

# Team Notebook

greedyLazy

September 1, 2024

## Contents

<b>1 DP</b>	<b>2</b>	2.11 Trie . . . . .	6	<b>5 Maths</b>	<b>10</b>
1.1 Divide and Conquer . . . . .	2	2.12 Wavelet . . . . .	6	5.1 Matrix . . . . .	10
<b>2 Data Structures</b>	<b>2</b>	<b>3 Geometry</b>	<b>7</b>	5.2 Mod Int . . . . .	10
2.1 Centroid Decomposition . . . . .	2	3.1 Basic . . . . .	7	5.3 Precal Modulo Inverse . . . . .	10
2.2 DSU Tree . . . . .	2	3.2 ConvexHull . . . . .	7	5.4 Rabin-Miller . . . . .	10
2.3 DSU . . . . .	3	3.3 LineContainer . . . . .	7	<b>6 Miscellaneous</b>	<b>11</b>
2.4 Fenwick . . . . .	3	<b>4 Graphs</b>	<b>8</b>	6.1 Clion . . . . .	11
2.5 HLD . . . . .	3	4.1 Articular . . . . .	8	6.2 Debug . . . . .	11
2.6 Link-Cut Tree . . . . .	4	4.2 Bellman-Ford . . . . .	8	6.3 mt19937 . . . . .	11
2.7 Ordered Set . . . . .	5	4.3 Bridge . . . . .	8	6.4 time . . . . .	11
2.8 Persistent Segment Tree . . . . .	5	4.4 Kosaraju . . . . .	8	<b>7 Strings</b>	<b>11</b>
2.9 RMQ . . . . .	5	4.5 Max Flow . . . . .	9	7.1 Hash . . . . .	11
2.10 Treap . . . . .	5	4.6 TwoSatSolver . . . . .	9	7.2 KMP . . . . .	11
				7.3 Z Function . . . . .	11

## 1 DP

### 1.1 Divide and Conquer

```
int m, n;
vector<long long> dp_before(n), dp_cur(n);

long long C(int i, int j);

// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr) {
    if (l > r)
        return;

    int mid = (l + r) >> 1;
    pair<long long, int> best = {LLONG_MAX, -1};

    for (int k = optl; k <= min(mid, optr); k++) {
        best = min(best, {(k ? dp_before[k - 1] : 0) + C(k, mid), k});
    }

    dp_cur[mid] = best.first;
    int opt = best.second;

    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}

int solve() {
    for (int i = 0; i < n; i++)
        dp_before[i] = C(0, i);

    for (int i = 1; i < m; i++) {
        compute(0, n - 1, 0, n - 1);
        dp_before = dp_cur;
    }

    return dp_before[n - 1];
}
```

## 2 Data Structures

### 2.1 Centroid Decomposition

```
vector<vector<int>>> adj;
vector<bool> is_removed;
```

```
vector<int> subtree_size;

/** DFS to calculate the size of the subtree rooted at 'node'
 */
int get_subtree_size(int node, int parent = -1) {
    subtree_size[node] = 1;
    for (int child : adj[node]) {
        if (child == parent || is_removed[child]) { continue; }
        subtree_size[node] += get_subtree_size(child, node);
    }
    return subtree_size[node];
}

/**
 * Returns a centroid (a tree may have two centroids) of the
 * subtree
 * containing node 'node' after node removals
 * @param node current node
 * @param tree_size size of current subtree after node
 * removals
 * @param parent parent of u
 * @return first centroid found
 */
int get_centroid(int node, int tree_size, int parent = -1) {
    for (int child : adj[node]) {
        if (child == parent || is_removed[child]) { continue; }
        if (subtree_size[child] * 2 > tree_size) {
            return get_centroid(child, tree_size, node);
        }
    }
    return node;
}

/** Build up the centroid decomposition recursively */
void build_centroid_decomp(int node = 0) {
    int centroid = get_centroid(node, get_subtree_size(node));

    // do something

    is_removed[centroid] = true;

    for (int child : adj[centroid]) {
        if (is_removed[child]) { continue; }
        build_centroid_decomp(child);
    }
}
```

### 2.2 DSU Tree

```
struct DSUTree {
    vector<int> par, in, out, val, d;
    vector<vector<int>>> adj;
    vector<vector<int>>> up, mx;
    int num, n;
    int cur = 0;
    int find(int v) {
        while (v != par[v]) v = par[v] = par[par[v]];
        return v;
    }
    int newnode() {
        num++;
        par.push_back({});
        adj.push_back({});
        in.push_back({});
        out.push_back({});
        val.push_back({});
        d.push_back({});
        par[num] = num;
        return num;
    }
    bool join(int u, int v, int i) {
        u = find(u);
        v = find(v);
        if (u == v) return false;
        int p = newnode();
        val[p] = i;
        par[u] = par[v] = p;
        adj[p].push_back(u);
        adj[p].push_back(v);
        return true;
    }
    void init(int n) {
        num = this->n = n;
        adj.resize(n + 1);
        par.resize(n + 1);
        in.resize(n + 1);
        out.resize(n + 1);
        val.resize(n + 1);
        d.resize(n + 1);
        iota(all(par), 0);
    }
    void dfs(int u) {
        in[u] = ++cur;
        for (int v : adj[u]) {
            d[v] = d[u] + 1;
            up[v][0] = u;
            mx[v][0] = max(val[v], val[u]);
        }
    }
}
```

```

    dfs(v);
}
out[u] = cur;
}
void work() {
    up = mx = vector<vector<int>>>(num + 1, vector<int>(18));
    for (int i = 1; i <= num; i++) {
        if (i == find(i)) {
            up[i][0] = i;
            mx[i][0] = val[i];
            dfs(i);
        }
    }
    for (int i = 1; i <= 18; i++) {
        for (int u = 1; u <= num; u++) {
            up[u][i] = up[up[u][i-1]][i-1];
            mx[u][i] = max(mx[u][i-1], mx[up[u][i-1]][i-1]);
        }
    }
}
pair<int, int> LCA(int u, int v) {
    int ans = max(val[u], val[v]);
    if (d[u] < d[v]) swap(u, v);
    int det = d[u] - d[v];
    for (int i = 18; i >= 0; i--) {
        if ((det >> i) & 1) {
            ans = max(ans, mx[u][i]);
            u = up[u][i];
        }
    }
    if (u == v) {
        return {u, ans};
    }
    for (int i = 18; i >= 0; i--) {
        if (up[u][i] != up[v][i]) {
            ans = max({ans, mx[u][i], mx[v][i]});
            u = up[u][i];
            v = up[v][i];
        }
    }
    return {up[v][0], max({ans, up[u][0], up[v][0]})};
}
};

```

## 2.3 DSU

```

struct DSU {
    int n;
    vector<int> par, sz;

```

```

DSU(int n): n(n) {
    par = sz = vector<int>(n + 1);
    fill(all(sz), 1LL);
    iota(all(par), 0LL);
}

int find(int v) {
    while (v != par[v]) v = par[v] = par[par[v]];
    return v;
}

bool join(int u, int v) {
    u = find(u);
    v = find(v);
    if (u == v) return false;
    if (sz[u] < sz[v]) swap(u, v);
    sz[u] += sz[v];
    par[v] = u;
    return true;
}

int same(int u, int v) {
    return find(u) == find(v);
}
};

```

## 2.4 Fenwick

```

struct fenwick {
    vector<int> f;
    int n;
    fenwick(int n): n(n) {
        f.resize(n + 1);
    }
    void add(int i, int v) {
        for (; i <= n; i += i & -i) f[i] += v;
    }
    int get(int i) {
        int ans = 0;
        for (; i >= 1; i -= i & -i) ans += f[i];
        return ans;
    }
    int range(int u, int v) {
        return get(v) - get(u-1);
    }
};

```

## 2.5 HLD

```

struct HLD {
    int n;
    int root;
    vector<int> sz, par, dep, top, in, out, seq;
    vector<vector<int>>> adj;
    int curdfs = 0;
    HLD(int n): n(n) {
        par = sz = dep = top = in = out = seq = vector<int>(n + 1);
        adj.resize(n + 1);
    }

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

    void start(int root = 1) {
        this->root = root;
        par[root] = root;
        top[root] = root;
        dfssz(root);
        dfshld(root);
    }

    void dfssz(int u) {
        if (u != root) {
            adj[u].erase(find(all(adj[u]), par[u]));
        }
        sz[u] = 1;
        for (int &v : adj[u]) {
            dep[v] = dep[u] + 1;
            par[v] = u;
            dfssz(v);
            sz[u] += sz[v];
            if (sz[v] > sz[adj[u][0]]) {
                swap(adj[u][0], v);
            }
        }
    }

    void dfshld(int u) {
        in[u] = ++curdfs;
        seq[in[u]] = u;
        for (int v : adj[u]) {
            top[v] = (v == adj[u][0] ? top[u] : v);
            dfshld(v);
        }
    }
};

```

```

    }
    out[u] = curdfs;
}

int lca(int u, int v) {
    while (top[u] != top[v]) {
        if (dep[top[u]] > dep[top[v]]) {
            u = par[top[u]];
        }
        else {
            v = par[top[v]];
        }
    }
    return (dep[u] < dep[v] ? u : v);
}

bool isancestor(int u, int v) {
    return in[u] <= in[v] && out[v] <= out[u];
}
};

```

## 2.6 Link-Cut Tree

```

#include <bits/stdc++.h>
using namespace std;

```

```

// BeginCodeSnip{Link Cut Tree}

```

```

struct Node {
    int x;
    Node *l = 0;
    Node *r = 0;
    Node *p = 0;
    bool rev = false;

    Node() = default;

    Node(int v) { x = v; }

    void push() {
        if (rev) {
            rev = false;
            swap(l, r);
            if (l) l->rev ^= true;
            if (r) r->rev ^= true;
        }
    }
}

```

```

bool is_root() { return p == 0 || (p->l != this && this != p->r); }

```

```

};

struct LCT {
    vector<Node> a;

    LCT(int n) {
        a.resize(n + 1);
        for (int i = 1; i <= n; ++i) a[i].x = i;
    }

    void rot(Node *c) {
        auto p = c->p;
        auto g = p->p;

        if (!p->is_root()) (g->r == p ? g->r : g->l) = c;

        p->push();
        c->push();

        if (p->l == c) { // rtr
            p->l = c->r;
            c->r = p;
            if (p->l) p->l->p = p;
        } else { // rtl
            p->r = c->l;
            c->l = p;
            if (p->r) p->r->p = p;
        }

        p->p = c;
        c->p = g;
    }

    void splay(Node *c) {
        while (!c->is_root()) {
            auto p = c->p;
            auto g = p->p;
            if (!p->is_root()) rot((g->r == p) == (p->r == c) ? p : c);
            rot(c);
        }
        c->push();
    }

    Node *access(int v) {
        Node *last = 0;
        Node *c = &a[v];
        for (Node *p = c; p; p = p->p) {
            splay(p);
            p->r = last;
        }
    }
}

```

```

    last = p;
}

splay(c);
return last;
}

void make_root(int v) {
    access(v);
    auto *c = &a[v];
    if (c->l) c->l->rev ^= true, c->l = 0;
}

void link(int u, int v) {
    make_root(v);
    Node *c = &a[v];
    c->p = &a[u];
}

void cut(int u, int v) {
    make_root(u);
    access(v);
    if (a[v].l) {
        a[v].l->p = 0;
        a[v].l = 0;
    }
}

bool connected(int u, int v) {
    access(u);
    access(v);
    return a[u].p;
}
};

// EndCodeSnip

int N, M;

int main() {
    cin.tie(0)->sync_with_stdio(0);

    cin >> N >> M;

    LCT lc(N);

    while (M--) {
        string a;
        cin >> a;
        int b, c;
        cin >> b >> c;
        if (a == "add") { lc.link(b, c); }
    }
}

```

```

if (a == "rem") { lc.cut(b, c); }

if (a == "conn") {
    cout << (lc.connected(b, c) ? "YES" : "NO") << "\n";
}
}
}

```

## 2.7 Ordered Set

```

#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;
template <class type1>
using ordered_set = tree<type1, null_type, less_equal<type1>,
    rb_tree_tag, tree_order_statistics_node_update>;

```

## 2.8 Persistent Segment Tree

```

struct Node {
    int val;
    Node *lef, *rig;
    Node (int val) {
        this->val = val;
        lef = rig = nullptr;
    }
    Node (Node *lef, Node *rig) {
        this->lef = lef;
        this->rig = rig;
    }
    Node (Node *p) {
        this->lef = p->lef;
        this->rig = p->rig;
        this->val = p->val;
    }
};

struct PSegtree {
    int n, nv;
    vector<Node*> rt;
    PSegtree() {}
    PSegtree(int n, int nv, vector<int> vec) {
        this->n = n;

```

```

        this->nv = nv;
        rt.resize(nv + 1);
        rt[0] = build(1, n, vec);
    }

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

    int get(int i, Node *p, int l, int r) {
        if (l == r) {
            return p->val;
        }
        int mi = (l + r) >> 1;
        if (i <= mi) return get(i, p->lef, l, mi);
        else return get(i, p->rig, mi+1, r);
    }

    int get(int i, int v) {
        return get(i, rt[v], 1, n);
    }

    Node *update(int i, int val, Node *p, int l, int r) {
        if (l == r) {
            return new Node(val);
        }
        int mi = (l + r) >> 1;
        if (i <= mi) return new Node(update(i, val, p
            ->lef, l, mi), p->rig);
        else return new Node(p->lef, update(i, val, p
            ->rig, mi+1, r));
    }

    void update(int i, int val, int v) {
        rt[v] = update(i, val, rt[v], 1, n);
    }

    void copy(int t1, int t2) {
        rt[t2] = rt[t1];
    }
};

```

## 2.9 RMQ

```

struct RMQ {
    int n, lg;

```

```

    vector<int> v;
    vector<vector<int>> st;
    RMQ(vector<int> &v): v(v), n(v.size() - 1) {
        lg = __lg(n) + 1;
        st.resize(n + 1, vector<int> (lg + 1));
        for (int i = 1; i <= n; i++) {
            st[i][0] = v[i];
        }
        for (int i = 1; i <= lg; i++) {
            for (int j = 1; j + (1LL << i) - 1 <= n
                ; j++) {
                st[j][i] = min(st[j][i - 1], st
                    [j + (1LL << (i - 1))][i -
                        1]);
            }
        }

        int get(int l, int r) {
            int len = __lg(r - l + 1);
            return min(st[l][len], st[r - (1 << len) + 1][
                len]);
        }
    };

```

## 2.10 Treap

```

struct Treap {
    Treap *lef, *rig;
    int num;
    int prio;
    int sz;
    Treap(int num):
        num(num),
        prio(rnd()),
        sz(1),
        lef(nullptr),
        rig(nullptr) {}
};

int sz(Treap *p) {
    return (p ? p->sz : 0);
}

void pull(Treap *p) {
    if (!p) return;
    p->sz = sz(p->lef) + 1 + sz(p->rig);
}

```

```

void merge(Treap *&p, Treap *lef, Treap *rig) {
    if (!lef || !rig) {
        p = (lef ? lef : rig);
        return;
    }
    if (lef->prio > rig->prio) {
        merge(lef->rig, lef->rig, rig);
        p = lef;
    }
    else {
        merge(rig->lef, lef, rig->lef);
        p = rig;
    }
    pull(p);
}

void split(Treap *p, Treap *&lef, Treap *&rig, int k) {
    if (!p) {
        lef = rig = nullptr;
        return;
    }
    if (sz(p->lef) < k) {
        split(p->rig, p->rig, rig, k - 1 - sz(p->lef));
        lef = p;
    }
    else {
        split(p->lef, lef, p->lef, k);
        rig = p;
    }
    pull(p);
}

```

## 2.11 Trie

```

struct trie {
    trie* ch[2];
    int sum, cnt, end;
    trie() {
        ch[0] = ch[1] = nullptr;
        sum = cnt = end = 0;
    }
};

int MXVAL = 0; // 2^(MX + 1) - 1

trie *rt = new trie();
void add(int num) {
    auto p = rt;
    for (int i = 30; i >= 0; i--) {

```

```

        int b = (num >> i) & 1LL;
        if (p->ch[b] == nullptr) p->ch[b] = new trie();
        p = p->ch[b];
        p->cnt++;
        p->sum += num;
    }
    p->end++;
}

int find(int num) {
    auto p = rt;
    for (int i = 30; i >= 0; i--) {
        int b = (num >> i) & 1LL;
        if (p->ch[b] == nullptr) return false;
        p = p->ch[b];
    }
    return p->end;
}

bool del(trie *p, int num, int i, int d) {
    if (i >= 0) {
        int b = (num >> i) & 1LL;
        if (del(p->ch[b], num, i-1, d)) p->ch[b] = nullptr;
    }
    else {
        p->end -= d;
    }
    if (p != rt) {
        p->cnt -= d;
        p->sum -= num * d;
        if (p->cnt == 0) {
            delete(p);
            return true;
        }
    }
    return false;
}

void del(int num, int t) {
    int d = find(num);
    if (d == 0) return;
    if (t == 0) del(rt, num, 30, 1);
    else del(rt, num, 30, d);
}

int kth(int k) {
    auto p = rt;
    int res = 0;
    for (int i = 30; i >= 0; i--) {
        for (int b = 0; b < 2; b++) {
            if (p->ch[b]) {
                if (p->ch[b]->cnt >= k) {
                    p = p->ch[b];
                    res += (b << i);

```

```

                    break;
                }
            }
            else {
                k -= p->ch[b]->cnt;
            }
        }
    }
    return res;
}

int sum(int u, int v, trie *p, int l, int r) {
    if (u > r || l > v || p == nullptr) {
        return 0;
    }
    if (u <= l && r <= v) {
        return p->cnt;
    }
    int mi = (l + r) / 2;
    return sum(u, v, p->ch[0], l, mi) + sum(u, v, p->ch[1], mi + 1, r);
}

int sum(int l, int r) {
    return sum(l, r, rt, 0, MXVAL);
}

int xr(int num) {
    auto p = rt;
    int res = 0;
    for (int i = 30; i >= 0; i--) {
        int b = (num >> i) & 1LL;
        if (p->ch[1 ^ b] != nullptr) {
            p = p->ch[1 ^ b];
            res |= (1 << i);
        }
        else {
            p = p->ch[b];
        }
    }
    return res;
}

```

## 2.12 Wavelet

```

struct wavelet {
    wavelet *lc, *rc;
    vector<int> pf;
    int lo, hi;

    wavelet(vector<int> a): wavelet(1 + all(a), *min_element(1 + all(a)), *max_element(1 + all(a))){}

```

```

wavelet(vector<int>::iterator l, vector<int>::iterator r,
        int lo, int hi): lo(lo), hi(hi) {
    if (lo == hi || l >= r) return;
    int mi = lo + (hi - lo)/2;
    pf.reserve(r - l + 1);
    pf.push_back(OLL);
    for (auto it = l; it != r; it++) {
        pf.push_back(pf.back() + (*it <= mi));
    }
    auto m = stable_partition(l, r, [&](int num) { return num
        <= mi; });
    lc = new wavelet(l, m, lo, mi);
    rc = new wavelet(m, r, mi+1, hi);
}
// tim so nho thu k trong doan (l, r);
int kth(int l, int r, int k) {
    if (l > r) return 0;
    if (lo == hi) {
        return lo;
    }
    int lef = pf[r] - pf[l-1];
    if (lef >= k) return lc->kth(pf[l-1] + 1, pf[r], k);
    else return rc->kth(l - pf[l-1], r - pf[r], k - lef);
}
// dem so > k trong doan (l, r)
int count(int l, int r, int num) {
    if (l > r || hi <= num) return OLL;
    if (lo > num) {
        return (r - l + 1);
    }
    return lc->count(pf[l-1] + 1, pf[r], num) + rc->count(l -
        pf[l-1], r - pf[r], num);
}
};

```

## 3 Geometry

### 3.1 Basic

```

struct Point {
    double x, y;
    Point() { x = y = 0.0; }
    Point(double x, double y) : x(x), y(y) {}

    Point operator + (const Point &a) const { return Point(x
        + a.x, y + a.y); }
    Point operator - (const Point &a) const { return Point(x
        - a.x, y - a.y); }
}

```

```

Point operator * (double k) const { return Point(x * k, y
    * k); }
Point operator / (double k) const { return Point(x / k, y
    / k); }
};

struct Line { // Ax + By = C
    double a, b, c;
    Line(double a = 0, double b = 0, double c = 0) : a(a), b(b), c(c) {}
    Line(Point A, Point B) {
        a = B.y - A.y;
        b = A.x - B.x;
        c = a * A.x + b * A.y;
    }
};

Line Perpendicular_Bisector(Point A, Point B) {
    Point M = (A + B) / 2;
    Line d = Line(A, B);
    // the equation of a perpendicular line has the form: -Bx
    + Ay = D
    double D = -d.b * M.x + d.a * M.y;
    return Line(-d.b, d.a, D);
}

```

### 3.2 ConvexHull

```

// Kiu im
struct Point {
    int x, y;
};

// Tch c hng ca AB v AC
long long cross(const Point &A, const Point &B, const Point
    &C) {
    return 1LL * (B.x - A.x) * (C.y - A.y) - 1LL * (C.x - A.x
        ) * (B.y - A.y);
}

// A -> B -> C i theo th t theo chiu kim ng h
(-1), thng hng (0), ngc chiu kim ng h
(1)
int ccw(const Point &A, const Point &B, const Point &C) {
    long long S = cross(A, B, C);
    if (S < 0) return -1;
    if (S == 0) return 0;
}

```

```

return 1;
}

// Tr v bao li vi th t cc im c
lit k ngc chiu kim ng h
vector<Point> convexHull(vector<Point> p, int n) {
    // a im c tung nh nht (v tri nht )
    ln u tp
    for (int i = 1; i < n; ++i) {
        if (p[0].y > p[i].y || (p[0].y == p[i].y && p[0].x >
            p[i].x)) {
            swap(p[0], p[i]);
        }
    }

    // Sp xp cc im I theo gc to bi trc
    honh theo chiu dng v OI
    sort(p.begin() + 1, p.end(), [&p](const Point &A, const
        Point &B) {
        int c = ccw(p[0], A, B);
        if (c > 0) return true;
        if (c < 0) return false;
        return A.x < B.x || (A.x == B.x && A.y < B.y);
    });

    // Tp bao li
    vector<Point> hull;
    hull.push_back(p[0]);

    // Dng bao li
    for (int i = 1; i < n; ++i) {
        while (hull.size() >= 2 && ccw(hull[hull.size() - 2],
            hull.back(), p[i]) < 0) {
            hull.pop_back();
        }
        hull.push_back(p[i]);
    }
    return hull;
}

```

### 3.3 LineContainer

```

/**
 * Author: Simon Lindholm
 * Date: 2017-04-20
 * License: CC0
 * Source: own work
 * Description: Container where you can add lines of the
    form kx+m, and query maximum values at points x.

```

```

* Useful for dynamic programming ('convex hull trick').
* Time:  $O(\log N)$ 
* Status: stress-tested
*/
#pragma once

struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) return x->p = inf, 0;
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};

```

## 4 Graphs

### 4.1 Articular

```

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;

```

```

        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p != -1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if (p == -1 && children > 1)
        IS_CUTPOINT(v);
}

```

### 4.2 Bellman-Ford

```

struct Edge {
    int a, b, cost;
};

int n, m;
vector<Edge> edges;
const int INF = 1000000000;

void solve()
{
    vector<int> d(n);
    vector<int> p(n, -1);
    int x;
    for (int i = 0; i < n; ++i) {
        x = -1;
        for (Edge e : edges) {
            if (d[e.a] + e.cost < d[e.b]) {
                d[e.b] = d[e.a] + e.cost;
                p[e.b] = e.a;
                x = e.b;
            }
        }
    }

    if (x == -1) {
        cout << "No negative cycle found.";
    } else {
        for (int i = 0; i < n; ++i)
            x = p[x];

        vector<int> cycle;
        for (int v = x;; v = p[v]) {
            cycle.push_back(v);

```

```

            if (v == x && cycle.size() > 1)
                break;
        }
        reverse(cycle.begin(), cycle.end());

        cout << "Negative cycle: ";
        for (int v : cycle)
            cout << v << ' ';
        cout << endl;
    }
}

```

### 4.3 Bridge

```

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
}

```

### 4.4 Kosaraju

```

vector<vector<int>>> adj, adj_rev;
vector<bool> used;
vector<int> order, component;

void dfs1(int v) {
    used[v] = true;

    for (auto u : adj[v])
        if (!used[u])
            dfs1(u);

    order.push_back(v);
}

void dfs2(int v) {
    used[v] = true;

```



```

    component.push_back(v);

    for (auto u : adj_rev[v])
        if (!used[u])
            dfs2(u);
}

int main() {
    int n;
    // ... read n ...

    for (;;) {
        int a, b;
        // ... read next directed edge (a,b) ...
        adj[a].push_back(b);
        adj_rev[b].push_back(a);
    }

    used.assign(n, false);

    for (int i = 0; i < n; i++)
        if (!used[i])
            dfs1(i);

    used.assign(n, false);
    reverse(order.begin(), order.end());

    for (auto v : order)
        if (!used[v]) {
            dfs2(v);

            // ... processing next component ...

            component.clear();
        }
}

```

## 4.5 Max Flow

```

int n;
vector<vector<int>> capacity;
vector<vector<int>> adj;

int bfs(int s, int t, vector<int>& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, INF});
}

```

```

while (!q.empty()) {
    int cur = q.front().first;
    int flow = q.front().second;
    q.pop();

    for (int next : adj[cur]) {
        if (parent[next] == -1 && capacity[cur][next]) {
            parent[next] = cur;
            int new_flow = min(flow, capacity[cur][next]);
            if (next == t)
                return new_flow;
            q.push({next, new_flow});
        }
    }
}

return 0;

int maxflow(int s, int t) {
    int flow = 0;
    vector<int> parent(n);
    int new_flow;

    while (new_flow = bfs(s, t, parent)) {
        flow += new_flow;
        int cur = t;
        while (cur != s) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}

```

## 4.6 TwoSatSolver

```

struct TSS {
    int nvar;
    int nvrt;
    vector<vector<int>> adj;
    vector<int> res, scc, topo, vis, in, low, del, idx;

    TSS(int nvar):
        nvar(nvar),
        nvrt(nvar * 2),

```

```

        adj(nvrt + 1),
        res(nvar + 1),
        scc(nvrt + 1),
        in(nvrt + 1),
        low(nvrt + 1),
        del(nvrt + 1),
        idx(nvrt + 1) {}

    int conj(int u) {
        if (u > nvar) return u - nvar;
        return u + nvar;
    }

    void add(int u, bool nu, int v, bool nv) {
        if (nu) u = conj(u);
        if (nv) v = conj(v);
        adj[conj(u)].push_back(v);
        adj[conj(v)].push_back(u);
    }

    int curdfs = 0;
    stack<int> st;
    int curidx = nvrt;
    void tarjan(int u) {
        in[u] = low[u] = ++curdfs;
        st.push(u);
        for (int v : adj[u]) {
            if (del[v]) continue;
            if (!in[v]) {
                tarjan(v);
                low[u] = min(low[u], low[v]);
            }
            else {
                low[u] = min(low[u], in[v]);
            }
        }
        if (low[u] == in[u]) {
            idx[u] = curidx--;
            while (st.top() != u) {
                int v = st.top();
                st.pop();
                del[v] = true;
                scc[v] = u;
            }
            scc[u] = u;
            del[u] = true;
            st.pop();
        }
    }

    bool solve() {

```

```

for (int i = 1; i <= nvrt; i++) {
    if (!in[i]) tarjan(i);
}
for (int i = 1; i <= nvar; i++) {
    if (scc[i] == scc[conj(i)]) return false;
    res[i] = idx[scc[i]] > idx[scc[conj(i)]];
}
return true;
}
};

```

## 5 Maths

### 5.1 Matrix

```

struct Matrix {
    vector<vector<type>>> data;

    // S lng hng ca ma trn
    int row() const { return data.size(); }

    // S lng hng ca ma trn
    int col() const { return data[0].size(); }

    auto & operator [] (int i) { return data[i]; }

    const auto & operator [] (int i) const { return data[i]; }

    Matrix() = default;

    Matrix(int r, int c): data(r, vector<type>(c)) { }

    Matrix(const vector<vector<type>>> &d): data(d) {

        // Kim tra cc hng c cng size khng v size c
        // ln hn 0 hay khng
        // Tuy nhin khng thc s cn thit , ta c
        // th b cc dng /**/ i
        /**/ assert(d.size());
        /**/ int size = d[0].size();
        /**/ assert(size);
        /**/ for (auto x : d) assert(x.size() == size);
    }

    // In ra ma trn .
    friend ostream & operator << (ostream &out, const Matrix
    &d) {
        for (auto x : d.data) {

```

```

        for (auto y : x) out << y << ' ';
        out << '\n';
    }
    return out;
}

// Ma trn n v
static Matrix identity(long long n) {
    Matrix a = Matrix(n, n);
    while (n--) a[n][n] = 1;
    return a;
}

// Nhn ma trn
Matrix operator * (const Matrix &b) {
    Matrix a = *this;

    // Kim tra iu kin nhn ma trn
    assert(a.col() == b.row());

    Matrix c(a.row(), b.col());
    for (int i = 0; i < a.row(); ++i)
        for (int j = 0; j < b.col(); ++j)
            for (int k = 0; k < a.col(); ++k)
                c[i][j] += a[i][k] * b[k][j];
    return c;
}

// Ly tha ma trn
Matrix pow(long long exp) {

    // Kim tra iu kin ly tha ma trn (1 ma
    // trn vung)
    assert(row() == col());

    Matrix base = *this, ans = identity(row());
    for (; exp > 0; exp >>= 1, base = base * base)
        if (exp & 1) ans = ans * base;
    return ans;
}
};

```

### 5.2 Mod Int

```

/**
 * Description: Modular arithmetic. Assumes $MOD$ is prime.
 * Source: KACTL
 * Verification: https://open.kattis.com/problems/modulararithmetic

```

```

 * Usage: mi a = MOD+5; inv(a); // 400000003
 */

template<int MOD, int RT> struct mint {
    static const int mod = MOD;
    static constexpr mint rt() { return RT; } // primitive root
    int v;
    explicit operator int() const { return v; }
    mint():v(0) {}
    mint(ll _v):v(int(_v%MOD)) { v += (v<0)*MOD; }
    mint& operator+=(mint o) {
        if ((v += o.v) >= MOD) v -= MOD;
        return *this;
    }
    mint& operator--=(mint o) {
        if ((v -= o.v) < 0) v += MOD;
        return *this;
    }
    mint& operator*=(mint o) {
        v = int((ll)v*o.v%MOD); return *this;
    }
    friend mint pow(mint a, ll p) { assert(p >= 0);
        return p==0?1:pow(a*a,p/2)*(p&1?a:1); }
    friend mint inv(mint a) { assert(a.v != 0); return pow(a,
        MOD-2); }
    friend mint operator+(mint a, mint b) { return a += b; }
    friend mint operator-(mint a, mint b) { return a -= b; }
    friend mint operator*(mint a, mint b) { return a *= b; }
};
using mi = mint<(int)1e9+7, 5>;
using vmi = V<mi>;

```

### 5.3 Precal Modulo Inverse

```

int n = 10, p = 1000000007;
int inv[n + 1];
inv[1] = 1;
for (int i = 2; i <= n; i++) inv[i] = 1LL * (p - p / i) *
    inv[p % i] % p;

```

### 5.4 Rabin-Miller

```

using u64 = uint64_t;
using u128 = __uint128_t;

```

```

u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)

```

```

        result = (u128)result * base % mod;
        base = (u128)base * base % mod;
        e >= 1;
    }
    return result;
}

bool check_composite(u64 n, u64 a, u64 d, int s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
};

bool MillerRabin(u64 n, int iter=5) { // returns true if n
    is probably prime, else returns false.
    if (n < 4)
        return n == 2 || n == 3;

    int s = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        s++;
    }

    for (int i = 0; i < iter; i++) {
        int a = 2 + rand() % (n - 3);
        if (check_composite(n, a, d, s))
            return false;
    }
    return true;
}

```

## 6 Miscellaneous

### 6.1 Clion

```

set(GCC_COVERAGE_COMPILE_FLAGS "-O2 -Dbinhball")
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${
    GCC_COVERAGE_COMPILE_FLAGS}")

```

## 6.2 Debug

```

#ifndef hvmegy
#define dbg(...) logger(__VA_ARGS__, __VA_ARGS__)
template<typename ...Args>
void logger(string vars, Args&&... values) {
    cerr << "[" << vars << " : ";
    string delim = "";
    (... , (cerr << delim << values, delim = ", "));
    cerr << "]" << '\n';
}
#else
#define dbg(...)
#endif

```

### 6.3 mt19937

```

mt19937_64 rnd(chrono::steady_clock::now().time_since_epoch
    ().count());

```

### 6.4 time

```

cerr << '\n' << clock() * 1000.0 / CLOCKS_PER_SEC << "ms" <<
    '\n';

```

## 7 Strings

### 7.1 Hash

```

int base = 311;
int const MOD = 1000000007;
int hashS[MAX];
int POW[MAX];
int gethash(int i, int j, int hashS[MAX]){
    return (hashS[j] - hashS[i - 1] * POW[j - i + 1] + MODMOD
        ) % MOD;
}
void setHash(){
    string s;
    n = s.length();
    POW[0] = 1;
    s = " " + s;
    t = " " + t;
    for (int i = 1; i <= n; i++)

```

```

    POW[i] = (base POW[i - 1])%MOD;
    for (int i = 1; i <= n; i++){
        hashS[i] = (hashS[i - 1] * base + s[i]) % MOD;
    }
    for (int i = 1; i <= m; i++){
        hashT[i] = (hashT[i - 1] * base + t[i]) % MOD;
    }

```

## 7.2 KMP

```

int next_x[N];
int j = next_x[0] = 0;
n = s.length();
m = t.length();
for (int i = 1; i < n; i++){
    while (j > 0 && s[j] != s[i]){
        j = next_x[j - 1];
    }
    if (s[i] == s[j])
        j++;
    next_x[i] = j;
}
j = 0;
for (int i = 0; i < m; i++){
    while (j > 0 && s[j] != t[i])
        j = next_x[j - 1];
    if (s[j] == t[i])
        j++;
    if (j == n){
        cout << i - n + 2 << " ";
    }
}
}

```

## 7.3 Z Function

```

z[0] = n;
int l = 0, r = 0;
for (int i = 1; i < n; i++){
    if (i > r){
        l = r = i;
        while (r < n && s[r] == s[r-l]) ++r;
        z[i] = r - l;
        r--;
    }
    else{
        int k = i - l;
        if (z[k] < r - i + 1) z[i] = z[k];
    }
}

```

<pre>else{   l = i;   while (r &lt; n &amp;&amp; s[r] == s[r - 1]) r++;</pre>		<pre>z[i]= r - 1; r--;</pre>		<pre>  }</pre>
		}		

---