Y, low);
        low_sum += low;
    }

    flow_t max_flow(int s, int t) {
        auto a = flow.max_flow(X, Y);
        auto b = flow.max_flow(s, Y);
        auto c = flow.max_flow(X, t);
        auto d = flow.max_flow(s, t);
        return b == c && a + b == low_sum ? b + d : -1;
    }
};
```

13.10 Primal Dual Mincost Flow

```
// 流せた場合はコスト，流せなかった場合は -1
// O(FElogV)
template<typename flow_t, typename cost_t>
struct PrimalDual {
    const cost_t INF;

    struct edge {
        int to;
        flow_t cap;
        cost_t cost;
        int rev;
        bool isrev;
    };
    vector<vector<edge> > graph;
    vector<cost_t> potential, min_cost;
    vector<int> prevv, preve;

    PrimalDual(int V) : graph(V), INF(numeric_limits<cost_t>::max()) {}

    void add_edge(int from, int to, flow_t cap, cost_t cost) {
        graph[from].emplace_back((edge) {to, cap, cost, (int) graph[to].size(), false});
        graph[to].emplace_back((edge) {from, 0, -cost, (int) graph[from].size() - 1, true});
    }

    cost_t min_cost_flow(int s, int t, flow_t f) {
        int V = (int) graph.size();
        cost_t ret = 0;
        using Pi = pair<cost_t, int>;
        priority_queue<Pi, vector<Pi>, greater<Pi> > que;
```

```
        potential.assign(V, 0);
        preve.assign(V, -1);
        prevv.assign(V, -1);

        while (f > 0) {
            min_cost.assign(V, INF);
            que.emplace(0, s);
            min_cost[s] = 0;
            while (!que.empty()) {
                Pi p = que.top();
                que.pop();
                if (min_cost[p.second] < p.first) continue;
                for (int i = 0; i < graph[p.second].size(); i++) {
                    edge &e = graph[p.second][i];
                    cost_t nextCost = min_cost[p.second] + e.cost + potential[p.second] -
                    potential[e.to];
                    if (e.cap > 0 && min_cost[e.to] > nextCost) {
                        min_cost[e.to] = nextCost;
                        prevv[e.to] = p.second, preve[e.to] = i;
                        que.emplace(min_cost[e.to], e.to);
                    }
                }
            }
            if (min_cost[t] == INF) return -1;
            for (int v = 0; v < V; v++) potential[v] += min_cost[v];
            flow_t addflow = f;
            for (int v = t; v != s; v = prevv[v]) {
                addflow = min(addflow, graph[prevv[v]][preve[v]].cap);
            }
            f -= addflow;
            ret += addflow * potential[t];
            for (int v = t; v != s; v = prevv[v]) {
                edge &e = graph[prevv[v]][preve[v]];
                e.cap -= addflow;
                graph[v][e.rev].cap += addflow;
            }
        }
        return ret;
    }

    void output() {
        for (int i = 0; i < graph.size(); i++) {
            for (auto &e : graph[i]) {
                if (e.isrev) continue;
                auto &rev_e = graph[e.to][e.rev];
                cout << i << "->" << e.to << " (flow: " << rev_e.cap << "/" << rev_e.cap +
                e.cap << ")" << endl;
            }
        }
    }
};
```

14 Others

14.1 Arbitrary Modulo Convolution

```
// thanks math314!
typedef long long ll;
typedef pair<int, int> Pii;

#define FOR(i, n) for(int i = 0; i < (n); i++)
#define sz(c) ((int)(c).size())

template<class T>
T extgcd(T a, T b, T &x, T &y) {
    for (T u = y = 1, v = x = 0; a;) {
        T q = b / a;
        swap(x -= q * u, u);
        swap(y -= q * v, v);
        swap(b -= q * a, a);
    }
    return b;
}

template<class T>
T mod_inv(T a, T m) {
    T x, y;
    extgcd(a, m, x, y);
    return (m + x % m) % m;
}

ll mod_pow(ll a, ll n, ll mod) {
    ll ret = 1;
    ll p = a % mod;
    while (n) {
        if (n & 1)
            ret = ret * p % mod;
        p = p * p % mod;
        n >>= 1;
    }
    return
            ret;
}

template<int mod, int primitive_root>
class NTT {
public:
    int get_mod() const { return mod; }

    void _ntt(vector<ll> &a, int sign) {
        const int n = sz(a);
        assert((n ^ (n & -n)) == 0); //n = 2^k

        const int g = 3; //g is primitive root of mod
        int h = (int) mod_pow(g, (mod - 1) / n, mod); // h^n = 1
        if (sign == -1) h = (int) mod_inv(h, mod); //h = h^-1 % mod

        //bit reverse
        int i = 0;
        for (int j = 1; j < n - 1; ++j) {
            for (int k = n >> 1; k > (i ^= k); k >>= 1);
            if (j < i) swap(a[i], a[j]);
        }
```

```
        for (int m = 1; m < n; m *= 2) {
            const int m2 = 2 * m;
            const ll base = mod_pow(h, n / m2, mod);
            ll w = 1;
            FOR(x, m) {
                for (int s = x; s < n; s += m2) {
                    ll u = a[s];
                    ll d = a[s + m] * w % mod;
                    a[s] = u + d;
                    if (a[s] >= mod) a[s] -= mod;
                    a[s + m] = u - d;
                    if (a[s + m] < 0) a[s + m] += mod;
                }
                w = w * base % mod;
            }
        }

        for (auto &x : a) if (x < 0) x += mod;
    }

    void ntt(vector<ll> &input) {
        _ntt(input, 1);
    }

    void intt(vector<ll> &input) {
        _ntt(input, -1);
        const int n_inv = mod_inv(sz(input), mod);
        for (auto &x : input) x = x * n_inv % mod;
    }

    // みみ演算を行う
    vector<ll> convolution(const vector<ll> &a, const vector<ll> &b) {
        int ntt_size = 1;
        while (ntt_size < sz(a) + sz(b)) ntt_size *= 2;

        vector<ll> _a = a, _b = b;
        _a.resize(ntt_size);
        _b.resize(ntt_size);

        ntt(_a);
        ntt(_b);

        FOR(i, ntt_size) {
            (_a[i] *= _b[i]) %= mod;
        }

        intt(_a);
        return _a;
    }
};

ll garner(vector<Pii> mr, int mod) {
    mr.emplace_back(mod, 0);

    vector<ll> coffs(sz(mr), 1);
    vector<ll> constants(sz(mr), 0);
    FOR(i, sz(mr) - 1) {
        // coffs[i] * v + constants[i] == mr[i].second (mod mr[i].first) を解く
        ll v = (mr[i].second - constants[i]) * mod_inv<ll>(coffs[i], mr[i].first) %
        mr[i].first;
        if (v < 0) v += mr[i].first;

        for (int j = i + 1; j < sz(mr); j++) {
            (constants[j] += coffs[j] * v) %= mr[j].first;
            (coffs[j] *= mr[i].first) %= mr[j].first;
        }
    }

    return constants[sz(mr) - 1];
}

typedef NTT<167772161, 3> NTT_1;
typedef NTT<469762049, 3> NTT_2;
typedef NTT<1224736769, 3> NTT_3;

//任意のmodでみみ演算 O(n log n)
vector<ll> int32mod_convolution(vector<ll> a, vector<ll> b, int mod) {
    for (auto &x : a) x %= mod;
    for (auto &x : b) x %= mod;
    NTT_1 ntt1;
    NTT_2 ntt2;
    NTT_3 ntt3;
    auto x = ntt1.convolution(a, b);
    auto y = ntt2.convolution(a, b);
    auto z = ntt3.convolution(a, b);

    vector<ll> ret(sz(x));
    vector<Pii> mr(3);
    FOR(i, sz(x)) {
        mr[0].first = ntt1.get_mod(), mr[0].second = (int) x[i];
        mr[1].first = ntt2.get_mod(), mr[1].second = (int) y[i];
        mr[2].first = ntt3.get_mod(), mr[2].second = (int) z[i];
        ret[i] = garner(mr, mod);
    }

    return ret;
}

// garnerのアルゴリズムを直書きしたversion 速い
vector<ll> fast_int32mod_convolution(vector<ll> a, vector<ll> b, int mod) {
    for (auto &x : a) x %= mod;
    for (auto &x : b) x %= mod;

    NTT_1 ntt1;
    NTT_2 ntt2;
    NTT_3 ntt3;
    assert(ntt1.get_mod() < ntt2.get_mod() && ntt2.get_mod() < ntt3.get_mod());
    auto x = ntt1.convolution(a, b);
    auto y = ntt2.convolution(a, b);
    auto z = ntt3.convolution(a, b);

    // garnerのアルゴリズムを極力高速化した
```

```
    const ll m1 = ntt1.get_mod(), m2 = ntt2.get_mod(), m3 = ntt3.get_mod();
    const ll m1_inv_m2 = mod_inv<ll>(m1, m2);
    const ll m12_inv_m3 = mod_inv<ll>(m1 * m2, m3);
    const ll m12_mod = m1 * m2 % mod;
    vector<ll> ret(sz(x));
    FOR(i, sz(x)) {
        ll v1 = (y[i] - x[i]) * m1_inv_m2 % m2;
        if (v1 < 0) v1 += m2;
        ll v2 = (z[i] - (x[i] + m1 * v1) % m3) * m12_inv_m3 % m3;
        if (v2 < 0) v2 += m3;
        ll constants3 = (x[i] + m1 * v1 + m12_mod * v2) % mod;
        if (constants3 < 0) constants3 += mod;
        ret[i] = constants3;
    }

    return ret;
}
```

14.2 Wavelet Matrix

```
// thanks beet-aizu!
struct FullyIndexableDictionary {
    int len, blk;
    vector<unsigned> bit;
    vector<int> sum;

    FullyIndexableDictionary() {}

    FullyIndexableDictionary(int len)
            : len(len), blk((len + 31) >> 5), bit(blk, 0), sum(blk, 0) {}

    void set(int k) {
        bit[k >> 5] |= 1u << (k & 31);
    }

    void build() {
        sum[0] = 0;
        for (int i = 1; i < blk; i++)
            sum[i] = sum[i - 1] + __builtin_popcount(bit[i - 1]);
    }

    bool operator[](int k) const {
        return bool((bit[k >> 5] >> (k & 31)) & 1);
    }

    int rank(int k) {
        return sum[k >> 5] + __builtin_popcount(bit[k >> 5] & ((1u << (k & 31)) - 1));
    }

    int rank(bool v, int k) {
        return (v ? rank(k) : k - rank(k));
    }

    int select(bool v, int k) {
        if (k < 0 || rank(v, len) <= k) return -1;
        int l = 0, r = len;
        while (l + 1 < r) {
            int m = (l + r) >> 1;
            if (rank(v, m) >= k + 1) r = m;
            else l = m;
        }
        return r - 1;
    }

    int select(bool v, int i, int l) {
        return select(v, i + rank(v, l));
    }
};

template<class T, int MAXLOG>
struct WaveletMatrix {
    int len;
    FullyIndexableDictionary mat[MAXLOG];
    int zs[MAXLOG], buff1[MAXLOG], buff2[MAXLOG];
    static const T npos = -1;

    int freq_dfs(int d, int l, int r, T val, T a, T b) {
        if (l == r) return 0;
        if (d == MAXLOG) return (a <= val && val < b) ? r - l : 0;
        T nv = T(1) << (MAXLOG - d - 1) | val;
        T nnv = ((T(1) << (MAXLOG - d - 1)) - 1) | nv;
        if (nnv < a || b <= val) return 0;
        if (a <= val && nnv < b) return r - l;
        int lc = mat[d].rank(1, l), rc = mat[d].rank(1, r);
        return freq_dfs(d + 1, l - lc, r - rc, val, a, b)
               + freq_dfs(d + 1, lc + zs[d], rc + zs[d], nv, a, b);
    }

    WaveletMatrix(vector<T> data) {
        len = data.size();
        vector<T> l(len), r(len);
        for (int dep = 0; dep < MAXLOG; dep++) {
            mat[dep] = FullyIndexableDictionary(len + 1);
            int p = 0, q = 0;
            for (int i = 0; i < len; i++) {
                bool k = (data[i] >> (MAXLOG - (dep + 1))) & 1;
                if (k) r[q++] = data[i], mat[dep].set(i);
                else l[p++] = data[i];
            }
            zs[dep] = p;
            mat[dep].build();
            swap(l, data);
            for (int i = 0; i < q; i++) data[p + i] = r[i];
        }
    }

    T access(int k) {
        T res = 0;
        for (int dep = 0; dep < MAXLOG; dep++) {
            bool bit = mat[dep][k];
            res = (res << 1) | bit;
            k = mat[dep].rank(bit, k) + zs[dep] * dep;
        }
        return res;
```

```
}

// return the number of v in [0,k)
int rank(T v, int k) {
    int l = 0, r = k;
    for (int dep = 0; dep < MAXLOG; dep++) {
        buff1[dep] = l;
        buff2[dep] = r;
        bool bit = (v >> (MAXLOG - (dep + 1))) & 1;
        l = mat[dep].rank(bit, l) + zs[dep] * bit;
        r = mat[dep].rank(bit, r) + zs[dep] * bit;
    }
    return r - l;
}

// return the position of k-th v
int select(T v, int k) {
    rank(v, len);
    for (int dep = MAXLOG - 1; dep >= 0; dep--) {
        bool bit = (v >> (MAXLOG - (dep + 1))) & 1;
        k = mat[dep].select(bit, k, buff1[dep]);
        if (k >= buff2[dep] || k < 0) return -1;
        k -= buff1[dep];
    }
    return k;
}

int select(T v, int k, int l) {
    return select(v, k + rank(v, l));
}

// return k-th largest value in [l,r)
T quantile(int l, int r, int k) {
    if (r - l <= k || k < 0) return -1;
    T res = 0;
    for (int dep = 0; dep < MAXLOG; dep++) {
        int p = mat[dep].rank(1, l);
        int q = mat[dep].rank(1, r);
        if (q - p > k) {
            l = p + zs[dep];
            r = q + zs[dep];
            res |= T(1) << (MAXLOG - (dep + 1));
        } else {
            k -= (q - p);
            l -= p;
            r -= q;
        }
    }
    return res;
}

T rquantile(int l, int r, int k) {
    return quantile(l, r, r - l - k - 1);
}

// return number of points in [left, right) * [lower, upper)
    int rangefreq(int left, int right, T lower, T upper) {
        return freq_dfs(0, left, right, 0, lower, upper);
    }

    pair<int, int> ll(int l, int r, T v) {
        int res = 0;
        for (int dep = 0; dep < MAXLOG; dep++) {
            buff1[dep] = l;
            buff2[dep] = r;
            bool bit = (v >> (MAXLOG - (dep + 1))) & 1;
            if (bit) res += r - l + mat[dep].rank(bit, l) - mat[dep].rank(bit, r);
            l = mat[dep].rank(bit, l) + zs[dep] * bit;
            r = mat[dep].rank(bit, r) + zs[dep] * bit;
        }
        return make_pair(res, r - l);
    }

    int lt(int l, int r, T v) {
        auto p = ll(l, r, v);
        return p.first;
    }

    int le(int l, int r, T v) {
        auto p = ll(l, r, v);
        return p.first + p.second;
    }

    T succ(int l, int r, T v) {
        int k = le(l, r, v);
        return k == r - l ? npos : rquantile(l, r, k);
    }

    T pred(int l, int r, T v) {
        int k = lt(l, r, v);
        return k ? rquantile(l, r, k - 1) : npos;
    }
};
```

14.3 Priority Sum

```
// thanks beet-aizu!
// return sum of top K element (default: maximum)
template<typename T, T identity, typename V=vector<T>,
        typename C1=less<T>, typename C2=greater<T> >
struct PrioritySum {
    size_t num;
    T sum;
    priority_queue<T, V, C1> pq1;
    priority_queue<T, V, C2> pq2;

    PrioritySum() : num(0), sum(identity) {}

    PrioritySum(size_t num) : num(num), sum(identity) {}

    void resolve() {
        assert(size() >= num);
        while (pq2.size() < num) {
            sum += pq1.top();
```

```
            pq2.emplace(pq1.top());
            pq1.pop();
        }
        while (pq2.size() > num) {
            sum -= pq2.top();
            pq1.emplace(pq2.top());
            pq2.pop();
        }
        if (pq1.empty() || pq2.empty()) return;
        while (C2()(pq1.top(), pq2.top())) {
            T t1 = pq1.top();
            pq1.pop();
            T t2 = pq2.top();
            pq2.pop();
            sum += t1;
            sum -= t2;
            pq1.emplace(t2);
            pq2.emplace(t1);
        }
    }

    T query() {
        resolve();
        return sum;
    }

    void push(const T &x) { pq1.emplace(x); }

    void expand() { num++; }

    void shrink() {
        assert(num);
        num--;
    }

    size_t size() const { return pq1.size() + pq2.size(); }
};

template<typename T>
using MaximumSum=PrioritySum<T, T(0), vector<T>, less<T>, greater<T>>;
template<typename T>
using MinimumSum=PrioritySum<T, T(0), vector<T>, greater<T>, less<T>>;
```

14.4 Modulo Gauss Jordan Elimination

```
// thanks drken!
// 逆元計算
long long modinv(long long a, long long mod) {
    long long b = mod, u = 1, v = 0;
    while (b) {
        long long t = a / b;
        a -= t * b;
        swap(a, b);
        u -= t * v;
        swap(u, v);
    }
    u %= mod;
    if (u < 0) u += mod;
    return u;
}

// matrix
template<int MOD>
struct Matrix {
    vector<vector<long long> > val;

    Matrix(int n, int m, long long x = 0) : val(n, vector<long long>(m, x)) {}

    void init(int n, int m, long long x = 0) { val.assign(n, vector<long long>(m, x)); }

    size_t size() const { return val.size(); }

    inline vector<long long> &operator[](int i) { return val[i]; }
};

template<int MOD>
int GaussJordan(Matrix<MOD> &A, bool is_extended = false) {
    int m = A.size(), n = A[0].size();
    for (int row = 0; row < m; ++row)
        for (int col = 0; col < n; ++col)
            A[row][col] = (A[row][col] % MOD + MOD) % MOD;

    int rank = 0;
    for (int col = 0; col < n; ++col) {
        if (is_extended && col == n - 1) break;
        int pivot = -1;
        for (int row = rank; row < m; ++row) {
            if (A[row][col] != 0) {
                pivot = row;
                break;
            }
        }
        if (pivot == -1) continue;
        swap(A[pivot], A[rank]);
        auto inv = modinv(A[rank][col], MOD);
        for (int col2 = 0; col2 < n; ++col2)
            A[rank][col2] = A[rank][col2] * inv % MOD;
        for (int row = 0; row < m; ++row) {
            if (row != rank && A[row][col]) {
                auto fac = A[row][col];
                for (int col2 = 0; col2 < n; ++col2) {
                    A[row][col2] -= A[rank][col2] * fac % MOD;
                    if (A[row][col2] < 0) A[row][col2] += MOD;
                }
            }
        }
        ++rank;
    }
    return rank;
}

template<int MOD>
int linear_equation(Matrix<MOD> A, vector<long long> b, vector<long long> &res) {
    int m = A.size(), n = A[0].size();
```

```
    Matrix<MOD> M(m, n + 1);
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) M[i][j] = A[i][j];
        M[i][n] = b[i];
    }
    int rank = GaussJordan(M, true);

    // check if it has no solution
    for (int row = rank; row < m; ++row) if (M[row][n]) return -1;

    // answer
    res.assign(n, 0);
    for (int i = 0; i < rank; ++i) res[i] = M[i][n];
    return rank;
}
```

15 Settings

15.1 CMakeLists.txt

```
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -O2 -D_GLIBCXX_DEBUG -fsanitize=signed-integer-overflow
-fno-sanitize-recover=all")
add_executable(a a.cpp)
add_executable(b b.cpp)
add_executable(c c.cpp)
add_executable(d d.cpp)
add_executable(e e.cpp)
add_executable(f f.cpp)
add_executable(g g.cpp)
add_executable(h h.cpp)
add_executable(i i.cpp)
add_executable(j j.cpp)
add_executable(k k.cpp)
```

15.2 Print PDF

```
~$ for d in {A..H}
> do
> google-chrome --headless --disable-gpu --print-to-pdf http://********/${d}_ja.html
> lpr output.pdf
> done
```