# Rábalabaxúrias [UFMG]

Bruno Monteiro, Pedro Papa e Rafael Grandsire

## Índice

# 1 Grafos

## 1.1 Centroid decomposition

```
// O(n log(n))

int n;
vector<vector<int> > g(MAX);
int subsize[MAX];
int rem[MAX];
int pai[MAX];

void dfs(int k, int last) {
    subsize[k] = 1;
    for (int i = 0; i < (int) g[k].size(); i++)
        if (g[k][i] != last and !rem[g[k][i]]) {
            dfs(g[k][i], k);
            subsize[k] += subsize[g[k][i]];
        }
}

int centroid(int k, int last, int size) {
    for (int i = 0; i < (int) g[k].size(); i++) {
        int u = g[k][i];
        if (rem[u] or u == last) continue;
        if (subsize[u] > size / 2)
            return centroid(u, k, size);
    }
    // k eh o centroid
    return k;
}

void decomp(int k, int last) {
    dfs(k, k);
```

```
    // acha e tira o centroid
    int c = centroid(k, k, subsize[k]);
    rem[c] = 1;
    pai[c] = last;
    if (k == last) pai[c] = c;

    // decompoe as sub-arvores
    for (int i = 0; i < (int) g[c].size(); i++)
        if (!rem[g[c][i]]) decomp(g[c][i], c);
}

void build() {
    memset(rem, 0, sizeof rem);
    decomp(0, 0);
}
```

## 1.2 Tarjan para Pontes

```
// Computa pontos de articulacao
// e pontes
//
// O(n+m)

int in[MAX];
int low[MAX];
int parent[MAX];
vector<int> g[MAX];

bool is_art[MAX];

void dfs_art(int v, int p, int &d){
    parent[v] = p;
    low[v] = in[v] = d++;
    is_art[v] = false;
    for (int j : g[v]){
        if (j == p) continue;
        if (in[j] == -1){
            dfs_art(j, v, d);

            if (low[j] >= in[v]) is_art[v] = true;
            //if (low[j] >  in[v]) this edge is a bridge
            low[v] = min(low[v], low[j]);
        }
        else low[v] = min(low[v], in[j]);
    }
    if (p == -1){
        is_art[v] = false;
        int k = 0;
        for (int j : g[v])
            k += (parent[j] == v);
        if (k > 1) is_art[v] = true;
    }
}

int d = 0;
memset(in, -1, sizeof in);
dfs_art(1, -1, d);
```

## 1.3 Dominator Tree - Kawakami

```
// Se vira pra usar ai
//
// build - O(n)
// dominates - O(1)

int n;

namespace DTree {
    vector<int> g[MAX];

    // The dominator tree
    vector<int> tree[MAX];
    int dfs_l[MAX], dfs_r[MAX];

    // Auxiliary data
    vector<int> rg[MAX], bucket[MAX];
    int idom[MAX], sdom[MAX], prv[MAX], pre[MAX];
    int ancestor[MAX], label[MAX];
    vector<int> preorder;

    void dfs(int v) {
        static int t = 0;
        pre[v] = ++t;
```

```
        sdom[v] = label[v] = v;
        preorder.push_back(v);
        for (int nxt: g[v]) {
            if (sdom[nxt] == -1) {
                prv[nxt] = v;
                dfs(nxt);
            }
            rg[nxt].push_back(v);
        }
    }
    int eval(int v) {
        if (ancestor[v] == -1) return v;
        if (ancestor[ancestor[v]] == -1) return label[v];
        int u = eval(ancestor[v]);
        if (pre[sdom[u]] < pre[sdom[label[v]]]) label[v] = u;
        ancestor[v] = ancestor[u];
        return label[v];
    }
    void dfs2(int v) {
        static int t = 0;
        dfs_l[v] = t++;
        for (int nxt: tree[v]) dfs2(nxt);
        dfs_r[v] = t++;
    }
    void build(int s) {
        for (int i = 0; i < n; i++) {
            sdom[i] = pre[i] = ancestor[i] = -1;
            rg[i].clear();
            tree[i].clear();
            bucket[i].clear();
        }
        preorder.clear();
        dfs(s);
        if (preorder.size() == 1) return;
        for (int i = int(preorder.size()) - 1; i >= 1; i--) {
            int w = preorder[i];
            for (int v: rg[w]) {
                int u = eval(v);
                if (pre[sdom[u]] < pre[sdom[w]]) sdom[w] =
                    sdom[u];
            }
            bucket[sdom[w]].push_back(w);
            ancestor[w] = prv[w];
            for (int v: bucket[prv[w]]) {
                int u = eval(v);
                idom[v] = (u == v) ? sdom[v] : u;
            }
            bucket[prv[w]].clear();
        }
        for (int i = 1; i < preorder.size(); i++) {
            int w = preorder[i];
            if (idom[w] != sdom[w]) idom[w] = idom[idom[w]];
            tree[idom[w]].push_back(w);
        }
        idom[s] = sdom[s] = -1;
        dfs2(s);
    }

    // Whether every path from s to v passes through u
    bool dominates(int u, int v) {
        if (pre[v] == -1) return 1; // vacuously true
        return dfs_l[u] <= dfs_l[v] && dfs_r[v] <= dfs_r[u];
    }
};
```

## 1.4 Kosaraju

```
// O(n + m)

int n;
vector<vector<int> > g(MAX);
vector<vector<int> > gi(MAX); // grafo invertido
int vis[MAX];
stack<int> S;
int comp[MAX]; // componente conexo de cada vertice

void dfs(int k) {
    vis[k] = 1;
    for (int i = 0; i < (int) g[k].size(); i++)
        if (!vis[g[k][i]]) dfs(g[k][i]);

    S.push(k);
}
```

```
void scc(int k, int c) {
    vis[k] = 1;
    comp[k] = c;
    for (int i = 0; i < (int) gi[k].size(); i++)
        if (!vis[gi[k][i]]) scc(gi[k][i], c);
}

void kosaraju() {
    for (int i = 0; i < n; i++) vis[i] = 0;
    for (int i = 0; i < n; i++) if (!vis[i]) dfs(i);

    for (int i = 0; i < n; i++) vis[i] = 0;
    while (S.size()) {
        int u = S.top();
        S.pop();
        if (!vis[u]) scc(u, u);
    }
}
```

## 1.5 Dijkstra

```
// encontra menor distancia de a
// para todos os vertices
// se ao final do algoritmo d[i] = INF,
// entao a nao alcanca i
//
// O(m log(n))

int n;
vector<vector<int> > g(MAX);
vector<vector<int> > w(MAX); // peso das arestas
int d[MAX];

void dijsktra(int a) {
    for (int i = 0; i < n; i++) d[i] = INF;
    d[a] = 0;
    priority_queue<pair<int, int> > Q;
    Q.push(make_pair(0, a));

    while (Q.size()) {
        int u = Q.top().second, dist = -Q.top().first;
        Q.pop();
        if (dist > d[u]) continue;

        for (int i = 0; i < (int) g[u].size(); i++) {
            int v = g[u][i];
            if (d[v] > d[u] + w[u][i]) {
                d[v] = d[u] + w[u][i];
                Q.push(make_pair(-d[v], v));
            }
        }
    }
}
```

## 1.6 Heavy-Light Decomposition sem Update

```
// query de min do caminho
//
// Complexidades:
// build - O(n)
// query_path - O(log(n))

#define f first
#define s second

namespace hld {
    vector<pair<int, int> > g[MAX];
    int in[MAX], sz[MAX];
    int sobe[MAX], pai[MAX];
    int h[MAX], v[MAX], t;
    int men[MAX], seg[2*MAX];

    void build_hld(int k, int p = -1, int f = 1) {
        v[in[k] = t++] = sobe[k]; sz[k] = 1;
        for (auto& i : g[k]) if (i.f != p) {
            sobe[i.f] = i.s; pai[i.f] = k;
            h[i.f] = (i == g[k][0] ? h[k] : i.f);
            men[i.f] = (i == g[k][0] ? min(men[k], i.s) :
               i.s);
            build_hld(i.f, k, f); sz[k] += sz[i.f];

            if (sz[i.f] > sz[g[k][0].f]) swap(i, g[k][0]);
        }
        if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
```

```
    }
    void build(int root = 0) {
        t = 0;
        build_hld(root);
        for (int i = 0; i < t; i++) seg[i+t] = v[i];
        for (int i = t-1; i; i--) seg[i] = min(seg[2*i],
            seg[2*i+1]);
    }
    int query_path(int a, int b) {
        if (a == b) return INF;
        if (in[a] < in[b]) swap(a, b);

        if (h[a] != h[b]) return min(men[a],
            query_path(pai[h[a]], b));
        int ans = INF, x = in[b]+1+t, y = in[a]+t;
        for (; x <= y; ++x/=2, --y/=2) ans = min({ans,
            seg[x], seg[y]});
        return ans;
    }
};
```

## 1.7 Heavy-Light Decomposition - vertice

```
// SegTree de soma
// query / update de soma dos vertices
//
// Complexidades:
// build - O(n)
// query_path - O(log^2 (n))
// update_path - O(log^2 (n))
// query_subtree - O(log(n))
// update_subtree - O(log(n))

namespace seg {
    ll seg[4*MAX], lazy[4*MAX];
    int n, *v;

    ll build(int p=1, int l=0, int r=n-1) {
        lazy[p] = 0;
        if (l == r) return seg[p] = v[l];
        int m = (l+r)/2;
        return seg[p] = build(2*p, l, m) + build(2*p+1, m+1,
            r);
    }
    void build(int n2, int* v2) {
        n = n2, v = v2;
        build();
    }
    void prop(int p, int l, int r) {
        seg[p] += lazy[p]*(r-l+1);
        if (l != r) lazy[2*p] += lazy[p], lazy[2*p+1] +=
            lazy[p];
        lazy[p] = 0;
    }
    ll query(int a, int b, int p=1, int l=0, int r=n-1) {
        prop(p, l, r);
        if (a <= l and r <= b) return seg[p];
        if (b < l or r < a) return 0;
        int m = (l+r)/2;
        return query(a, b, 2*p, l, m) + query(a, b, 2*p+1,
            m+1, r);
    }
    ll update(int a, int b, int x, int p=1, int l=0, int
        r=n-1) {
        prop(p, l, r);
        if (a <= l and r <= b) {
            lazy[p] += x;
            prop(p, l, r);
            return seg[p];
        }
        if (b < l or r < a) return seg[p];
        int m = (l+r)/2;
        return seg[p] = update(a, b, x, 2*p, l, m) +
            update(a, b, x, 2*p+1, m+1, r);
    }
};

namespace hld {
    vector<int> g[MAX];
    int in[MAX], out[MAX], sz[MAX];
    int peso[MAX], pai[MAX];
    int h[MAX], v[MAX], t;

    void build_hld(int k, int p = -1, int f = 1) {
```

```
    v[in[k] = t++] = peso[k]; sz[k] = 1;
    for (auto& i : g[k]) if (i != p) {
        pai[i] = k;
        h[i] = (i == g[k][0] ? h[k] : i);
        build_hld(i, k, f); sz[k] += sz[i];

        if (sz[i] > sz[g[k][0]]) swap(i, g[k][0]);
    }
    out[k] = t;
    if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
}
void build(int root = 0) {
    t = 0;
    build_hld(root);
    seg::build(t, v);
}
ll query_path(int a, int b) {
    if (a == b) return seg::query(in[a], in[a]);
    if (in[a] < in[b]) swap(a, b);

    if (h[a] == h[b]) return seg::query(in[b], in[a]);
    return seg::query(in[h[a]], in[a]) +
        query_path(pai[h[a]], b);
}
void update_path(int a, int b, int x) {
    if (a == b) return (void)seg::update(in[a], in[a],
        x);
    if (in[a] < in[b]) swap(a, b);

    if (h[a] == h[b]) return (void)seg::update(in[b],
        in[a], x);
    seg::update(in[h[a]], in[a], x);
        update_path(pai[h[a]], b, x);
}
ll query_subtree(int a) {
    if (in[a] == out[a]-1) return seg::query(in[a],
        in[a]);
    return seg::query(in[a], out[a]-1);
}
void update_subtree(int a, int x) {
    if (in[a] == out[a]-1) return
        (void)seg::update(in[a], in[a], x);
        seg::update(in[a], out[a]-1, x);
    }
    int lca(int a, int b) {
        if (in[a] < in[b]) swap(a, b);
        return h[a] == h[b] ? b : lca(pai[h[a]], b);
    }
};
```

## 1.8 LCA com HLD

```
// Assume que um vertice eh ancestral dele mesmo, ou seja,
// se a eh ancestral de b, lca(a, b) = a
// Para buildar pasta chamar build(root)
//
// Complexidades:
// build - O(n)
// lca - O(log(n))

vector<vector<int> > g(MAX);
int in[MAX], h[MAX], sz[MAX];
int pai[MAX], t;

void build(int k, int p = -1, int f = 1) {
    in[k] = t++; sz[k] = 1;
    for (int& i : g[k]) if (i != p) {
        pai[i] = k;
        h[i] = (i == g[k][0] ? h[k] : i);
        build(i, k, f); sz[k] += sz[i];

        if (sz[i] > sz[g[k][0]]) swap(i, g[k][0]);
    }
    if (p*f == -1) t = 0, h[k] = k, build(k, -1, 0);
}

int lca(int a, int b) {
    if (in[a] < in[b]) swap(a, b);
    return h[a] == h[b] ? b : lca(pai[h[a]], b);
}
```

## 1.9 LCA com RMQ

```
// Assume que um vertice eh ancestral dele mesmo, ou seja,
// se a eh ancestral de b, lca(a, b) = a
//
// Complexidades:
// build - O(n) + build_RMQ
// lca - RMQ

int n;
vector<vector<int> > g(MAX);
int pos[MAX];     // pos[i] : posicao de i em v (primeira
   aparicao
int ord[2 * MAX]; // ord[i] : i-esimo vertice na ordem de
   visitacao da dfs
int v[2 * MAX];   // vetor de alturas que eh usado na RMQ
int p;

void dfs(int k, int l) {
    ord[p] = k;
    pos[k] = p;
    v[p++] = l;
    for (int i = 0; i < (int) g[k].size(); i++)
        if (pos[g[k][i]] == -1) {
            dfs(g[k][i], l + 1);
            ord[p] = k;
            v[p++] = l;
        }
}

void build(int root) {
    for (int i = 0; i < n; i++) pos[i] = -1;

    p = 0;
    dfs(root, 0);

    build_RMQ();
}

int lca(int u, int v) {
    int a = pos[u], b = pos[v];
    if (a > b) swap(a, b);
    return ord[RMQ(a, b)];
}
```

## 1.10 LCA com binary lifting

```
// Assume que um vertice eh ancestral dele mesmo, ou seja,
// se a eh ancestral de b, lca(a, b) = a
// MAX2 = ceil(log(MAX))
//
// Complexidades:
// build - O(n log(n))
// lca - O(log(n))

vector<vector<int> > g(MAX);
int n, p;
int pai[MAX2][MAX];
int in[MAX], out[MAX];

void dfs(int k) {
    in[k] = p++;
    for (int i = 0; i < (int) g[k].size(); i++)
        if (in[g[k][i]] == -1) {
            pai[0][g[k][i]] = k;
            dfs(g[k][i]);
        }
    out[k] = p++;
}

void build(int raiz) {
    for (int i = 0; i < n; i++) pai[0][i] = i;
    p = 0, memset(in, -1, sizeof in);
    dfs(raiz);

    // pd dos pais
    for (int k = 1; k < MAX2; k++) for (int i = 0; i < n;
       i++)
        pai[k][i] = pai[k - 1][pai[k - 1][i]];
}

bool anc(int a, int b) { // se a eh ancestral de b
    return in[a] <= in[b] and out[a] >= out[b];
}

int lca(int a, int b) {
    if (anc(a, b)) return a;
```

```
    if (anc(b, a)) return b;

    // sobe a
    for (int k = MAX2 - 1; k >= 0; k--)
        if (!anc(pai[k][a], b)) a = pai[k][a];

    return pai[0][a];
}
```

## 1.11 Heavy-Light Decomposition - aresta

```
// SegTree de soma
// query / update de soma das arestas
//
// Complexidades:
// build - O(n)
// query_path - O(log^2 (n))
// update_path - O(log^2 (n))
// query_subtree - O(log(n))
// update_subtree - O(log(n))

#define f first
#define s second

namespace seg {
    ll seg[4*MAX], lazy[4*MAX];
    int n, *v;

    ll build(int p=1, int l=0, int r=n-1) {
        lazy[p] = 0;
        if (l == r) return seg[p] = v[l];
        int m = (l+r)/2;
        return seg[p] = build(2*p, l, m) + build(2*p+1, m+1,
            r);
    }
    void build(int n2, int* v2) {
        n = n2, v = v2;
        build();
    }
    void prop(int p, int l, int r) {
        seg[p] += lazy[p]*(r-l+1);
        if (l != r) lazy[2*p] += lazy[p], lazy[2*p+1] +=
            lazy[p];
        lazy[p] = 0;
    }
    ll query(int a, int b, int p=1, int l=0, int r=n-1) {
        prop(p, l, r);
        if (a <= l and r <= b) return seg[p];
        if (b < l or r < a) return 0;
        int m = (l+r)/2;
        return query(a, b, 2*p, l, m) + query(a, b, 2*p+1,
            m+1, r);
    }
    ll update(int a, int b, int x, int p=1, int l=0, int
        r=n-1) {
        prop(p, l, r);
        if (a <= l and r <= b) {
            lazy[p] += x;
            prop(p, l, r);
            return seg[p];
        }
        if (b < l or r < a) return seg[p];
        int m = (l+r)/2;
        return seg[p] = update(a, b, x, 2*p, l, m) +
            update(a, b, x, 2*p+1, m+1, r);
    }
};

namespace hld {
    vector<pair<int, int> > g[MAX];
    int in[MAX], out[MAX], sz[MAX];
    int sobe[MAX], pai[MAX];
    int h[MAX], v[MAX], t;

    void build_hld(int k, int p = -1, int f = 1) {
        v[in[k] = t++] = sobe[k]; sz[k] = 1;
        for (auto& i : g[k]) if (i.f != p) {
            sobe[i.f] = i.s; pai[i.f] = k;
            h[i.f] = (i == g[k][0] ? h[k] : i.f);
            build_hld(i.f, k, f); sz[k] += sz[i.f];

            if (sz[i.f] > sz[g[k][0].f]) swap(i, g[k][0]);
        }
        out[k] = t;
```

```
        if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
    }
    void build(int root = 0) {
        t = 0;
        build_hld(root);
        seg::build(t, v);
    }
    ll query_path(int a, int b) {
        if (a == b) return 0;
        if (in[a] < in[b]) swap(a, b);

        if (h[a] == h[b]) return seg::query(in[b]+1, in[a]);
        return seg::query(in[h[a]], in[a]) +
            query_path(pai[h[a]], b);
    }
    void update_path(int a, int b, int x) {
        if (a == b) return;
        if (in[a] < in[b]) swap(a, b);

        if (h[a] == h[b]) return (void)seg::update(in[b]+1,
            in[a], x);
        seg::update(in[h[a]], in[a], x);
            update_path(pai[h[a]], b, x);
    }
    ll query_subtree(int a) {
        if (in[a] == out[a]-1) return 0;
        return seg::query(in[a]+1, out[a]-1);
    }
    void update_subtree(int a, int x) {
        if (in[a] == out[a]-1) return;
        seg::update(in[a]+1, out[a]-1, x);
    }
    int lca(int a, int b) {
        if (in[a] < in[b]) swap(a, b);
        return h[a] == h[b] ? b : lca(pai[h[a]], b);
    }
};
```

## 1.12 Bellman-Ford

```
// Calcula a menor distancia
// entre a e todos os vertices e
// detecta ciclo negativo
// Retorna 1 se ha ciclo negativo
// Nao precisa representar o grafo,
// soh armazenar as arestas
//
// O(nm)

int n, m;
int d[MAX];
vector<pair<int, int> > ar; // vetor de arestas
vector<int> w;              // peso das arestas

bool bellman_ford(int a) {
    for (int i = 0; i < n; i++) d[i] = INF;
    d[a] = 0;

    for (int i = 0; i <= n; i++)
        for (int j = 0; j < m; j++) {
            if (d[ar[j].second] > d[ar[j].first] + w[j]) {
                if (i == n) return 1;

                d[ar[j].second] = d[ar[j].first] + w[j];
            }
        }

    return 0;
}
```

## 1.13 Tarjan para SCC

```
// O(n + m)

int n;
vector<vector<int> > g(MAX);
stack<int> s;
int vis[MAX], comp[MAX];
int id[MAX], p;

int dfs(int k) {
    int lo = id[k] = p++;
    s.push(k);
    vis[k] = 2; // ta na pilha
```

```
    // calcula o menor cara q ele alcanca
    // que ainda nao esta em um scc
    for (int i = 0; i < g[k].size(); i++) {
        if (!vis[g[k][i]])
            lo = min(lo, dfs(g[k][i]));
        else if (vis[g[k][i]] == 2)
            lo = min(lo, id[g[k][i]]);
    }

    // nao alcanca ninguem menor -> comeca scc
    if (lo == id[k]) while (1) {
        int u = s.top();
        s.pop(); vis[u] = 1;
        comp[u] = k;
        if (u == k) break;
    }

    return lo;
}

void tarjan() {
    memset(vis, 0, sizeof(vis));

    p = 0;
    for (int i = 0; i < n; i++) if (!vis[i]) dfs(i);
}
```

## 1.14 Dinic (Dilson)

```
// O(n^2 m)
// Grafo bipartido -> O(sqrt(n)*m)

template <class T> struct dinic {
    struct edge {
        int v, rev;
        T cap;
        edge(int v_, T cap_, int rev_) : v(v_), cap(cap_),
            rev(rev_) {}
    };
    vector<vector<edge>> g;
    vector<int> level;
    queue<int> q;
    T flow;
    int n;
    dinic(int n_) : g(n_), level(n_), n(n_) {}
    void add_edge(int u, int v, T cap) {
        if (u == v)
            return;
        edge e(v, cap, int(g[v].size()));
        edge r(u, 0, int(g[u].size()));
        g[u].push_back(e);
        g[v].push_back(r);
    }

    bool build_level_graph(int src, int sink) {
        fill(level.begin(), level.end(), -1);
        while (not q.empty())
            q.pop();
        level[src] = 0;
        q.push(src);
        while (not q.empty()) {
            int u = q.front();
            q.pop();
            for (auto e = g[u].begin(); e != g[u].end();
                ++e) {
                if (not e->cap or level[e->v] != -1)
                    continue;
                level[e->v] = level[u] + 1;
                if (e->v == sink)
                    return true;
                q.push(e->v);
            }
        }
        return false;
    }

    T blocking_flow(int u, int sink, T f) {
        if (u == sink or not f)
            return f;
        T fu = f;
        for (auto e = g[u].begin(); e != g[u].end(); ++e) {
            if (not e->cap or level[e->v] != level[u] + 1)
                continue;
```

```
            T mincap = blocking_flow(e->v, sink, min(fu,
               e->cap));
            if (mincap) {
                g[e->v][e->rev].cap += mincap;
                e->cap -= mincap;
                fu -= mincap;
            }
        }
        if (f == fu)
            level[u] = -1;
        return f - fu;
    }
    T max_flow(int src, int sink) {
        flow = 0;
        while (build_level_graph(src, sink))
            flow += blocking_flow(src, sink,
               numeric_limits<T>::max());
        return flow;
    }
};
```

## 1.15 Centro da Arvore

```
// Centro eh o vertice que minimiza
// a maior distancia dele pra alguem
// O centro fica no meio do diametro
// A funcao center retorna um par com
// o diametro e o centro
//
// O(n+m)

vector<vector<int> > g(MAX);
int n, vis[MAX];
int d[2][MAX];

// retorna ultimo vertice visitado
int bfs(int k, int x) {
        queue<int> q; q.push(k);
    memset(vis, 0, sizeof(vis));
    vis[k] = 1;
    d[x][k] = 0;
    int last = k;
    while (q.size()) {
        int u = q.front(); q.pop();
        last = u;
        for (int i : g[u]) if (!vis[i]) {
            vis[i] = 1;
            q.push(i);
            d[x][i] = d[x][u] + 1;
        }
    }
    return last;
}

pair<int, int> center() {
    int a = bfs(0, 0);
    int b = bfs(a, 1);
    bfs(b, 0);
    int c, mi = INF;
    for (int i = 0; i < n; i++) if (max(d[0][i], d[1][i]) <
       mi) {
        mi = max(d[0][i], d[1][i]), c = i;
    return {d[0][a], c};
}
```

## 1.16 Sack (DSU em arvores)

```
// Responde queries de todas as sub-arvores
// offline
//
// O(n log(n))

int sz[MAX], cor[MAX], cnt[MAX];
vector<vector<int> > g(MAX);

void build(int k, int d=0) {
    sz[k] = 1;
    for (auto& i : g[k]) {
        build(i, d+1); sz[k] += sz[i];
        if (sz[i] > sz[g[k][0]]) swap(i, g[k][0]);
    }
}
```

```
void compute(int k, int x, bool dont=1) {
    cnt[cor[k]] += x;
    for (int i = dont; i < g[k].size(); i++)
        compute(g[k][i], x, 0);
}

void solve(int k, bool keep=0) {
    for (int i = int(g[k].size())-1; i >= 0; i--)
        solve(g[k][i], !i);
    compute(k, 1);

        // agora cnt[i] tem quantas vezes a cor
        // i aparece na sub-arvore do k

    if (!keep) compute(k, -1, 0);
}
```

## 1.17 Centroid

```
// Computa os 2 centroids da arvore
//
// O(n)

int n, subsize[MAX];
vector<vector<int> > g(MAX);

void dfs(int k, int p=-1) {
    subsize[k] = 1;
    for (int i : g[k]) if (i != p) {
        dfs(i, k);
        subsize[k] += subsize[i];
    }
}

int centroid(int k, int p=-1, int size=-1) {
    if (size == -1) size = subsize[k];
    for (int i : g[k]) if (i != p) if (subsize[i] > size/2)
        return centroid(i, k, size, t);
    return k;
}

pair<int, int> centroids(int k=0) {
    dfs(k);
    int i = centroid(k), i2 = i;
    for (int j : g[i]) if (2*subsize[j] == subsize[k]) i2 =
       j;
    return {i, i2};
}
```

## 1.18 Kruskal

```
// Gera AGM a partir do vetor de arestas
//
// O(m log(n))

int n;
vector<pair<int, pair<int, int> > > ar; // vetor de arestas
int v[MAX];

// Union-Find em O(log(n))
void build();
int find(int k);
void une(int a, int b);

void kruskal() {
    build();

    sort(ar.begin(), ar.end());
    for (int i = 0; i < (int) ar.size(); i++) {
        int a = ar[i].s.f, b = ar[i].s.s;
        if (find(a) != find(b)) {
            une(a, b);
            // aresta faz parte da AGM
        }
    }
}
```

## 1.19 Blossom - matching maximo em grafo geral

```
// O(n^3)
// Se for bipartido, nao precisa da funcao
// 'contract', e roda em O(nm)
```

```
vector<vector<int> > g(MAX);
int match[MAX]; // match[i] = com quem i esta matchzado ou -1
int n, pai[MAX], base[MAX], vis[MAX];
queue<int> q;

void contract(int u, int v, bool first = 1) {
    static vector<bool> bloss;
    static int l;
    if (first) {
        bloss = vector<bool>(n, 0);
        vector<bool> teve(n, 0);
        int k = u; l = v;
        while (1) {
            teve[k = base[k]] = 1;
            if (match[k] == -1) break;
            k = pai[match[k]];
        }
        while (!teve[l = base[l]]) l = pai[match[l]];
    }
    while (base[u] != l) {
        bloss[base[u]] = bloss[base[match[u]]] = 1;
        pai[u] = v;
        v = match[u];
        u = pai[match[u]];
    }
    if (!first) return;
    contract(v, u, 0);
    for (int i = 0; i < n; i++) if (bloss[base[i]]) {
        base[i] = l;
        if (!vis[i]) q.push(i);
        vis[i] = 1;
    }
}

int getpath(int s) {
    for (int i = 0; i < n; i++) base[i] = i, pai[i] = -1,
       vis[i] = 0;
    vis[s] = 1; q = queue<int>(); q.push(s);
    while (q.size()) {
        int u = q.front(); q.pop();
        for (int i : g[u]) {
            if (base[i] == base[u] or match[u] == i)
                continue;
            if (i == s or (match[i] != -1 and pai[match[i]]
               != -1))
                contract(u, i);
            else if (pai[i] == -1) {
                pai[i] = u;
                if (match[i] == -1) return i;
                i = match[i];
                vis[i] = 1; q.push(i);
            }
        }
    }
    return -1;
}

int blossom() {
    int ans = 0;
    memset(match, -1, sizeof(match));
    for (int i = 0; i < n; i++) if (match[i] == -1)
        for (int j : g[i]) if (match[j] == -1) {
            match[i] = j;
            match[j] = i;
            ans++;
            break;
        }
    for (int i = 0; i < n; i++) if (match[i] == -1) {
        int j = getpath(i);
        if (j == -1) continue;
        ans++;
        while (j != -1) {
            int p = pai[j], pp = match[p];
            match[p] = j;
            match[j] = p;
            j = pp;
        }
    }
    return ans;
}
```

## 1.20 Max flow com lower bound

```
// Manda passar pelo menos 'lb' de fluxo
```

```
// em cada aresta
//
// O(dinic)

struct lb_max_flow : dinic {
    vector<int> d;
    vector<int> e;
    lb_max_flow(int n):dinic(n + 2), d(n, 0){}
    void add(int a, int b, int c, int lb = 0){
        c -= lb;
        d[a] -= lb;
        d[b] += lb;
        dinic::add(a, b, c);
    }
    bool check_flow(int src, int snk, int F){
        int n = d.size();
        d[src] += F;
        d[snk] -= F;

        for (int i = 0; i < n; i++){
            if (d[i] > 0){
                dinic::add(n, i, d[i]);
            } else if (d[i] < 0){
                dinic::add(i, n+1, -d[i]);
            }
        }

        int f = max_flow(n, n+1);
        return (f == F);
    }
};
```

## 1.21 Isomorfismo de Arvores

```
// Duas arvores T1 e T2 sao isomorfas
// sse T1.getHash() = T2.getHash()
//
// O(n log(n))

map<vector<int>, int> mapp;

struct tree {
    int n;
    vector<vector<int> > g;
    vector<int> subsize;

    tree(int n) {
        g.resize(n);
        subsize.resize(n);
    }
    void dfs(int k, int p=-1) {
        subsize[k] = 1;
        for (int i : g[k]) if (i != p) {
            dfs(i, k);
            subsize[k] += subsize[i];
        }
    }
    int centroid(int k, int p=-1, int size=-1) {
        if (size == -1) size = subsize[k];
        for (int i : g[k]) if (i != p)
            if (subsize[i] > size/2)
                return centroid(i, k, size);
        return k;
    }
    pair<int, int> centroids(int k=0) {
        dfs(k);
        int i = centroid(k), i2 = i;
        for (int j : g[i]) if (2*subsize[j] == subsize[k])
            i2 = j;
        return {i, i2};
    }
    int hashh(int k, int p=-1) {
        vector<int> v;
        for (int i : g[k]) if (i != p) v.push_back(hashh(i,
            k));
        sort(v.begin(), v.end());
        if (!mapp.count(v)) mapp[v] = int(mapp.size());
        return mapp[v];
    }
    ll getHash(int k=0) {
        pair<int, int> c = centroids(k);
        ll a = hashh(c.first), b = hashh(c.second);
        if (a > b) swap(a, b);
        return (a<<30)+b;
```

```
    }
};
```

## 1.22 MinCostMaxFlow Papa

```
/*
   s e t pre-definidos como MAXN - 1 e MAXN - 2.
   minCostFlow(f) computa o par (fluxo, custo) com
      max(fluxo) <= f que tenha min(custo).
   minCostFlow(INF) -> Fluxo maximo de custo minimo.
   Se tomar TLE, aleatorizar a ordem dos vertices no SPFA
 */

const int MAXN = 230;

template<typename T> struct MCMF {

    struct edge {
        int to, rev, flow, cap, residual;
        T cost;
        edge() { to = 0; rev = 0; flow = 0; cap = 0; cost =
           0; residual = false; }
        edge(int _to, int _rev, int _flow, int _cap, T
           _cost, bool _residual) {
            to = _to; rev = _rev;
            flow = _flow; cap = _cap;
            cost = _cost;
            residual = _residual;
        }
    };

    int s = MAXN - 1, t = MAXN - 2;
    vector<edge> G[MAXN];

    void addEdge(int u, int v, int w, T cost) {
        edge t = edge(v, G[v].size(), 0, w, cost, false);
        edge r = edge(u, G[u].size(), 0, 0, -cost, true);

        G[u].push_back(t);
        G[v].push_back(r);
    }

    deque<int> Q;
    bool is_inside[MAXN];
    int par_idx[MAXN], par[MAXN];
    T dist[MAXN];
    bool spfa() {
        for(int i = 0; i < MAXN; i++)
            dist[i] = INF;
        dist[t] = INF;

        Q.clear();
        dist[s] = 0;
        is_inside[s] = true;
        Q.push_back(s);

        while(!Q.empty()) {
            int u = Q.front();
            is_inside[u] = false;
            Q.pop_front();

            for(int i = 0; i < (int)G[u].size(); i++)
                if(G[u][i].cap > G[u][i].flow && dist[u] +
                   G[u][i].cost < dist[G[u][i].to]) {
                    dist[G[u][i].to] = dist[u] +
                       G[u][i].cost;
                    par_idx[G[u][i].to] = i;
                    par[G[u][i].to] = u;

                    if(is_inside[G[u][i].to]) continue;
                    if(!Q.empty() && dist[G[u][i].to] >
                       dist[Q.front()])
                       Q.push_back(G[u][i].to);
                    else Q.push_front(G[u][i].to);

                    is_inside[G[u][i].to] = true;
                }
        }

        return dist[t] != INF;
    }

    pair<int, T>  minCostFlow(int flow) {
        int f = 0;
```

```
        T ret = 0;
        while(f <= flow && spfa()) {
            int mn_flow = flow - f, u = t;
            while(u != s){
                mn_flow = min(mn_flow,
                    G[par[u]][par_idx[u]].cap -
                    G[par[u]][par_idx[u]].flow);
                u = par[u];
            }

            u = t;
            while(u != s) {
                G[par[u]][par_idx[u]].flow += mn_flow;
                G[u][G[par[u]][par_idx[u]].rev].flow -=
                    mn_flow;
                ret += G[par[u]][par_idx[u]].cost * mn_flow;
                u = par[u];
            }

            f += mn_flow;
        }

        return make_pair(f, ret);
    }

    /*
       Opcional.
       Retorna todas as arestas originais por onde passa
          fluxo = capacidade.
     */
    vector<pair<int ,int > > recover() {
        vector<pair<int, int > > used;
        for(int i = 0; i < MAXN; i++)
            for(edge e : G[i])
                if(e.flow == e.cap && !e.residual)
                    used.push_back({i, e.to});
        return used;
    }
};
```

## 1.23 Dinic (Bruno)

```
// O(n^2 m)
// Grafo com capacidades 1 -> O(sqrt(n)*m)

struct dinic{
    struct edge {
        int p, c, id; // para, capacidade, id
        edge(int p_, int c_, int id_) : p(p_), c(c_),
           id(id_) {}
    };

    vector<vector<edge>> g;
    vector<int> lev;
    dinic(int n): g(n){}
    void add(int a, int b, int c) { // de a pra b com cap. c
        g[a].pb(edge(b, c, g[b].size()));
        g[b].pb(edge(a, 0, g[a].size()-1));
    }

    bool bfs(int s, int t) {
        lev = vector<int>(g.size(), -1); lev[s] = 0;
        queue<int> q; q.push(s);
        while (q.size()) {
            int u = q.front(); q.pop();
            for (auto& i : g[u]) {
                if (lev[i.p] != -1 or !i.c) continue;
                lev[i.p] = lev[u] + 1;
                if (i.p == t) return 1;
                q.push(i.p);
            }
        }
        return 0;
    }

    int dfs(int v, int s, int f = INF){
        if (v == s) return f;
        int tem = f;
        for (auto& i : g[v]) {
            if (lev[i.p] != lev[v] + 1 or !i.c) continue;
            int foi = dfs(i.p, s, min(tem, i.c));
            tem -= foi, i.c -= foi, g[i.p][i.id].c += foi;
        }
        if (f == tem) lev[v] = -1;
```

```
        return f - tem;
    }

    int max_flow(int s, int t) {
        int f = 0;
        while (bfs(s, t)) f += dfs(s, t);
        return f;
    }
};
```

## 1.24 Line Tree

```
// Reduz min-query em arvore para RMQ
// Se o grafo nao for uma arvore, as queries
// sao sobre a arvore geradora maxima
// Queries de minimo
//
// build - O(n log(n))
// query - O(log(n))

int n;

namespace linetree {
    int id[MAX], seg[2*MAX], pos[MAX];
    vector<int> v[MAX], val[MAX];
    vector<pair<int, pair<int, int> > > ar;

    void add(int a, int b, int p) { ar.pb({p, {a, b}}); }
    void build() {
        sort(ar.rbegin(), ar.rend());
        for (int i = 0; i < n; i++) id[i] = i, v[i] = {i},
            val[i].clear();
        for (auto i : ar) {
            int a = id[i.second.first], b =
                id[i.second.second];
            if (a == b) continue;
            if (v[a].size() < v[b].size()) swap(a, b);
            for (auto j : v[b]) id[j] = a, v[a].push_back(j);
            val[a].push_back(i.first);
            for (auto j : val[b]) val[a].push_back(j);
            v[b].clear(), val[b].clear();
        }
        for (int i = 0; i < n; i++) pos[v[id[0]][i]] = i;
        for (int i = n; i < 2*n-1; i++) seg[i] =
            val[id[0]][i-n];
        for (int i = n-1; i > 0; i--) seg[i] = min(seg[2*i],
            seg[2*i+1]);
    }
    int query(int a, int b) {
        a = pos[a]+n, b = pos[b]+n;
        if (a > b) swap(a, b);
        int ans = INF; b--;
        for (; a <= b; ++a/=2, --b/=2) ans = min({ans,
            seg[a], seg[b]});
        return ans;
    }
};
```

## 1.25 2-SAT

```
// Retorna se eh possivel atribuir valores
// Grafo tem que caber 2n vertices
// add(x, y) adiciona implicacao x -> y
// Para adicionar uma clausula (x ou y)
// chamar add(nao(x), y)
// Se x tem que ser verdadeiro, chamar add(nao(x), x)
// O tarjan deve computar o componente conexo
// de cada vertice em comp
//
// O(|V|+|E|)

vector<vector<int> > g(MAX);
int n;

int nao(int x){ return (x + n) % (2*n); }

// x -> y  =  !x ou y
void add(int x, int y){
    g[x].pb(y);
    // contraposicao
    g[nao(y)].pb(nao(x));
}

bool doisSAT(){
```

```
    tarjan();
    for (int i = 0; i < m; i++)
        if (comp[i] == comp[nao(i)]) return 0;
    return 1;
}
```

### 1.26 Floyd-Warshall

```
// encontra o menor caminho entre todo
// par de vertices e detecta ciclo negativo
// returna 1 sse ha ciclo negativo
// d[i][i] deve ser 0
// para i != j, d[i][j] deve ser w se ha uma aresta
// (i, j) de peso w, INF caso contrario
//
// O(n^3)

int n;
int d[MAX][MAX];

bool floyd_warshall() {
    for (int k = 0; k < n; k++)
    for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

    for (int i = 0; i < n; i++)
        if (d[i][i] < 0) return 1;

    return 0;
}
```

## 2 Matematica

### 2.1 Ordem de elemento do grupo

```
// Calcula a ordem de a em Z_n
// O grupo Zn eh ciclico sse n =
// 1, 2, 4, p^k ou 2 p^k, p primo impar
// Retorna -1 se nao achar
//
// O(sqrt(n) log(n))

int tot(int n); // totiente em O(sqrt(n))
int expo(int a, int b, int m); // (a^b)%m em O(log(b))

// acha todos os divisores ordenados em O(sqrt(n))
vector<int> div(int n) {
    vector<int> ret1, ret2;
    for (int i = 1; i*i <= n; i++) if (n % i == 0) {
        ret1.pb(i);
        if (i*i != n) ret2.pb(n/i);
    }

    for (int i = ret2.size()-1; i+1; i--) ret1.pb(ret2[i]);
    return ret1;
}

int ordem(int a, int n) {
    vector<int> v = div(tot(n));
    for (int i : v) if (expo(a, i, n) == 1) return i;
    return -1;
}
```

### 2.2 Pollard's Rho Alg

```
// Usa o algoritmo de deteccao de ciclo de Brent
// A fatoracao nao sai necessariamente ordenada
// O algoritmo rho encontra um fator de n,
// e funciona muito bem quando n possui um fator pequeno
// Eh recomendado chamar srand(time(NULL)) na main
//
// Complexidades (considerando mul constante):
// rho - esperado O(n^(1/4)) no pior caso
// fact - esperado menos que O(n^(1/4) log(n)) no pior caso

ll mdc(ll a, ll b) { return !b ? a : mdc(b, a % b); }

ll mul(ll a, ll b, ll m) {
    return (a*b-ll(a*(long double)b/m+0.5)*m+m)%m;
}
```

```
ll exp(ll a, ll b, ll m) {
    if (!b) return 1;
    ll ans = exp(mul(a, a, m), b/2, m);
    return b%2 ? mul(a, ans, m) : ans;
}

bool prime(ll n) {
    if (n < 2) return 0;
    if (n <= 3) return 1;
    if (n % 2 == 0) return 0;

    ll d = n - 1;
    int r = 0;
    while (d % 2 == 0) {
        r++;
        d /= 2;
    }

    int a[9] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    for (int i = 0; i < 9; i++) {
        if (a[i] >= n) break;
        ll x = exp(a[i], d, n);
        if (x == 1 or x == n - 1) continue;

        bool deu = 1;
        for (int j = 0; j < r - 1; j++) {
            x = mul(x, x, n);
            if (x == n - 1) {
                deu = 0;
                break;
            }
        }
        if (deu) return 0;
    }
    return 1;
}

ll rho(ll n) {
    if (n == 1 or prime(n)) return n;
    if (n % 2 == 0) return 2;

    while (1) {
        ll x = 2, y = 2;
        ll ciclo = 2, i = 0;

        ll c = (rand() / (double) RAND_MAX) * (n - 1) + 1;
        ll d = 1;

        while (d == 1) {
            if (++i == ciclo) ciclo *= 2, y = x;
            x = (mul(x, x, n) + c) % n;

            if (x == y) break;

            d = mdc(abs(x - y), n);
        }

        if (x != y) return d;
    }
}

void fact(ll n, vector<ll>& v) {
    if (n == 1) return;
    if (prime(n)) v.pb(n);
    else {
        ll d = rho(n);
        fact(d, v);
        fact(n / d, v);
    }
}
```

## 2.3 Algoritmo de Euclides extendido

```
// acha x e y tal que ax + by = mdc(a, b)
//
// O(log(min(a, b)))

int mdce(int a, int b, int *x, int *y){
    if(!a){
        *x = 0;
        *y = 1;
        return b;
    }
```

```
    int X, Y;
    int mdc = mdce(b % a, a, &X, &Y);
    *x = Y - (b / a) * X;
    *y = X;

    return mdc;
}
```

## 2.4 Algoritmo de Euclides

```
// O(log(min(a, b)))

int mdc(int a, int b) {
    return !b ? a : mdc(b, a % b);
}
```

## 2.5 Divisão de Polinomios

```
// Divide p1 por p2
// Retorna um par com o quociente e o resto
// Os coeficientes devem estar em ordem
// decrescente pelo grau. Ex:
// 3x^2 + 2x - 1 -> [3, 2, -1]
//
// O(nm), onde n e m sao os tamanhos dos
// polinomios

typedef vector<int> vi;

pair<vi, vi> div(vi p1, vi p2) {
    vi quoc, resto;
    int a = p1.size(), b = p2.size();
    for (int i = 0; i <= a - b; i++) {
        int k = p1[i] / p2[0];
        quoc.pb(k);
        for (int j = i; j < i + b; j++)
            p1[j] -= k * p2[j - i];
    }

    for (int i = a - b + 1; i < a; i++)
        resto.pb(p1[i]);

    return mp(quoc, resto);
}
```

## 2.6 FFT

```
// Exemplos na main
//
// Soma O(n) & Multiplicacao O(nlogn)

const int MAX = 5e5;
const int MAX2 = (1 << 20);//(1 << (ceil(log2(MAX)) + 1)) + 1

const double PI = acos(-1);

struct cplx{
    double r, i;
    cplx(double r_ = 0, double i_ = 0):r(r_), i(i_){}
    const cplx operator+(const cplx &x) const{
        return cplx(r + x.r, i + x.i);
    }
    const cplx operator-(const cplx &x) const{
        return cplx(r - x.r, i - x.i);
    }
    const cplx operator*(double a) const {
        return cplx(r*a, i*a);
    }
    const cplx operator/(double a)const {
        return cplx(r/a, i/a);
    }
    const cplx operator*(const cplx x) const {
        return cplx(r*x.r - i*x.i, r*x.i + i*x.r);
    }
};

const cplx I(0, 1);
cplx X[MAX2], Y[MAX2];
int rev[MAX2];
cplx roots[MAX2];

cplx rt(int i, int n){
    double alpha = (2*i*PI)/n;
```

```
    return cplx(cos(alpha), sin(alpha));
}

void fft(cplx *a, bool f, int N){
    for (int i = 0; i < N; i++)
        if (i < rev[i])
            swap(a[i], a[rev[i]]);
    int l, r, m;
    for (int n = 2; n <= N; n *= 2){
        cplx root = (f ? rt(1, n) : rt(-1, n));
        roots[0] = 1;
        for (int i = 1; i < n/2; i++)
            roots[i] = roots[i-1]*root;
        for (int pos = 0; pos < N; pos += n){
            l = pos+0, r = pos+n/2, m = 0;
            while (m < n/2){
                auto t = roots[m]*a[r];
                a[r] = a[l] - t;
                a[l] = a[l] + t;
                l++; r++; m++;
            }
        }
    }
    if (f) for(int i = 0; i < N; i++) a[i] = a[i]/N;
}


//p(x) = at(i)*x^i
template<typename T> struct poly : vector<T> {
    poly(const vector<T> &coef):vector<T>(coef){}
    poly(unsigned size):vector<T>(size){}
    poly(){}
    T operator()(T x){
        T ans = 0, curr_x(1);
        for (auto c : *this) {
            ans = ans+c*curr_x;
            curr_x = curr_x*x;
        }
        return ans;
    }
    poly<T> operator+(const poly<T> &r){
        const poly<T> &l = *this;
        int sz = max(l.size(), r.size());
        poly<T> ans(sz);
        for (unsigned i = 0; i < l.size(); i++)
            ans[i] = ans[i]+l[i];
        for (unsigned i = 0; i < r.size(); i++)
            ans[i] = ans[i]+r[i];
        return ans;
    }
    poly<T> operator-(poly<T> &r){
        for (auto &it : r) it = -it;
        return (*this)+r;
    }
    void fix(int k){
        if (k < this->size()) throw
           logic_error("normalizando errado");
        while (this->size() < k) this->push_back(0);
    }
    pair<poly<T>, poly<T>> split(){
        const poly<T> &p = *this;
        poly<T> l, r;
        for (int i = 0; i < p.size(); i++)
            if (i&1) l.push_back(p[i]);
            else r.push_back(p[i]);
        return {l, r};
    }
    poly<T> operator*(const poly<T> r){
        const poly<T> &l = *this;
        int ln = l.size(), rn = r.size();
        int N = ln+rn+1;
        int n = 1, log_n = 0;
        while (n <= N) { n <<= 1; log_n++; }
        for (int i = 0; i < n; ++i){
            rev[i] = 0;
            for (int j = 0; j < log_n; ++j)
                if (i & (1<<j))
                    rev[i] |= 1 << (log_n-1-j);
        }
        for (int i = 0; i < ln; i++) X[i] = l[i];
        for (int i = ln; i < n; i++) X[i] = 0;
        for (int i = 0; i < rn; i++) Y[i] = r[i];
        for (int i = rn; i < n; i++) Y[i] = 0;
        fft(X, false, n);//call dft if possible
```

```
        fft(Y, false, n);

        for (int i = 0; i < n; i++)
            Y[i] = X[i]*Y[i];

        fft(Y, true, n);
        poly<T> ans(N);
        for (int i = 0; i < N; i++){
            ans[i] = floor(Y[i].r + 0.25); //if T is integer
            //ans[i] = Y[i].r; //if T is floating point
        }


        while (!ans.empty() && ans.back() == 0)
            ans.pop_back();
        return ans;
    }
    pair<poly<T>, T> briot_ruffini(T r){//for p = Q(x - r) +
       R, returns (Q, R)
        const poly<T> &l = *this;
        int sz = l.size();
        if (sz == 0) return {poly<T>(0), 0};
        poly<T> q(sz - 1);
        q.back() = l.back();
        for (int i = q.size()-2; i >= 0; i--){
            cout << i << "~" << q.size() << endl;
            q[i] = q[i+1]*r + l[i+1];
        }
        return {q, q[0]*r + l[0]};
    }
};
template<typename T> ostream& operator<<(ostream &out, const
   poly<T> &p){
    if (p.empty()) return out;
    out << p.at(0);
    for (int i = 1; i < p.size(); i++)
        out << " + " << p.at(i) << "x^" << i;
    out << endl;
    return out;
}

int main(){ _
    poly<int> p({-2, -1, 2, 1});
    poly<int> q({1, 1, 1});
    poly<int> sum = p+q;
    poly<int> mult = p*q;
    cout << "p: " << p << endl;
    cout << "q: " << q << endl;
    cout << "pq: " << mult << endl;
    for (int i = 1; i <= 50; i++){
        auto P = p(i), Q = q(i), M = mult(i);
        cout << P*Q << "\t\tvs\t\t" << M << endl;
        if (abs(P*Q - M) > 1e-5) throw logic_error("bad
           implementation :(");
    }
    cout << "sucesso!" << endl;
    exit(0);
    for (int root : {1, -1, 2, -2, 3}){
        poly<int> t; int r;
        tie(t, r) = p.briot_ruffini(root);
        cout << p << "/" << poly<int>({-root, 1});
        cout << " = " << endl;
        cout << t << " + " << r << endl;
        cout << endl;
    }

    exit(0);
}
```

## 2.7 Miller-Rabin

```
// Testa se n eh primo, n <= 3 * 10^18
//
// O(log(n)), considerando multiplicacao
// e exponenciacao constantes

// multiplicacao modular
ll mul(ll x, ll y, ll m); // x*y mod m
ll exp(ll x, ll y, ll m); // x^y mod m;

bool prime(ll n) {
    if (n < 2) return 0;
    if (n <= 3) return 1;
    if (n % 2 == 0) return 0;
```

```
    ll d = n - 1;
    int r = 0;
    while (d % 2 == 0) r++, d /= 2;

    // com esses primos, o teste funciona garantido para n
       <= 3*10^18
    // funciona para n <= 3*10^24 com os primos ate 41
    int a[9] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    // outra opcao para n <= 2^64:
    // int a[7] = {2, 325, 9375, 28178, 450775, 9780504,
       1795265022};

    for (int i = 0; i < 9; i++) {
        if (a[i] >= n) break;
        ll x = exp(a[i], d, n);
        if (x == 1 or x == n - 1) continue;

        bool deu = 1;
        for (int j = 0; j < r - 1; j++) {
            x = mul(x, x, n);
            if (x == n - 1) {
                deu = 0;
                break;
            }
        }
        if (deu) return 0;
    }
    return 1;
}
```

## 2.8 Inverso Modular

```
// Computa o inverso de a modulo b
// Se b eh primo, basta fazer
// a^(b-2)

long long inv(long long a, long long b){
    return 1<a ? b - inv(b%a,a)*b/a : 1;
}
```

## 2.9 Variacoes do crivo de Eratosthenes

```
// "O" crivo
//
// Encontra maior divisor primo
// Um numero eh primo sse div[x] == x
// fact fatora um numero <= lim
// A fatoracao sai ordenada
//
// crivo - O(n log(log(n)))
// fact - O(log(n))

int divi[MAX];

void crivo(int lim) {
    for (int i = 1; i <= lim; i++) divi[i] = 1;

    for (int i = 2; i <= lim; i++) if (divi[i] == 1)
        for (int j = i; j <= lim; j += i) divi[j] = i;
}

void fact(vector<int>& v, int n) {
    if (n != divi[n]) fact(v, n/divi[n]);
    v.push_back(divi[n]);
}

// Crivo de divisores
//
// Encontra numero de divisores
// ou soma dos divisores
//
// O(n log(n))

int divi[MAX];

void crivo(int lim) {
    for (int i = 1; i <= lim; i++) divi[i] = 1;

    for (int i = 2; i <= lim; i++)
        for (int j = i; j <= lim; j += i) {
            // para numero de divisores
            divi[j]++;
```

```
            // para soma dos divisores
            divi[j] += i;
        }
}

// Crivo de totiente
//
// Encontra o valor da funcao
// totiente de Euler
//
// O(n log(log(n)))

int tot[MAX];

void crivo(int lim) {
    for (int i = 1; i <= lim; i++) tot[i] = i;

    for (int i = 2; i <= lim; i++) if (tot[i] == i)
        for (int j = i; j <= lim; j += i)
            tot[j] -= tot[j] / i;
}
```

## 2.10 Totiente

```
// O(sqrt(n))

int tot(int n){
    int ret = n;

    for (int i = 2; i*i <= n; i++) if (n % i == 0) {
        while (n % i == 0) n /= i;
        ret -= ret / i;
    }
    if (n > 1) ret -= ret / n;

    return ret;
}
```

## 2.11 Exponenciacao rapida

```
// (x^y mod m) em O(log(y))

typedef long long int ll;

ll pow(ll x, ll y, ll m) { // iterativo
    ll ret = 1;
    while (y) {
        if (y & 1) ret = (ret * x) % m;
        y >>= 1;
        x = (x * x) % m;
    }
    return ret;
}

ll pow(ll x, ll y, ll m) { // recursivo
    if (!y) return 1;
    ll ans = pow(x*x%m, y/2, m);
    return y%2 ? x*ans%m : ans;
}
```

## 2.12 Produto de dois long long mod m

```
// O(1)

typedef long long int ll;

ll mul(ll a, ll b, ll m) { // a*b % m
    return (a*b-ll(a*(long double)b/m+0.5)*m+m)%m;
}
```

# 3 Primitivas

## 3.1 Primitivas de Polinomios

```
#include <bits/stdc++.h>

using namespace std;
namespace algebra {
    const int inf = 1e9;
    const int magic = 500; // threshold for sizes to run the
        naive algo
```

```
namespace fft {
    const int maxn = 1 << 18;

    typedef double ftype;
    typedef complex<ftype> point;

    point w[maxn];
    const ftype pi = acos(-1);
    bool initiated = 0;
    void init() {
        if(!initiated) {
            for(int i = 1; i < maxn; i *= 2) {
                for(int j = 0; j < i; j++) {
                    w[i + j] = polar(ftype(1), pi * j /
                       i);
                }
            }
            initiated = 1;
        }
    }
    template<typename T>
        void fft(T *in, point *out, int n, int k = 1) {
            if(n == 1) {
                *out = *in;
            } else {
                n /= 2;
                fft(in, out, n, 2 * k);
                fft(in + k, out + n, n, 2 * k);
                for(int i = 0; i < n; i++) {
                    auto t = out[i + n] * w[i + n];
                    out[i + n] = out[i] - t;
                    out[i] += t;
                }
            }
        }

    template<typename T>
        void mul_slow(vector<T> &a, const vector<T> &b) {
            vector<T> res(a.size() + b.size() - 1);
            for(size_t i = 0; i < a.size(); i++) {
                for(size_t j = 0; j < b.size(); j++) {
                    res[i + j] += a[i] * b[j];
                }
            }
            a = res;
        }


    template<typename T>
        void mul(vector<T> &a, const vector<T> &b) {
            if(min(a.size(), b.size()) < magic) {
                mul_slow(a, b);
                return;
            }
            init();
            static const int shift = 15, mask = (1 <<
               shift) - 1;
            size_t n = a.size() + b.size() - 1;
            while(__builtin_popcount(n) != 1) {
                n++;
            }
            a.resize(n);
            static point A[maxn], B[maxn];
            static point C[maxn], D[maxn];
            for(size_t i = 0; i < n; i++) {
                A[i] = point(a[i] & mask, a[i] >> shift);
                if(i < b.size()) {
                    B[i] = point(b[i] & mask, b[i] >>
                       shift);
                } else {
                    B[i] = 0;
                }
            }
            fft(A, C, n); fft(B, D, n);
            for(size_t i = 0; i < n; i++) {
                point c0 = C[i] + conj(C[(n - i) % n]);
                point c1 = C[i] - conj(C[(n - i) % n]);
                point d0 = D[i] + conj(D[(n - i) % n]);
                point d1 = D[i] - conj(D[(n - i) % n]);
                A[i] = c0 * d0 - point(0, 1) * c1 * d1;
                B[i] = c0 * d1 + d0 * c1;
            }
            fft(A, C, n); fft(B, D, n);
```

```
            reverse(C + 1, C + n);
            reverse(D + 1, D + n);
            int t = 4 * n;
            for(size_t i = 0; i < n; i++) {
                int64_t A0 = llround(real(C[i]) / t);
                T A1 = llround(imag(D[i]) / t);
                T A2 = llround(imag(C[i]) / t);
                a[i] = A0 + (A1 << shift) + (A2 << 2 *
                    shift);
            }
            return;
        }
}
template<typename T>
    T bpow(T x, size_t n) {
        return n ? n % 2 ? x * bpow(x, n - 1) : bpow(x *
            x, n / 2) : T(1);
    }
template<typename T>
    T bpow(T x, size_t n, T m) {
        return n ? n % 2 ? x * bpow(x, n - 1, m) % m :
            bpow(x * x % m, n / 2, m) : T(1);
    }
template<typename T>
    T gcd(const T &a, const T &b) {
        return b == T(0) ? a : gcd(b, a % b);
    }
template<typename T>
    T nCr(T n, int r) { // runs in O(r)
        T res(1);
        for(int i = 0; i < r; i++) {
            res *= (n - T(i));
            res /= (i + 1);
        }
        return res;
    }

template<int m>
    struct modular {
        int64_t r;
        modular() : r(0) {}
        modular(int64_t rr) : r(rr) {if(abs(r) >= m) r
            %= m; if(r < 0) r += m;}
        modular inv() const {return bpow(*this, m - 2);}
        modular operator * (const modular &t) const
            {return (r * t.r) % m;}
        modular operator / (const modular &t) const
            {return *this * t.inv();}
        modular operator += (const modular &t) {r +=
            t.r; if(r >= m) r -= m; return *this;}
        modular operator -= (const modular &t) {r -=
            t.r; if(r < 0) r += m; return *this;}
        modular operator + (const modular &t) const
            {return modular(*this) += t;}
        modular operator - (const modular &t) const
            {return modular(*this) -= t;}
        modular operator *= (const modular &t) {return
            *this = *this * t;}
        modular operator /= (const modular &t) {return
            *this = *this / t;}

        bool operator == (const modular &t) const
            {return r == t.r;}
        bool operator != (const modular &t) const
            {return r != t.r;}

        operator int64_t() const {return r;}
    };
template<int T>
    istream& operator >> (istream &in, modular<T> &x) {
        return in >> x.r;
    }


template<typename T>
    struct poly {
        vector<T> a;

        void normalize() { // get rid of leading zeroes
            while(!a.empty() && a.back() == T(0)) {
                a.pop_back();
            }
        }
```

```
poly(){}
poly(T a0) : a{a0}{normalize();}
poly(vector<T> t) : a(t){normalize();}

poly operator += (const poly &t) {
    a.resize(max(a.size(), t.a.size()));
    for(size_t i = 0; i < t.a.size(); i++) {
        a[i] += t.a[i];
    }
    normalize();
    return *this;
}
poly operator -= (const poly &t) {
    a.resize(max(a.size(), t.a.size()));
    for(size_t i = 0; i < t.a.size(); i++) {
        a[i] -= t.a[i];
    }
    normalize();
    return *this;
}
poly operator + (const poly &t) const {return
   poly(*this) += t;}
poly operator - (const poly &t) const {return
   poly(*this) -= t;}

poly mod_xk(size_t k) const { // get same
   polynomial mod x^k
    k = min(k, a.size());
    return vector<T>(begin(a), begin(a) + k);
}
poly mul_xk(size_t k) const { // multiply by x^k
    poly res(*this);
    res.a.insert(begin(res.a), k, 0);
    return res;
}
poly div_xk(size_t k) const { // divide by x^k,
   dropping coefficients
    k = min(k, a.size());
    return vector<T>(begin(a) + k, end(a));
}
poly substr(size_t l, size_t r) const { //
   return mod_xk(r).div_xk(l)
    l = min(l, a.size());
    r = min(r, a.size());
    return vector<T>(begin(a) + l, begin(a) + r);
}
poly inv(size_t n) const { // get inverse series
   mod x^n
    assert(!is_zero());
    poly ans = a[0].inv();
    size_t a = 1;
    while(a < n) {
        poly C = (ans * mod_xk(2 * a)).substr(a,
           2 * a);
        ans -= (ans * C).mod_xk(a).mul_xk(a);
        a *= 2;
    }
    return ans.mod_xk(n);
}

poly operator *= (const poly &t) {fft::mul(a,
   t.a); normalize(); return *this;}
poly operator * (const poly &t) const {return
   poly(*this) *= t;}

poly reverse(size_t n, bool rev = 0) const { //
   reverses and leaves only n terms
    poly res(*this);
    if(rev) { // If rev = 1 then tail goes to
       head
        res.a.resize(max(n, res.a.size()));
    }
    std::reverse(res.a.begin(), res.a.end());
    return res.mod_xk(n);
}

pair<poly, poly> divmod_slow(const poly &b)
   const { // when divisor or quotient is small
    vector<T> A(a);
    vector<T> res;
    while(A.size() >= b.a.size()) {
        res.push_back(A.back() / b.a.back());
        if(res.back() != T(0)) {
            for(size_t i = 0; i < b.a.size();
```

```
                i++) {
                 A[A.size() - i - 1] -=
                    res.back() * b.a[b.a.size() -
                    i - 1];
            }
        }
        A.pop_back();
    }
    std::reverse(begin(res), end(res));
    return {res, A};
}

pair<poly, poly> divmod(const poly &b) const {
    // returns quotiend and remainder of a mod b
    if(deg() < b.deg()) {
        return {poly{0}, *this};
    }
    int d = deg() - b.deg();
    if(min(d, b.deg()) < magic) {
        return divmod_slow(b);
    }
    poly D = (reverse(d + 1) * b.reverse(d +
        1).inv(d + 1)).mod_xk(d + 1).reverse(d +
        1, 1);
    return {D, *this - D * b};
}

poly operator / (const poly &t) const {return
    divmod(t).first;}
poly operator % (const poly &t) const {return
    divmod(t).second;}
poly operator /= (const poly &t) {return *this =
    divmod(t).first;}
poly operator %= (const poly &t) {return *this =
    divmod(t).second;}
poly operator *= (const T &x) {
    for(auto &it: a) {
        it *= x;
    }
    normalize();
    return *this;
}
```

```
poly operator /= (const T &x) {
    for(auto &it: a) {
        it /= x;
    }
    normalize();
    return *this;
}
poly operator * (const T &x) const {return
    poly(*this) *= x;}
poly operator / (const T &x) const {return
    poly(*this) /= x;}

void print() const {
    for(auto it: a) {
        cout << it << ' ';
    }
    cout << endl;
}
T eval(T x) const { // evaluates in single point
    x
    T res(0);
    for(int i = int(a.size()) - 1; i >= 0; i--) {
        res *= x;
        res += a[i];
    }
    return res;
}

T& lead() { // leading coefficient
    return a.back();
}
int deg() const { // degree
    return a.empty() ? -inf : a.size() - 1;
}
bool is_zero() const { // is polynomial zero
    return a.empty();
}
T operator [](int idx) const {
    return idx >= (int)a.size() || idx < 0 ?
        T(0) : a[idx];
}
```

```
T& coef(size_t idx) { // mutable reference at
    coefficient
    return a[idx];
}
bool operator == (const poly &t) const {return a
    == t.a;}
bool operator != (const poly &t) const {return a
    != t.a;}

poly deriv() { // calculate derivative
    vector<T> res;
    for(int i = 1; i <= deg(); i++) {
        res.push_back(T(i) * a[i]);
    }
    return res;
}
poly integr() { // calculate integral with C = 0
    vector<T> res = {0};
    for(int i = 0; i <= deg(); i++) {
        res.push_back(a[i] / T(i + 1));
    }
    return res;
}
size_t leading_xk() const { // Let p(x) = x^k *
    t(x), return k
    if(is_zero()) {
        return inf;
    }
    int res = 0;
    while(a[res] == T(0)) {
        res++;
    }
    return res;
}
poly log(size_t n) { // calculate log p(x) mod
    x^n
    assert(a[0] == T(1));
    return (deriv().mod_xk(n) *
        inv(n)).integr().mod_xk(n);
}
poly exp(size_t n) { // calculate exp p(x) mod
    x^n
    if(is_zero()) {
        return T(1);
    }
    assert(a[0] == T(0));
    poly ans = T(1);
    size_t a = 1;
    while(a < n) {
        poly C = ans.log(2 * a).div_xk(a) -
            substr(a, 2 * a);
        ans -= (ans * C).mod_xk(a).mul_xk(a);
        a *= 2;
    }
    return ans.mod_xk(n);

}
poly pow_slow(size_t k, size_t n) { // if k is
    small
    return k ? k % 2 ? (*this * pow_slow(k - 1,
        n)).mod_xk(n) : (*this *
        *this).mod_xk(n).pow_slow(k / 2, n) :
        T(1);
}
poly pow(size_t k, size_t n) { // calculate
    p^k(n) mod x^n
    if(is_zero()) {
        return *this;
    }
    if(k < magic) {
        return pow_slow(k, n);
    }
    int i = leading_xk();
    T j = a[i];
    poly t = div_xk(i) / j;
    return bpow(j, k) * (t.log(n) *
        T(k)).exp(n).mul_xk(i * k).mod_xk(n);
}
poly mulx(T x) { // component-wise
    multiplication with x^k
    T cur = 1;
    poly res(*this);
    for(int i = 0; i <= deg(); i++) {
        res.coef(i) *= cur;
```

```
        cur *= x;
    }
    return res;
}
poly mulx_sq(T x) { // component-wise
   multiplication with x^{k^2}
    T cur = x;
    T total = 1;
    T xx = x * x;
    poly res(*this);
    for(int i = 0; i <= deg(); i++) {
        res.coef(i) *= total;
        total *= cur;
        cur *= xx;
    }
    return res;
}
vector<T> chirpz_even(T z, int n) { // P(1),
   P(z^2), P(z^4), ..., P(z^2(n-1))
    int m = deg();
    if(is_zero()) {
        return vector<T>(n, 0);
    }
    vector<T> vv(m + n);
    T zi = z.inv();
    T zz = zi * zi;
    T cur = zi;
    T total = 1;
    for(int i = 0; i <= max(n - 1, m); i++) {
        if(i <= m) {vv[m - i] = total;}
        if(i < n) {vv[m + i] = total;}
        total *= cur;
        cur *= zz;
    }
    poly w = (mulx_sq(z) * vv).substr(m, m +
       n).mulx_sq(z);
    vector<T> res(n);
    for(int i = 0; i < n; i++) {
        res[i] = w[i];
    }
    return res;
}
```

```
vector<T> chirpz(T z, int n) { // P(1), P(z),
   P(z^2), ..., P(z^(n-1))
    auto even = chirpz_even(z, (n + 1) / 2);
    auto odd = mulx(z).chirpz_even(z, n / 2);
    vector<T> ans(n);
    for(int i = 0; i < n / 2; i++) {
        ans[2 * i] = even[i];
        ans[2 * i + 1] = odd[i];
    }
    if(n % 2 == 1) {
        ans[n - 1] = even.back();
    }
    return ans;
}
template<typename iter>
    vector<T> eval(vector<poly> &tree, int v,
       iter l, iter r) { // auxiliary evaluation
       function
        if(r - l == 1) {
            return {eval(*l)};
        } else {
            auto m = l + (r - l) / 2;
            auto A = (*this % tree[2 *
               v]).eval(tree, 2 * v, l, m);
            auto B = (*this % tree[2 * v +
               1]).eval(tree, 2 * v + 1, m, r);
            A.insert(end(A), begin(B), end(B));
            return A;
        }
    }
vector<T> eval(vector<T> x) { // evaluate
   polynomial in (x1, ..., xn)
    int n = x.size();
    if(is_zero()) {
        return vector<T>(n, T(0));
    }
    vector<poly> tree(4 * n);
    build(tree, 1, begin(x), end(x));
    return eval(tree, 1, begin(x), end(x));
}
template<typename iter>
    poly inter(vector<poly> &tree, int v, iter
```

```
            l, iter r, iter ly, iter ry) { //
            auxiliary interpolation function
             if(r - l == 1) {
                 return {*ly / a[0]};
             } else {
                 auto m = l + (r - l) / 2;
                 auto my = ly + (ry - ly) / 2;
                 auto A = (*this % tree[2 *
                    v]).inter(tree, 2 * v, l, m, ly,
                    my);
                 auto B = (*this % tree[2 * v +
                    1]).inter(tree, 2 * v + 1, m, r,
                    my, ry);
                 return A * tree[2 * v + 1] + B *
                    tree[2 * v];
             }
         }
    };
template<typename T>
    poly<T> operator * (const T& a, const poly<T>& b) {
        return b * a;
    }

template<typename T>
    poly<T> xk(int k) { // return x^k
        return poly<T>{1}.mul_xk(k);
    }

template<typename T>
    T resultant(poly<T> a, poly<T> b) { // computes
       resultant of a and b
        if(b.is_zero()) {
            return 0;
        } else if(b.deg() == 0) {
            return bpow(b.lead(), a.deg());
        } else {
            int pw = a.deg();
            a %= b;
            pw -= a.deg();
            T mul = bpow(b.lead(), pw) * T((b.deg() &
               a.deg() & 1) ? -1 : 1);
            T ans = resultant(b, a);
            return ans * mul;
        }
    }
    template<typename iter>
        poly<typename iter::value_type> kmul(iter L, iter R)
           { // computes (x-a1)(x-a2)...(x-an) without
           building tree
            if(R - L == 1) {
                return vector<typename
                   iter::value_type>{-*L, 1};
            } else {
                iter M = L + (R - L) / 2;
                return kmul(L, M) * kmul(M, R);
            }
        }
    template<typename T, typename iter>
        poly<T> build(vector<poly<T>> &res, int v, iter L,
           iter R) { // builds evaluation tree for
           (x-a1)(x-a2)...(x-an)
            if(R - L == 1) {
                return res[v] = vector<T>{-*L, 1};
            } else {
                iter M = L + (R - L) / 2;
                return res[v] = build(res, 2 * v, L, M) *
                   build(res, 2 * v + 1, M, R);
            }
        }
    template<typename T>
        poly<T> inter(vector<T> x, vector<T> y) { //
           interpolates minimum polynomial from (xi, yi)
           pairs
            int n = x.size();
            vector<poly<T>> tree(4 * n);
            return build(tree, 1, begin(x),
               end(x)).deriv().inter(tree, 1, begin(x),
               end(x), begin(y), end(y));
        }
};

using namespace algebra;

const int mod = 1e9 + 7;
```

```
typedef modular<mod> base;
typedef poly<base> polyn;

using namespace algebra;

signed main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int n = 100000;
    polyn a;
    vector<base> x;
    for(int i = 0; i <= n; i++) {
        a.a.push_back(1 + rand() % 100);
        x.push_back(1 + rand() % (2 * n));
    }
    sort(begin(x), end(x));
    x.erase(unique(begin(x), end(x)), end(x));
    auto b = a.eval(x);
    cout << clock() / double(CLOCKS_PER_SEC) << endl;
    auto c = inter(x, b);
    polyn md = kmul(begin(x), end(x));
    cout << clock() / double(CLOCKS_PER_SEC) << endl;
    assert(c == a % md);
    return 0;
}
```

## 3.2 Primitivas Geometricas

```
#include <bits/stdc++.h>

using namespace std;

#define sc(a) scanf("%d", &a)
#define sc2(a,b) scanf("%d %d", &a, &b)
#define pri(x) printf("%d\n", x)
#define prie(x) printf("%d ", x)
#define sz(x) (int)((x).size())
#define mp make_pair
#define pb push_back
#define f first
#define s second
#define sq(x) ((x)*(x))
#define BUFF ios::sync_with_stdio(false)

typedef long long int ll;
typedef double ld;
typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<vi> vvi;
typedef vector<ii> vii;
const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;
const ld DINF = 1e18;
const ld pi = acos(-1.0);
const ld eps = 1e-9;

bool eq(ld a, ld b) {
    return abs(a - b) <= eps;
}

struct pt { // ponto
    ld x, y;
    pt() {}
    pt(ld x, ld y) : x(x), y(y) {}
    bool operator < (const pt p) const {
        if (!eq(x, p.x)) return x < p.x;
        return y < p.y;
    }
    bool operator == (const pt p) const {
        return eq(x, p.x) and eq(y, p.y);
    }
    pt operator + (const pt p) const { return pt(x+p.x,
       y+p.y); }
    pt operator - (const pt p) const { return pt(x-p.x,
       y-p.y); }
    pt operator * (const ld c) const { return pt(x*c  , y*c
       ); }
    pt operator / (const ld c) const { return pt(x/c  , y/c
       ); }
};

struct line { // reta
    pt p, q;
    line() {}
```

```
    line(pt p, pt q) : p(p), q(q) {}
};

// PONTO & VETOR

ld dist(pt p, pt q) { // distancia
    return sqrt(sq(p.x - q.x) + sq(p.y - q.y));
}

ld dist2(pt p, pt q) { // quadrado da distancia
    return sq(p.x - q.x) + sq(p.y - q.y);
}

ld norm(pt v) { // norma do vetor
    return dist(pt(0, 0), v);
}

pt normalize(pt v) { // vetor normalizado
    if (!norm(v)) return v;
    v = v / norm(v);
    return v;
}

ld dot(pt u, pt v) { // produto escalar
    return u.x * v.x + u.y * v.y;
}

ld cross(pt u, pt v) { // norma do produto vetorial
    return u.x * v.y - u.y * v.x;
}

ld sarea(pt p, pt q, pt r) { // area com sinal
    return cross(q - p, r - q) / 2;
}

bool col(pt p, pt q, pt r) { // se p, q e r sao colin.
    return eq(sarea(p, q, r), 0);
}

int paral(pt u, pt v) { // se u e v sao paralelos
    u = normalize(u);
    v = normalize(v);
    if (eq(u.x, v.x) and eq(u.y, v.y)) return 1;
    if (eq(u.x, -v.x) and eq(u.y, -v.y)) return -1;
    return 0;
}

bool ccw(pt p, pt q, pt r) { // se p, q, r sao ccw
    return sarea(p, q, r) > 0;
}

pt rotate(pt p, ld th) { // rotaciona o ponto th radianos
    return pt(p.x * cos(th) - p.y * sin(th),
              p.x * sin(th) + p.y * cos(th));
}

pt rotate90(pt p) { // rotaciona 90 graus
    return pt(-p.y, p.x);
}

// RETA

bool isvert(line r) { // se r eh vertical
    return eq(r.p.x, r.q.x);
}

ld getm(line r) { // coef. ang. de r
    if (isvert(r)) return DINF;
    return (r.p.y - r.q.y) / (r.p.x - r.q.x);
}

ld getn(line r) { // coef. lin. de r
    if (isvert(r)) return DINF;
    return r.p.y - getm(r) * r.p.x;
}

bool lineeq(line r, line s) { // r == s
    return col(r.p, r.q, s.p) and col(r.p, r.q, s.q);
}

bool paraline(line r, line s) { // se r e s sao paralelas
    if (isvert(r) and isvert(s)) return 1;
    if (isvert(r) or isvert(s)) return 0;
    return eq(getm(r), getm(s));
```

```
}

bool isinline(pt p, line r) { // se p pertence a r
    return col(p, r.p, r.q);
}

bool isinseg(pt p, line r) { // se p pertence ao seg de r
    if (p == r.p or p == r.q) return 1;
    return paral(p - r.p, p - r.q) == -1;
}

pt proj(pt p, line r) { // projecao do ponto p na reta r
    if (r.p == r.q) return r.p;
    r.q = r.q - r.p; p = p - r.p;
    pt proj = r.q * (dot(p, r.q) / dot(r.q, r.q));
    return proj + r.p;
}

pt inter(line r, line s) { // r inter s
    if (paraline(r, s)) return pt(DINF, DINF);

    if (isvert(r)) return pt(r.p.x, getm(s) * r.p.x +
       getn(s));
    if (isvert(s)) return pt(s.p.x, getm(r) * s.p.x +
       getn(r));

    ld x = (getn(s) - getn(r)) / (getm(r) - getm(s));
    return pt(x, getm(r) * x + getn(r));
}

bool interseg(line r, line s) { // se o seg de r intercepta
   o seg de s
    if (paraline(r, s)) {
        return isinseg(r.p, s) or isinseg(r.q, s)
            or isinseg(s.p, r) or isinseg(s.q, r);
    }

    pt i = inter(r, s);
    return isinseg(i, r) and isinseg(i, s);
}

ld disttoline(pt p, line r) { // distancia do ponto a reta
    return dist(p, proj(p, r));
}

ld disttoseg(pt p, line r) { // distancia do ponto ao seg
    if (isinseg(proj(p, r), r))
        return disttoline(p, r);
    return min(dist(p, r.p), dist(p, r.q));
}

ld distseg(line a, line b) { // distancia entre seg
    if (interseg(a, b)) return 0;

    ld ret = DINF;
    ret = min(ret, disttoseg(a.p, b));
    ret = min(ret, disttoseg(a.q, b));
    ret = min(ret, disttoseg(b.p, a));
    ret = min(ret, disttoseg(b.q, a));

    return ret;
}

// POLIGONO

ld polper(vector<pt> v) { // perimetro do poligono
    ld ret = 0;
    for (int i = 0; i < sz(v); i++)
        ret += dist(v[i], v[(i + 1) % sz(v)]);
    return ret;
}

ld polarea(vector<pt> v) { // area do poligono
    ld ret = 0;
    for (int i = 0; i < sz(v); i++)
        ret += sarea(pt(0, 0), v[i], v[(i + 1) % sz(v)]);
    return abs(ret);
}

bool onpol(pt p, vector<pt> v) { // se um ponto esta na
   fronteira do poligono
    for (int i = 0; i < sz(v); i++)
        if (isinseg(p, line(v[i], v[(i + 1) % sz(v)])))
            return 1;
```

```
    return 0;
}

bool inpol(pt p, vector<pt> v) { // se um ponto pertence ao
   poligono
    if (onpol(p, v)) return 1;
    int c = 0;
    line r = line(p, pt(DINF, pi * DINF));
    for (int i = 0; i < sz(v); i++) {
        line s = line(v[i], v[(i + 1) % sz(v)]);
        if (interseg(r, s)) c++;
    }
    return c & 1;
}

bool interpol(vector<pt> v1, vector<pt> v2) { // se dois
   poligonos se interceptam
    for (int i = 0; i < sz(v1); i++) if (inpol(v1[i], v2))
       return 1;
    for (int i = 0; i < sz(v2); i++) if (inpol(v2[i], v1))
       return 1;
    return 0;
}

ld distpol(vector<pt> v1, vector<pt> v2) { // distancia
   entre poligonos
    if (interpol(v1, v2)) return 0;

    ld ret = DINF;

    for (int i = 0; i < sz(v1); i++) for (int j = 0; j <
       sz(v2); j++)
        ret = min(ret, distseg(line(v1[i], v1[(i + 1) %
           sz(v1)]),
                   line(v2[j], v2[(j + 1) % sz(v2)])));
    return ret;
}

vector<pt> convexhull(vector<pt> v) { // convex hull
    vector<pt> l, u;

    sort(v.begin(), v.end());

    for (int i = 0; i < sz(v); i++) {
        while (sz(l) > 1 and !ccw(v[i], l[sz(l) - 1],
           l[sz(l) - 2]))
            l.pop_back();
        l.pb(v[i]);
    }
    for (int i = sz(v) - 1; i >= 0; i--) {
        while (sz(u) > 1 and !ccw(v[i], u[sz(u) - 1],
           u[sz(u) - 2]))
            u.pop_back();
        u.pb(v[i]);
    }

    l.pop_back(); u.pop_back();

    for (int i = 0; i < sz(u); i++) l.pb(u[i]);

    return l;
}

// CIRCULO

pt getcenter(pt a, pt b, pt c) { // centro da circunferencia
   dado 3 pontos
    b = (a + b) / 2;
    c = (a + c) / 2;
    return inter(line(b, b + rotate90(a - b)),
            line(c, c + rotate90(a - c)));
}


circle minCirc(vector<PT> v) { // minimum enclosing circle
    int n = v.size();
    random_shuffle(v.begin(), v.end());
    PT p = PT(0.0, 0.0);
    circle ret = circle(p, 0.0);
    for(int i = 0; i < n; i++) {
        if(!inside(ret, v[i])) {
            ret = circle(v[i], 0);
            for(int j = 0; j < i; j++) {
                if(!inside(ret, v[j])) {
```

```
                    ret = circle((v[i] + v[j]) / 2.0,
                        sqrt(dist2(v[i], v[j])) / 2.0);
                    for(int k = 0; k < j; k++) {
                        if(!inside(ret, v[k])) {
                            p = best0f3(v[i], v[j], v[k]);
                            ret = circle(p, sqrt(dist2(p,
                                v[i])));
                        }
                    }
                }
            }
        }
    }
    return ret;
}

// comparador pro set para fazer sweep angle com segmentos
double ang;
struct cmp {
    bool operator () (const line& a, const line& b) {
        line r = line(pt(0, 0), rotate(pt(1, 0), ang));
        return norm(inter(r, a)) < norm(inter(r, b));
    }
};
```

## 3.3 Primitivas de matriz (Rafael)

```
ll mod(ll v){ return (v + MOD) % MOD; }
ll sum(ll l, ll r){ return mod(l+r); }
ll mult(ll l, ll r){ return mod(l*r); }
ll inverse(ll l){ return inv(l, MOD); }
bool equal(ll l, ll r){ return mod(l-r) == 0; }

template<typename T> struct matrix {
    vector<vector<T>> in;
    int row, col;

    void print(){//
        for (int i = 0; i < row; i++){
            for (int j = 0; j < col; j++)
                cout << in[i][j] << " ";
            cout << endl;
        }
    }

    matrix(int row, int col, int op = 0):row(row), col(col),
       in(row, vector<T>(col, 0)){
        if (op) for (int i = 0; i < row; i++) in[i][i] = 1;
    }
    matrix(initializer_list<initializer_list<T>> c):
        row(c.size()), col((*c.begin()).size()){
            in = vector<vector<T>>(row, vector<T>(col, 0));
            int i, j;
            i = 0;
            for (auto &it : c){
                j = 0;
                for (auto &jt : it){
                    in[i][j] = jt;
                    j++;
                }
                i++;
            }
        }
    T &operator()(int i, int j){ return in[i][j]; }
    //in case of a transposed matrix, swap i and j
    matrix<T>& operator*=(T t){
        matrix<T> &l = *this;
        for (int i = 0; i < row; i++)
            for (int j = 0; j < col; j++)
                l(i, j) = mult(l(i, j), t); //% MOD) % MOD;
        return l;
    }
    matrix<T> operator+(matrix<T> &r){
        matrix<T> &l = *this;
        matrix<T> m(row, col, 0);
        for (int i = 0; i < row; i++)
            for (int j = 0; j < col; j++)
                m(i, j) = sum(l(i, j), r(i, j)); //% MOD) %
                    MOD;
        return m;
    }
    matrix<T> operator*(matrix<T> &r){
        matrix<T> &l = *this;
        int row = l.row;
```

```
    int col = r.col;
    int K = l.col;
    matrix<T> m(row, col, 0);
    for (int i = 0; i < row; i++)
        for (int j = 0; j < col; j++)
            for (int k = 0; k < K; k++)
                m(i, j) = sum(m(i, j), mult(l(i, k),
                    r(k, j)));
    return m;
}
matrix<T> operator^(long long e){
    matrix<T> &m = (*this);
    if (e == 0) return matrix(m.row, m.row, 1);
    if (e == 1) return m;
    if (e == 2) return m*m;
    auto m_ = m^(e/2); m_ = m_*m_;
    if (e%2 == 1) m_ = m_ * m;
    return m_;
}
void multiply_r(int i, T k){
    matrix<T> &m = (*this);
    for (int j = 0; j < col; j++)
        m(i, j) = mult(m(i, j), k);
}
void multiply_c(int j, T k){
    matrix<T> &m = (*this);
    for (int i = 0; i < row; j++)
        m(i, j) = mult(m(i, j), k);
}
void sum_r(int i1, int i2, T k){
    matrix<T> &m = (*this);
    for (int j = 0; j < col; j++)
        m(i1, j) = sum(m(i1, j), mult(k, m(i2, j)));

}
bool gaussian(int I, int J){
    matrix<T> &m = (*this);
    T tmp = m(I, J);
    if (equal(tmp, 0)) return false;
    multiply_r(I, inverse(tmp));
    for (int i = 0; i < row; i++)
        if (i != I) sum_r(i, I, mult(-1, m(i, J)));
        multiply_r(I, tmp);
        return true;
    }
    T determinant(){
        matrix<T> m = (*this);
        for (int i = 0; i < row; i++)
            if (!m.gaussian(i, i)) return 0;

        T ans = 1;
        for (int i = 0; i < row; i++)
            ans = mult(ans, m(i, i));
        return ans;
    }
};
```

# 4 Estruturas

## 4.1 Sparse Table

```
// Resolve RMQ
// MAX2 = log(MAX)
//
// Complexidades:
// build - O(n log(n))
// query - O(1)

namespace sparse {
    int m[MAX2][MAX], n;
    void build(int n2, int* v) {
        n = n2;
        for (int i = 0; i < n; i++) m[0][i] = v[i];
        for (int j = 1; (1<<j) <= n; j++) for (int i = 0;
            i+(1<<j) <= n; i++)
             m[j][i] = min(m[j-1][i], m[j-1][i+(1<<(j-1))]);
    }
    int query(int a, int b) {
        int j = __builtin_clz(0) - __builtin_clz(b-a+1) - 1;
        return min(m[j][a], m[j][b-(1<<j)+1]);
    }
}
```

## 4.2 Treap

```
// Usar static treap<int> t;
// Para usar, chamar o Rafael
//
// Complexidades:
//
// insert - O(log(n))
// erase - O(log(n))
// query - O(log(n))

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

template<typename T> struct treap {
    struct node {
        int p;
        int l, r;
        T v;
        int sz;
        T sum;
        bool rev;
        node(){}
        node(T v):p(rng()), l(-1), r(-1), v(v), sz(1),
            rev(false){}
    } t[MAX];
    int it;
    //vector<node> t;
    treap(){ it = 0; }
    int size(int i){
        if (i == -1) return 0;
        return t[i].sz;
    }
    void fix(int i){
        if (i == -1) return;
        if (t[i].rev) {
            int &l = t[i].l;
            int &r = t[i].r;
            swap(l, r);
            t[i].sz = 1 + size(l) + size(r);
            if (l != -1)
                t[l].rev ^= true;
            if (r != -1)
                t[r].rev ^= true;
            t[i].rev = false;
        }
    }
    void update(int i){
        if (i == -1) return;
        t[i].sum = t[i].v;
        int l = t[i].l;
        int r = t[i].r;
        t[i].sz = 1 + size(l) + size(r);
    }

    void split_value(int i, int k, int &l, int &r){ //values
        must be ordered
        if (i == -1){
            l = -1; r = -1;
            return;
        }
        fix(i);
        if (t[i].v < k){
            split_value(t[i].r, k, l, r);
            t[i].r = l;
            l = i;
        }
        else{
            split_value(t[i].l, k, l, r);
            t[i].l = r;
            r = i;
        }
        update(i);
    }

    //implicit
    void split(int i, int k, int &l, int &r, int sz = 0){
        //key
        if (i == -1){
            l = -1; r = -1;
            return;
        }
        fix(i);
        int inc = size(t[i].l); //quantidade elementos menor
```

```
        que k
    if (sz+inc < k){
        split(t[i].r, k, l, r, sz+inc+1);
        t[i].r = l;
        l = i;
    }
    else{
        split(t[i].l, k, l, r, sz);
        t[i].l = r;
        r = i;
    }
    update(i);
}
int merge(int l, int r){ //priority
    if (l == -1) return r;
    if (r == -1) return l;
    fix(l); fix(r);
    if (t[l].p > t[r].p){
        t[l].r = merge(t[l].r, r);
        update(l);
        return l;
    }
    else{
        t[r].l = merge(l, t[r].l);
        update(r);
        return r;
    }
}

void insert(int &root, T v, int pos){
    int m = it++;
    t[m] = node(v);
    if (root == -1){
        root = m;
        return;
    }
    int l, r;
    split(root, pos, l, r);
    l = merge(l, m);
    l = merge(l, r);
    root = l;
}

    T query(int &root, int M){
        int l, m, r;
        split(root, M+1, m, r);
        split(m, M, l, m);

        T ans = t[m].v;
        l = merge(l, m);
        l = merge(l, r);
        root = l;
        return ans;
    }
    void reverse(int &root, int L, int R){
        int l, m, r;
        split(root, R+1, m, r);
        split(m, L, l, m);
        t[m].rev ^= 1;
        l = merge(l, m);
        l = merge(l, r);
        root = l;
    }
    void print(int i, int &size){
        if (i == -1) return;
        print(t[i].l, size);
        cout << "#" << size << ": " << t[i].v << endl;
        size++;
        print(t[i].r, size);
    }
};
```

## 4.3 SQRT-decomposition

```
// Resolve RMQ
// 0-indexed
// MAX2 = sqrt(MAX)
//
// O bloco da posicao x eh
// sempre x/q
//
// Complexidades:
// build - O(n)
// query - O(sqrt(n))
```

```
int n, q;
int v[MAX];
int bl[MAX2];

void build() {
    q = (int) sqrt(n);

     // computa cada bloco
    for (int i = 0; i <= q; i++) {
        bl[i] = INF;
        for (int j = 0; j < q and q * i + j < n; j++)
            bl[i] = min(bl[i], v[q * i + j]);
    }
}

int query(int a, int b) {
    int ret = INF;

    // linear no bloco de a
    for (; a <= b and a % q; a++) ret = min(ret, v[a]);

    // bloco por bloco
    for (; a + q <= b; a += q) ret = min(ret, bl[a / q]);

    // linear no bloco de b
    for (; a <= b; a++) ret = min(ret, v[a]);

    return ret;
}
```

## 4.4 BIT 2D

```
// BIT de soma 1-based
// Para mudar o valor da posicao (x, y) para k,
// faca: poe(x, y, k - sum(x, y, x, y))
//
// Complexidades:
// poe - O(log^2(n))
// query - O(log^2(n))

int n;
int bit[MAX][MAX];

void poe(int x, int y, int k) {
    for (int y2 = y; x <= n; x += x & -x)
        for (y = y2; y <= n; y += y & -y)
            bit[x][y] += k;
}

int sum(int x, int y) {
    int ret = 0;
    for (int y2 = y; x; x -= x & -x)
        for (y = y2; y; y -= y & -y)
            ret += bit[x][y];

    return ret;
}

int query(int x, int y, int z, int w) {
    return sum(z, w) - sum(x-1, w)
        - sum(z, y-1) + sum(x-1, y-1);
}
```

## 4.5 MergeSort Tree

```
// query(a, b, val) retorna numero de
// elementos em [a, b] <= val
// Usa O(n log(n)) de memoria
//
// Complexidades:
// build - O(n log(n))
// query - O(log^2(n))

#define ALL(x) x.begin(),x.end()

int v[MAX], n;
vector<vector<int> > tree(4*MAX);

void build(int p, int l, int r) {
    if (l == r) return tree[p].push_back(v[l]);
    int m = (l+r)/2;
    build(2*p, l, m), build(2*p+1, m+1, r);
    merge(ALL(tree[2*p]), ALL(tree[2*p+1]),
```

```
        back_inserter(tree[p]));
}

int query(int a, int b, int val, int p=1, int l=0, int
    r=n-1) {
    if (b < l or r < a) return 0; // to fora
    if (a <= l and r <= b) // to totalmente dentro
        return lower_bound(ALL(tree[p]), val+1) -
            tree[p].begin();
    int m = (l+r)/2;
    return query(a, b, val, 2*p, l, m) + query(a, b, val,
        2*p+1, m+1, r);
}
```

## 4.6 SegTree

```
// Recursiva com Lazy Propagation
// Query: soma do range [a, b]
// Update: soma x em cada elemento do range [a, b]
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))

namespace seg {
    ll seg[4*MAX], lazy[4*MAX];
    int n, *v;

    ll build(int p=1, int l=0, int r=n-1) {
        lazy[p] = 0;
        if (l == r) return seg[p] = v[l];
        int m = (l+r)/2;
        return seg[p] = build(2*p, l, m) + build(2*p+1, m+1,
            r);
    }
    void build(int n2, int* v2) {
        n = n2, v = v2;
        build();
    }
    void prop(int p, int l, int r) {
        if (!lazy[p]) return;
        int m = (l+r)/2;
        seg[2*p] += lazy[p]*(m-l+1);
        seg[2*p+1] += lazy[p]*(r-(m+1)+1);
        lazy[2*p] += lazy[p], lazy[2*p+1] += lazy[p];
        lazy[p] = 0;
    }
    ll query(int a, int b, int p=1, int l=0, int r=n-1) {
        if (b < l or r < a) return 0;
        if (a <= l and r <= b) return seg[p];
        prop(p, l, r);
        int m = (l+r)/2;
        return query(a, b, 2*p, l, m) + query(a, b, 2*p+1,
            m+1, r);
    }
    ll update(int a, int b, int x, int p=1, int l=0, int
        r=n-1) {
        if (b < l or r < a) return seg[p];
        if (a <= l and r <= b) {
            seg[p] += (ll)x*(r-l+1);
            lazy[p] += x;
            return seg[p];
        }
        prop(p, l, r);
        int m = (l+r)/2;
        return seg[p] = update(a, b, x, 2*p, l, m) +
            update(a, b, x, 2*p+1, m+1, r);
    }
};
```

## 4.7 SegTree Esparca

```
// Query: soma do range [a, b]
// Update: flipa os valores de [a, b]
//
// Complexidades:
// build - O(n)
// query - O(log^2(n))
// update - O(log^2(n))

typedef long long ll;

namespace seg {
```

```
    unordered_map<ll, int> t, laz;

    void build() { t.clear(), lazy.clear(); }

    void prop(ll p, int l, int r) {
        if (!lazy[p]) return;
        t[p] = r-l+1-t[p];
        if (l != r) lazy[2*p]^=lazy[p], lazy[2*p+1]^=lazy[p];
        lazy[p] = 0;
    }

    int query(int a, int b, ll p=1, int l=0, int r=N-1) {
        prop(p, l, r);
        if (b < l or r < a) return 0;
        if (a <= l and r <= b) return t[p];

        int m = l+r>>1;
        return query(a, b, 2*p, l, m)+query(a, b, 2*p+1,
           m+1, r);
    }

    int update(int a, int b, ll p=1, int l=0, int r=N-1) {
        prop(p, l, r);
        if (b < l or r < a) return t[p];
        if (a <= l and r <= b) {
            lazy[p] ^= 1;
            prop(p, l, r);
            return t[p];
        }
        int m = l+r>>1;
        return t[p] = update(a, b, 2*p, l, m)+update(a, b,
           2*p+1, m+1, r);
    }
};
```

## 4.8 SegTree Iterativa com Lazy Propagation

```
// Query: soma do range [a, b]
// Update: soma x em cada elemento do range [a, b]
// Para mudar, mudar as funcoes junta, poe e query
// LOG = ceil(log2(MAX))
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))

namespace seg {
    ll seg[2*MAX], lazy[2*MAX];
    int n;

    ll junta(ll a, ll b) {
        return a+b;
    }

    // soma x na posicao p de tamanho tam
    void poe(int p, ll x, int tam, bool prop=1) {
        seg[p] += x*tam;
        if (prop and p < n) lazy[p] += x;
    }

    // atualiza todos os pais da folha p
    void sobe(int p) {
        for (int tam = 2; p /= 2; tam *= 2) {
            seg[p] = junta(seg[2*p], seg[2*p+1]);
            poe(p, lazy[p], tam, 0);
        }
    }

    // propaga o caminho da raiz ate a folha p
    void prop(int p) {
        int tam = 1 << (LOG-1);
        for (int s = LOG; s; s--, tam /= 2) {
            int i = p >> s;
            if (lazy[i]) {
                poe(2*i, lazy[i], tam);
                poe(2*i+1, lazy[i], tam);
                lazy[i] = 0;
            }
        }
    }

    void build(int n2, int* v) {
        n = n2;
```

```
        for (int i = 0; i < n; i++) seg[n+i] = v[i];
        for (int i = n-1; i; i--) seg[i] = junta(seg[2*i],
           seg[2*i+1]);
        for (int i = 0; i < 2*n; i++) lazy[i] = 0;
    }

    ll query(int a, int b) {
        ll ret = 0;
        for (prop(a+=n), prop(b+=n); a <= b; ++a/=2, --b/=2)
           {
            if (a%2 == 1) ret = junta(ret, seg[a]);
            if (b%2 == 0) ret = junta(ret, seg[b]);
        }
        return ret;
    }

    void update(int a, int b, int x) {
        int a2 = a += n, b2 = b += n, tam = 1;
        for (; a <= b; ++a/=2, --b/=2, tam *= 2) {
            if (a%2 == 1) poe(a, x, tam);
            if (b%2 == 0) poe(b, x, tam);
        }
        sobe(a2), sobe(b2);
    }
};
```

## 4.9 SegTree Beats

```
// query(a, b) - {{min(v[a..b]), max(v[a..b])}, sum(v[a..b])}
// updatemin(a, b, x) faz com que v[i] <- min(v[i], x),
// para i em [a, b]
// updatemax faz o mesmo com max, e updatesum soma x
// em todo mundo do intervalo [a, b]
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log^2 (n)) amortizado
// (se nao usar updatesum, fica log(n) amortizado)

#define f first
#define s second
typedef long long ll;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;

namespace beats {
    struct node {
        int tam;
        ll sum, lazy; // lazy pra soma
        ll mi1, mi2, mi; // mi = #mi1
        ll ma1, ma2, ma; // ma = #ma1

        node(ll x = 0) {
            sum = mi1 = ma1 = x;
            mi2 = LINF, ma2 = -LINF;
            mi = ma = tam = 1;
            lazy = 0;
        }
        node(const node& l, const node& r) {
            sum = l.sum + r.sum, tam = l.tam + r.tam;
            lazy = 0;
            if (l.mi1 > r.mi1) {
                mi1 = r.mi1, mi = r.mi;
                mi2 = min(l.mi1, r.mi2);
            } else if (l.mi1 < r.mi1) {
                mi1 = l.mi1, mi = l.mi;
                mi2 = min(r.mi1, l.mi2);
            } else {
                mi1 = l.mi1, mi = l.mi+r.mi;
                mi2 = min(l.mi2, r.mi2);
            }
            if (l.ma1 < r.ma1) {
                ma1 = r.ma1, ma = r.ma;
                ma2 = max(l.ma1, r.ma2);
            } else if (l.ma1 > r.ma1) {
                ma1 = l.ma1, ma = l.ma;
                ma2 = max(r.ma1, l.ma2);
            } else {
                ma1 = l.ma1, ma = l.ma+r.ma;
                ma2 = max(l.ma2, r.ma2);
            }
        }
        void setmin(ll x) {
            if (x >= ma1) return;
```

```
        sum += (x - ma1)*ma;
        if (mi1 == ma1) mi1 = x;
        if (mi2 == ma1) mi2 = x;
        ma1 = x;
    }
    void setmax(ll x) {
        if (x <= mi1) return;
        sum += (x - mi1)*mi;
        if (ma1 == mi1) ma1 = x;
        if (ma2 == mi1) ma2 = x;
        mi1 = x;
    }
    void setsum(ll x) {
        mi1 += x, mi2 += x, ma1 += x, ma2 += x;
        sum += x*tam;
        lazy += x;
    }
};

node seg[4*MAX];
int n, *v;

node build(int p=1, int l=0, int r=n-1) {
    if (l == r) return seg[p] = {v[l]};
    int m = (l+r)/2;
    return seg[p] = {build(2*p, l, m), build(2*p+1, m+1,
       r)};
}
void build(int n2, int* v2) {
    n = n2, v = v2;
    build();
}
void prop(int p, int l, int r) {
    if (l == r) return;
    for (int k = 0; k < 2; k++) {
        if (seg[p].lazy) seg[2*p+k].setsum(seg[p].lazy);
        seg[2*p+k].setmin(seg[p].ma1);
        seg[2*p+k].setmax(seg[p].mi1);
    }
    seg[p].lazy = 0;
}
pair<pair<ll, ll>, ll> query(int a, int b, int p=1, int
   l=0, int r=n-1) {
    if (b < l or r < a) return {{INF, -INF}, 0};
    if (a <= l and r <= b) return {{seg[p].mi1,
       seg[p].ma1}, seg[p].sum};
    prop(p, l, r);
    int m = (l+r)/2;
    auto L = query(a, b, 2*p, l, m), R = query(a, b,
       2*p+1, m+1, r);
    return {{min(L.f.f, R.f.f), max(L.f.s, R.f.s)},
       L.s+R.s};
}
node updatemin(int a, int b, ll x, int p=1, int l=0, int
   r=n-1) {
    if (b < l or r < a or seg[p].ma1 <= x) return seg[p];
    if (a <= l and r <= b and seg[p].ma2 < x) {
        seg[p].setmin(x);
        return seg[p];
    }
    prop(p, l, r);
    int m = (l+r)/2;
    return seg[p] = {updatemin(a, b, x, 2*p, l, m),
                     updatemin(a, b, x, 2*p+1, m+1, r)};
}
node updatemax(int a, int b, ll x, int p=1, int l=0, int
   r=n-1) {
    if (b < l or r < a or seg[p].mi1 >= x) return seg[p];
    if (a <= l and r <= b and seg[p].mi2 > x) {
        seg[p].setmax(x);
        return seg[p];
    }
    prop(p, l, r);
    int m = (l+r)/2;
    return seg[p] = {updatemax(a, b, x, 2*p, l, m),
                     updatemax(a, b, x, 2*p+1, m+1, r)};
}
node updatesum(int a, int b, ll x, int p=1, int l=0, int
   r=n-1) {
    if (b < l or r < a) return seg[p];
    if (a <= l and r <= b) {
        seg[p].setsum(x);
        return seg[p];
    }
```

```
        prop(p, l, r);
        int m = (l+r)/2;
        return seg[p] = {updatesum(a, b, x, 2*p, l, m),
                         updatesum(a, b, x, 2*p+1, m+1, r)};
    }
};
```

## 4.10 SegTree Iterativa

```
// Consultas 0-based
// Valores iniciais devem estar em (seg[n], ... , seg[2*n-1])
// Query: soma do range [a, b]
// Update: muda o valor da posicao p para x
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))

int seg[2 * MAX];
int n;

void build() {
    for (int i = n - 1; i; i--) seg[i] = seg[2*i] +
       seg[2*i+1];
}

int query(int a, int b) {
    int ret = 0;
    for(a += n, b += n; a <= b; ++a /= 2, --b /= 2) {
        if (a % 2 == 1) ret += seg[a];
        if (b % 2 == 0) ret += seg[b];
    }
    return ret;
}

void update(int p, int x) {
    seg[p += n] = x;
    while (p /= 2) seg[p] = seg[2*p] + seg[2*p+1];
}
```

## 4.11 SegTree Persistente

```
// SegTree de soma, update de somar numa posicao
//
// query(a, b, t) retorna a query de [a, b] depois de
// t updates
// update(a, x, t) faz um update v[a]+=x a partir de
// como era depois de t updates
//
// build - O(n)
// query - O(log(n))
// update - O(log(n))

const int MAX = 3e4+10, UPD = 2e5+10, LOG = 20;
const int MAXS = 4*MAX+UPD*LOG;

namespace perseg {
    ll seg[MAXS], lazy[MAXS];
    int T[UPD], L[MAXS], R[MAXS], cnt, t;
    int n, *v;

    ll build(int p, int l, int r) {
        if (l == r) return seg[p] = v[l];
        L[p] = cnt++, R[p] = cnt++;
        int m = (l+r)/2;
        return seg[p] = build(L[p], l, m) + build(R[p], m+1,
           r);
    }
    void build(int n2, int* v2) {
        n = n2, v = v2;
        T[0] = cnt++;
        build(0, 0, n-1);
    }
    ll query(int a, int b, int p, int l, int r) {
        if (b < l or r < a) return 0;
        if (a <= l and r <= b) return seg[p];
        int m = (l+r)/2;
        return query(a, b, L[p], l, m) + query(a, b, R[p],
           m+1, r);
    }
    ll query(int a, int b, int tt) {
        return query(a, b, T[tt], 0, n-1);
```

```
    }
    ll update(int a, int x, int lp, int p, int l, int r) {
        if (l == r) return seg[p] = seg[lp]+x;

        int m = (l+r)/2;
        if (a <= m)
            return seg[p] = update(a, x, L[lp], L[p]=cnt++,
                l, m) + seg[R[p]=R[lp]];
        return seg[p] = seg[L[p]=L[lp]] + update(a, x,
            R[lp], R[p]=cnt++, m+1, r);
    }
    void update(int a, int x, int tt=t) {
        update(a, x, T[tt], T[++t]=cnt++, 0, n-1);
    }
};
```

## 4.12 SegTree 2D Iterativa

```
// Consultas 0-based
// Um valor inicial em (x, y) deve ser colocado em
   seg[x+n][y+n]
// Query: soma do retangulo ((x1, y1), (x2, y2))
// Update: muda o valor da posicao (x, y) para val
// Nao pergunte como que essa coisa funciona
//
// Para query com distancia de manhattan <= d, faca
// nx = x+y, ny = x-y
// Update em (nx, ny), query em ((nx-d, ny-d), (nx+d, ny+d))
//
// Se for de min/max, pode tirar os if's da 'query', e fazer
// sempre as 4 operacoes. Fica mais rapido
//
// Complexidades:
// build - O(n^2)
// query - O(log^2(n))
// update - O(log^2(n))

int seg[2*MAX][2*MAX], n;

void build() {
    for (int x = 2*n; x; x--) for (int y = 2*n; y; y--) {
        if (x < n) seg[x][y] = seg[2*x][y] + seg[2*x+1][y];
        if (y < n) seg[x][y] = seg[x][2*y] + seg[x][2*y+1];
    }
}

int query(int x1, int y1, int x2, int y2) {
    int ret = 0, y3 = y1 + n, y4 = y2 + n;
    for (x1 += n, x2 += n; x1 <= x2; ++x1 /= 2, --x2 /= 2)
        for (y1 = y3, y2 = y4; y1 <= y2; ++y1 /= 2, --y2 /=
           2) {
            if (x1%2 == 1 and y1%2 == 1) ret += seg[x1][y1];
            if (x1%2 == 1 and y2%2 == 0) ret += seg[x1][y2];
            if (x2%2 == 0 and y1%2 == 1) ret += seg[x2][y1];
            if (x2%2 == 0 and y2%2 == 0) ret += seg[x2][y2];
        }

    return ret;
}

void update(int x, int y, int val) {
    int y2 = y += n;
    for (x += n; x; x /= 2, y = y2) {
        if (x >= n) seg[x][y] = val;
        else seg[x][y] = seg[2*x][y] + seg[2*x+1][y];

        while (y /= 2) seg[x][y] = seg[x][2*y] +
           seg[x][2*y+1];
    }
}
```

## 4.13 BIT

```
// BIT de soma 1-based, v 0-based
// Para mudar o valor da posicao p para x,
// faca: poe(x - query(p, p), p)
// l_bound(x) retorna o menor p tal que
// query(1, p+1) > x    (0 based!)
//
// Complexidades:
// build - O(n)
// poe - O(log(n))
// query - O(log(n))
// l_bound - O(log(n))
```

```
int n;
int bit[MAX];
int v[MAX];

void build() {
    bit[0] = 0;
    for (int i = 1; i <= n; i++) bit[i] = v[i - 1];

    for (int i = 1; i <= n; i++) {
        int j = i + (i & -i);
        if (j <= n) bit[j] += bit[i];
    }
}

// soma x na posicao p
void poe(int x, int p) {
    for (; p <= n; p += p & -p) bit[p] += x;
}

// soma [1, p]
int pref(int p) {
    int ret = 0;
    for (; p; p -= p & -p) ret += bit[p];
    return ret;
}

// soma [a, b]
int query(int a, int b) {
    return pref(b) - pref(a - 1);
}

int l_bound(ll x) {
    int p = 0;
    for (int i = MAX2; i+1; i--) if (p + (1<<i) <= n
        and bit[p + (1<<i)] <= x) x -= bit[p += (1<<i)];
    return p;
}
```

## 4.14 Order Statistic Set

```
// Funciona do C++11 pra cima
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <class T>
    using ord_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

// para declarar:
ord_set<int> s;
// coisas do set normal funcionam:
for (auto i : s) cout << i << endl;
cout << s.size() << endl;
// k-esimo maior elemento O(log|s|):
// k=0: menor elemento
cout << *s.find_by_order(k) << endl;
// quantos sao menores do que k O(log|s|):
cout << s.order_of_key(k) << endl;

// Para fazer um multiset, tem que
// usar ord_set<pair<int, int> > com o
// segundo parametro sendo algo para diferenciar
// os ementos iguais.
// s.order_of_key({k, -INF}) vai retornar o
// numero de elementos < k
```

## 4.15 Min queue

```
// Tudo O(1) amortizado

template<class T> struct minstack {
    stack<pair<T, T> > s;

    void push(T x) {
        if (!s.size()) s.push({x, x});
        else s.push({x, std::min(s.top().second, x)});
    }
    T top() { return s.top().first; }
    T pop() {
        T ans = s.top().first;
        s.pop();
        return ans;
```

```
    }
    T size() { return s.size(); }
    T min() { return s.top().second; }
};

template<class T> struct minqueue {
    minstack<T> s1, s2;

    void push(T x) { s1.push(x); }
    void move() {
        if (s2.size()) return;
        while (s1.size()) {
            T x = s1.pop();
            s2.push(x);
        }
    }
    T front() { return move(), s2.top(); }
    T pop() { return move(), s2.pop(); }
    T size() { return s1.size()+s2.size(); }
    T min() {
        if (!s1.size()) return s2.min();
        else if (!s2.size()) return s1.min();
        return std::min(s1.min(), s2.min());
    }
};
```

## 4.16 DSU Persistente

```
// Complexidades:
// build - O(n)
// find - O(log(n))
// une - O(log(n))

int n, p[MAX], sz[MAX], ti[MAX];

void build() {
    for (int i = 0; i < n; i++) {
        p[i] = i;
        sz[i] = 1;
        ti[i] = -INF;
    }
}

int find(int k, int t) {
    if (p[k] == k or ti[k] > t) return k;
    return find(p[k], t);
}

void une(int a, int b, int t) {
    a = find(a); b = find(b);
    if (a == b) return;
    if (sz[a] > sz[b]) swap(a, b);

    sz[b] += sz[a];
    p[a] = b;
    ti[a] = t;
}
```

## 4.17 SQRT Tree

```
// RMQ em O(log log n) com O(n log log n) pra buildar
// Funciona com qualquer operacao associativa
// Tao rapido quanto a sparse table, mas usa menos memoria
// (log log (1e9) < 5, entao a query eh praticamente O(1))
//
// build - O(n log log n)
// query - O(log log n)

namespace sqrtTree {
    int n, *v;
    int pref[4][MAX], sulf[4][MAX], getl[4][MAX],
       entre[4][MAX], sz[4];

    int op(int a, int b) { return min(a, b); }
    inline int getblk(int p, int i) { return
       (i-getl[p][i])/sz[p]; }
    void build(int p, int l, int r) {
        if (l+1 >= r) return;
        for (int i = l; i <= r; i++) getl[p][i] = l;
        for (int L = l; L <= r; L += sz[p]) {
            int R = min(L+sz[p]-1, r);
            pref[p][L] = v[L], sulf[p][R] = v[R];
            for (int i = L+1; i <= R; i++) pref[p][i] =
               op(pref[p][i-1], v[i]);
```

```
            for (int i = R-1; i >= L; i--) sulf[p][i] =
                op(v[i], sulf[p][i+1]);
            build(p+1, L, R);
        }
        for (int i = 0; i <= sz[p]; i++) {
            int at = entre[p][l+i*sz[p]+i] =
                sulf[p][l+i*sz[p]];
            for (int j = i+1; j <= sz[p]; j++)
                entre[p][l+i*sz[p]+j] = at =
                    op(at, sulf[p][l+j*sz[p]]);
        }
    }
    void build(int n2, int* v2) {
        n = n2, v = v2;
        for (int p = 0; p < 4; p++) sz[p] = n2 = sqrt(n2);
        build(0, 0, n-1);
    }
    int query(int l, int r) {
        if (l+1 >= r) return l == r ? v[l] : op(v[l], v[r]);
        int p = 0;
        while (getblk(p, l) == getblk(p, r)) p++;
        int ans = sulf[p][l], a = getblk(p, l)+1, b =
            getblk(p, r)-1;
        if (a <= b) ans = op(ans,
            entre[p][getl[p][l]+a*sz[p]+b]);
        return op(ans, pref[p][r]);
    }
}
```

## 4.18 Wavelet Tree

```
// Usa O(sigma + n log(sigma)) de memoria,
// onde sigma = MAXN - MINN
// Depois do build, o v fica ordenado
// count(i, j, x, y) retorna o numero de elementos de
// v[i, j) que pertencem a [x, y]
// kth(i, j, k) retorna o elemento que estaria
// na poscicao k-1 de v[i, j), se ele fosse ordenado
// sum(i, j, x, y) retorna a soma dos elementos de
// v[i, j) que pertencem a [x, y]
// sumk(i, j, k) retorna a soma dos k-esimos menores
// elementos de v[i, j) (sum(i, j, 1) retorna o menor)
//
// Complexidades:
// build - O(n log(sigma))
// count - O(log(sigma))
// kth   - O(log(sigma))
// sum   - O(log(sigma))
// sumk  - O(log(sigma))

int n, v[MAX];
vector<vector<int> > esq(4*(MAXN-MINN)), pref(4*(MAXN-MINN));

void build(int b = 0, int e = n, int p = 1, int l = MINN,
    int r = MAXN) {
    int m = (l+r)/2; esq[p].push_back(0);
        pref[p].push_back(0);
    for (int i = b; i < e; i++) {
        esq[p].push_back(esq[p].back()+(v[i]<=m));
        pref[p].push_back(pref[p].back()+v[i]);
    }
    if (l == r) return;
    int m2 = stable_partition(v+b, v+e, [=](int i){return i
        <= m;}) - v;
    build(b, m2, 2*p, l, m), build(m2, e, 2*p+1, m+1, r);
}

int count(int i, int j, int x, int y, int p = 1, int l =
    MINN, int r = MAXN) {
    if (y < l or r < x) return 0;
    if (x <= l and r <= y) return j-i;
    int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
    return count(ei, ej, x, y, 2*p, l, m)+count(i-ei, j-ej,
        x, y, 2*p+1, m+1, r);
}

int kth(int i, int j, int k, int p=1, int l = MINN, int r =
    MAXN) {
    if (l == r) return l;
    int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
    if (k <= ej-ei) return kth(ei, ej, k, 2*p, l, m);
    return kth(i-ei, j-ej, k-(ej-ei), 2*p+1, m+1, r);
}
```

```
int sum(int i, int j, int x, int y, int p = 1, int l = MINN,
    int r = MAXN) {
    if (y < l or r < x) return 0;
    if (x <= l and r <= y) return pref[p][j]-pref[p][i];
    int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
    return sum(ei, ej, x, y, 2*p, l, m) + sum(i-ei, j-ej, x,
        y, 2*p+1, m+1, r);
}

int sumk(int i, int j, int k, int p = 1, int l = MINN, int r
    = MAXN) {
    if (l == r) return l*k;
    int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
    if (k <= ej-ei) return sumk(ei, ej, k, 2*p, l, m);
    return pref[2*p][ej]-pref[2*p][ei]+sumk(i-ei, j-ej,
        k-(ej-ei), 2*p+1, m+1, r);
}
```

## 4.19 Trie

```
// N deve ser maior ou igual ao numero de nos da trie
// fim indica se alguma palavra acaba nesse no
//
// Complexidade:
// Inserir e conferir string S -> O(|S|)

// usar static trie T
// T.insert(s) para inserir
// T.find(s) para ver se ta
// T.prefix(s) printa as strings
// que tem s como prefixo

struct trie{
    map<char, int> t[MAX+5];
    int p;
    trie(){
        p = 1;
    }
    void insert(string s){
        s += '$';
        int i = 0;
        for (char c : s){
            auto it = t[i].find(c);
            if (it == t[i].end())
                i = t[i][c] = p++;
            else
                i = it->second;
        }
    }
    bool find(string s){
        s += '$';
        int i = 0;
        for (char c : s){
            auto it = t[i].find(c);
            if (it == t[i].end()) return false;
            i = it->second;
        }
        return true;
    }
    void prefix(string &l, int i){
        if (t[i].find('$') != t[i].end())
            cout << "  " << l << endl;
        for (auto p : t[i]){
            l += p.first;
            prefix(l, p.second, k);
            l.pop_back();
        }
    }
    void prefix(string s){
        int i = 0;
        for (char c : s){
            auto it = t[i].find(c);
            if (it == t[i].end()) return;
            i = it->second;
        }
        int k = 0;
        prefix(s, i, k);
    }
};
```

# 5 Papa

## 5.1 BIT Persistente

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

typedef long long int ll;

const ll LINF = 0x3f3f3f3f3f3f3f3fll;

#define MAXN 100010
vector<pair<int,ll> > FT[MAXN];
int n;
void clear()
{
    for(int i=1;i<=n;i++)
    {
        FT[i].clear();
        FT[i].push_back({-1,0});
    }
}
void add(int i,int v,int time)
{
    for(;i<=n;i+=i&(-i))
    {
        ll last=FT[i].back().second;
        FT[i].push_back({time,last+v});
    }
}
ll get(int i,int time)
{
    ll ret=0;
    for(;i>0;i-=i&(-i))
    {
        int pos = upper_bound(FT[i].begin(),FT[i].end(),
                    make_pair(time,LINF))-FT[i].begin()-1;
        ret+=FT[i][pos].second;
    }
    return ret;
}
ll getRange(int a,int b,int time)
{
    return get(b,time)-get(a-1,time);
}
```

## 5.2 Baby step Giant step

```
// Resolve Logaritmo Discreto a^x = b mod m, m primo em
   O(sqrt(n)*hash(n))
// Meet In The Middle, decompondo x = i * ceil(sqrt(n)) -j,
   i,j<=ceil(sqrt(n))

int babyStep(int a,int b,int m)
{
    unordered_map<int,int> mapp;
    int sq=sqrt(m)+1;
    ll asq=1;
    for(int i=0; i<sq; i++)
        asq=(asq*a)%m;
    ll curr=asq;
    for(int i=1; i<=sq; i++)
    {
        if(!mapp.count(curr))
            mapp[curr]=i;
        curr=(curr*asq)%m;
    }
    int ret=INF;
    curr=b;
    for(int j=0; j<=sq; j++)
    {
        if(mapp.count(curr))
            ret=min(ret,(int)(mapp[curr]*sq-j));
        curr=(curr*a)%m;
    }
    if(ret<INF) return ret;
    return -1;
}
int main()
{
    int a,b,m;
    while(cin>>a>>b>>m,a or b or m)
    {
```

```
        int x=babyStep(a,m,b);
        if(x!=-1)
            cout<<x<<endl;
        else
            cout<<"No Solution"<<endl;
    }
    return 0;
}
```

## 5.3 LIS Rec. Resp.

```
#include<bits/stdc++.h>
using namespace std;
#define sc(a) scanf("%d", &a)

typedef long long int ll;
const int INF = 0x3f3f3f3f;

#define MAXN 100100
int aux[MAXN],endLis[MAXN];
//usar upper_bound se puder >=
vector<int> LisRec(vector<int> v){
    int n=v.size();
    int lis=0;
    for (int i = 0; i < n; i++){
        int it = lower_bound(aux, aux+lis, v[i]) - aux;
        endLis[i] = it+1;
        lis = max(lis, it+1);
        aux[it] = v[i];
    }
    vector<int> resp;
    int prev=INF;
    for(int i=n-1;i>=0;i--){
        if(endLis[i]==lis && v[i]<=prev){
            lis--;
            prev=v[i];
            resp.push_back(i);
        }
    }
    reverse(resp.begin(),resp.end());
    return resp;
}

int main()
{
    int n;
    sc(n);
    vector<int> v(n);
    for(int i=0;i<n;i++)
        sc(v[i]);
    cout<<LisRec(v).size()<<endl;
    return 0;
}
```

## 5.4 Aho Corasick

```
const int N=100010;
const int M=26;
//N= tamanho da trie, M tamanho do alfabeto
int to[N][M], Link[N], fim[N];
int idx = 1;
void add_str(string &s)
{
    int v = 0;
    for (int i = 0; i < s.size(); i++) {
        if (!to[v][s[i]]) to[v][s[i]] = idx++;
        v = to[v][s[i]];
    }
    fim[v] = 1;
}

void process()
{
    queue<int> fila;
    fila.push(0);
    while (!fila.empty()) {
        int cur = fila.front();
        fila.pop();
        int l = Link[cur];
        fim[cur] |= fim[l];
        for (int i = 0; i < M; i++) {
            if (to[cur][i]) {
                if (cur != 0) {
                    Link[to[cur][i]] = to[l][i];
```

```
                }
                else
                    Link[to[cur][i]] = 0;
                fila.push(to[cur][i]);
            }
            else {
                to[cur][i] = to[l][i];
            }
        }
    }
}

int resolve(string &s)
{
    int v = 0, r = 0;
    for (int i = 0; i < s.size(); i++) {
        v = to[v][s[i]];
        if (fim[v]) r++, v = 0;
    }
    return r;
}
```

# 6 Problemas

## 6.1 Inversion Count

```
// O(n log(n))

int n;
int v[MAX];

// bit de soma
void poe(int p);
int query(int p);

// converte valores do array pra
// numeros de 1 a n
void conv() {
    vector<int> a;
    for (int i = 0; i < n; i++) a.push_back(v[i]);
    sort(a.begin(), a.end());

    for (int i = 0; i < n; i++)
        v[i] = 1 + (lower_bound(a.begin(), a.end(), v[i]) -
            a.begin());
}

long long inv() {
    conv();
    build();

    long long ret = 0;
    for (int i = n - 1; i >= 0; i--) {
        ret += query(v[i] - 1);
        poe(v[i]);
    }
    return ret;
}
```

## 6.2 Merge Sort Rafael

```
// Melhor do Brasil, segundo o autor
//
// O(n log(n))

long long merge_sort(int l, int r, vector<int> &t){
    if (l >= r) return 0;
    int m = (l+r)/2;
    auto ans = merge_sort(l, m, t) + merge_sort(m+1, r, t);
    static vector<int> aux; if (aux.size() != t.size())
       aux.resize(t.size());
    for (int i = l; i <= r; i++) aux[i] = t[i];

    int i_l = l, i_r = m+1, i = l;
    auto move_l = [&](){
        t[i++] = aux[i_l++];
    };
    auto move_r = [&](){
        t[i++] = aux[i_r++];
    };
```

```
    while (i <= r){
        if (i_l > m) move_r();
        else if (i_r > r) move_l();
        else{
            if (aux[i_l] <= aux[i_r]) move_l();
            else{
                move_r();
                ans += m - i_l + 1;
            }

        }
    }
    return ans;
}

//inversions to turn r into l
template<typename T> ll inv_count(vector<T> &l, vector<T>
   &r){
    int n = l.size();
    map<T, int> occ;
    map<pair<T, int>, int> rk;
    for (int i = 0; i < n; i++)
        rk[make_pair(l[i], occ[l[i]]++)] = i;
    occ.clear();
    vector<int> v(n);
    for (int i = 0; i < n; i++)
        v[i] = rk[make_pair(r[i], occ[r[i]]++)];
    return merge_sort(0, n-1, v);
}
```

## 6.3 RMQ com Divide and Conquer

```
// Responde todas as queries em
// O(n log(n))

typedef pair<pair<int, int>, int> iii;
#define f first
#define s second

int n, q, v[MAX];
iii qu[MAX];
int ans[MAX], pref[MAX], sulf[MAX];

void solve(int l=0, int r=n-1, int ql=0, int qr=q-1) {
    if (l > r or ql > qr) return;
    int m = (l+r)/2;
    int qL = partition(qu+ql, qu+qr+1, [=](iii x){return
       x.f.s < m;}) - qu;
    int qR = partition(qu+qL, qu+qr+1, [=](iii x){return
       x.f.f <=m;}) - qu;

    pref[m] = sulf[m] = v[m];
    for (int i = m-1; i >= l; i--) pref[i] = min(v[i],
       pref[i+1]);
    for (int i = m+1; i <= r; i++) sulf[i] = min(v[i],
       sulf[i-1]);

    for (int i = qL; i < qR; i++)
        ans[qu[i].s] = min(pref[qu[i].f.f], sulf[qu[i].f.s]);

    solve(l, m-1, ql, qL-1), solve(m+1, r, qR, qr);
}
```

## 6.4 SOS DP

```
// O(n 2^n)

//iterative version
for(int mask = 0; mask < (1<<N); ++mask){
    dp[mask][-1] = A[mask]; //handle base case separately
       (leaf states)
    for(int i = 0;i < N; ++i){
        if(mask & (1<<i))
            dp[mask][i] = dp[mask][i-1] +
               dp[mask^(1<<i)][i-1];
        else dp[mask][i] = dp[mask][i-1];
    }
    F[mask] = dp[mask][N-1];
}

//memory optimized, super easy to code.
for(int i = 0; i<(1<<N); ++i) F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N);
   ++mask){
```

```
    if(mask & (1<<i))
        F[mask] += F[mask^(1<<i)];
}
```

## 6.5 LIS2

```
// O(n log(n))

template<typename T> int lis(vector<T> &v){
    vector<T> ans;
    for (T t : v){
        auto it = upper_bound(ans.begin(), ans.end(), t);
        if (it == ans.end()) ans.push_back(t);
        else *it = t;
    }
    return ans.size()
}
```

## 6.6 Convex Hull Trick (Rafael)

```
// linear

struct CHT {
    int it;
    vector<ll> a, b;
    CHT():it(0){}
    ll eval(int i, ll x){
        return a[i]*x + b[i];
    }
    bool useless(){
        int sz = a.size();
        int r = sz-1, m = sz-2, l = sz-3;
        return  (b[l] - b[r])*(a[m] - a[l]) <
            (b[l] - b[m])*(a[r] - a[l]);
    }
    void add(ll A, ll B){
        a.push_back(A); b.push_back(B);
        while (!a.empty()){
            if ((a.size() < 3) || !useless()) break;
            a.erase(a.end() - 2);
            b.erase(b.end() - 2);
        }
    }
    ll get(ll x){
        it = min(it, int(a.size()) - 1);
        while (it+1 < a.size()){
            if (eval(it+1, x) > eval(it, x)) it++;
            else break;
        }
        return eval(it, x);
    }
};
```

## 6.7 Mininum Enclosing Circle Vasek

```
// O(n) com alta probabilidade

const long double EPS = 1e-12;

struct pt {
    long double x, y;
    pt() {}
    pt(long double x, long double y) : x(x), y(y) {}
    pt(const pt& p) : x(p.x), y(p.y) {}
    pt operator + (const pt& p) const { return pt(x+p.x,
       y+p.y); }
    pt operator - (const pt& p) const { return pt(x-p.x,
       y-p.y); }
    pt operator * (long double c) const { return pt(x*c, y*c
       ); }
    pt operator / (long double c) const { return pt(x/c, y/c
       ); }
};

long double dot(pt p, pt q) { return p.x*q.x+p.y*q.y; }
long double dist2(pt p, pt q) { return dot(p-q, p-q); }
long double cross(pt p, pt q) { return p.x*q.y-p.y*q.x; }

pt rotate90(pt p) { return pt(p.y, -p.x); }

pt interline(pt a, pt b, pt c, pt d) {
    b = b-a; d = c-d; c = c-a;
    return a+b*cross(c, d)/cross(b, d);
```

```
}

pt center(pt a, pt b, pt c) {
    b = (a+b)/2;
    c = (a+c)/2;
    return interline(b, b+rotate90(a-b), c, c+rotate90(a-c));
}

struct circle {
    pt cen;
    long double r;
    circle() {}
    circle(pt cen, long double r) : cen(cen), r(r) {}
};

bool inside(circle& c, pt& p) {
    return c.r*c.r+1e-9 > dist2(p, c.cen);
}

pt bestof3(pt a, pt b, pt c) {
    if (dot(b-a, c-a) < 1e-9) return (b+c)/2;
    if (dot(a-b, c-b) < 1e-9) return (a+c)/2;
    if (dot(a-c, b-c) < 1e-9) return (a+b)/2;
    return center(a, b, c);
}

circle minCirc(vector<pt> v) {
    int n = v.size();
    random_shuffle(v.begin(), v.end());
    pt p = pt(0, 0);
    circle ret = circle(p, 0);
    for (int i = 0; i < n; i++) if (!inside(ret, v[i])) {
        ret = circle(v[i], 0);
        for (int j = 0; j < i; j++) if (!inside(ret, v[j])) {
            ret = circle((v[i]+v[j])/2, sqrt(dist2(v[i],
                v[j]))/2);
            for (int k = 0; k < j; k++) if (!inside(ret,
                v[k])) {
                p = bestof3(v[i], v[j], v[k]);
                ret = circle(p, sqrt(dist2(p, v[i])));
            }
        }
    }
    return ret;
}
```

## 6.8 Nim

```
// Calcula movimento otimo do jogo classico de Nim
// Assume que o estado atual eh perdedor
// Funcao move retorna um par com a pilha (0 indexed)
// e quanto deve ser tirado dela
// XOR deve estar armazenado em x
// Para mudar um valor, faca insere(novo_valor),
// atualize o XOR e mude o valor em v
//
// MAX2 = teto do log do maior elemento
// possivel nas pilhas
//
// O(log(n)) amortizado

int v[MAX], n, x;
stack<int> pi[MAX2];

void insere(int p) {
    for (int i = 0; i < MAX2; i++) if (v[p] & (1 << i))
        pi[i].push(p);
}

pair<int, int> move() {
    int bit = 0; while (x >> bit) bit++; bit--;

    // tira os caras invalidos
    while ((v[pi[bit].top()] & (1 << bit)) == 0)
        pi[bit].pop();

    int cara = pi[bit].top();
    int tirei = v[cara] - (x^v[cara]);
    v[cara] -= tirei;

    insere(cara);

    return make_pair(cara, tirei);
}
```

```
// Acha o movimento otimo baseado
// em v apenas
//
// O(n)

pair<int, int> move() {
    int x = 0;
    for (int i = 0; i < n; i++) x ^= v[i];

    for (int i = 0; i < n; i++) if ((v[i]^x) < v[i])
        return make_pair(i, v[i] - (v[i]^x));
}
```

## 6.9 Distinct Range Query com Update

```
// build - O(n log^2(n))
// query - O(log^2(n))
// update - O(log^2(n))

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <class T>
    using ord_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

int v[MAX], n, nxt[MAX], prv[MAX];
map<int, set<int> > ocor;

namespace seg {
    ord_set<ii> seg[4*MAX];

    void build(int p=1, int l=0, int r=n-1) {
        if (l == r) return (void)seg[p].insert({nxt[l], l});
        int m = (l+r)/2;
        build(2*p, l, m), build(2*p+1, m+1, r);
        for (ii i : seg[2*p]) seg[p].insert(i);
        for (ii i : seg[2*p+1]) seg[p].insert(i);
    }
    int query(int a, int b, int x, int p=1, int l=0, int
       r=n-1) {
        if (b < l or r < a) return 0;
        if (a <= l and r <= b) return
           seg[p].order_of_key({x, -INF});
        int m = (l+r)/2;
        return query(a, b, x, 2*p, l, m)+query(a, b, x,
           2*p+1, m+1, r);
    }
    void update(int a, int x, int p=1, int l=0, int r=n-1) {
        if (a < l or r < a) return;
        seg[p].erase({nxt[a], a});
        seg[p].insert({x, a});
        if (l == r) return;
        int m = (l+r)/2;
        update(a, x, 2*p, l, m), update(a, x, 2*p+1, m+1, r);
    }
}

void build() {
    for (int i = 0; i < n; i++) nxt[i] = INF;
    for (int i = 0; i < n; i++) prv[i] = -INF;
    vector<ii> t;
    for (int i = 0; i < n; i++) t.push_back({v[i], i});
    sort(t.begin(), t.end());
    for (int i = 0; i < n; i++) {
        if (i and t[i].f == t[i-1].f) prv[t[i].s] = t[i-1].s;
        if (i+1 < n and t[i].f == t[i+1].f) nxt[t[i].s] =
           t[i+1].s;
    }

    for (int i = 0; i < n; i++) ocor[v[i]].insert(i);

    seg::build();
}

void muda(int p, int x) {
    seg::update(p, x);
    nxt[p] = x;
}

int query(int a, int b) {
    return b-a+1 - seg::query(a, b, b+1);
}
```

```
void update(int p, int x) { // mudar valor na pos. p para x
    if (prv[p] > -INF) muda(prv[p], nxt[p]);
    if (nxt[p] < INF) prv[nxt[p]] = prv[p];

    ocor[v[p]].erase(p);
    if (!ocor[x].size()) {
        muda(p, INF);
        prv[p] = -INF;
    } else if (*ocor[x].rbegin() < p) {
        int i = *ocor[x].rbegin();
        prv[p] = i;
        muda(p, INF);
        muda(i, p);
    } else {
        int i = *ocor[x].lower_bound(p);
        if (prv[i] > -INF) {
            muda(prv[i], p);
            prv[p] = prv[i];
        } else prv[p] = -INF;
        prv[i] = p;
        muda(p, i);
    }
    v[p] = x; ocor[x].insert(p);
}
```

## 6.10 LIS1

```
// Calcula uma LIS
// Para ter o tamanho basta fazer lis().size()
// Implementacao do algotitmo descrito em:
// https://goo.gl/HiFkn2
//
// O(n log(n))

const int INF = 0x3f3f3f3f;

int n, v[MAX];

vector<int> lis() {
    int I[n + 1], L[n];

    // pra BB funfar bacana
    I[0] = -INF;
    for (int i = 1; i <= n; i++) I[i] = INF;

    for (int i = 0; i < n; i++) {
        // BB
        int l = 0, r = n;
        while (l < r) {
            int m = (l + r) / 2;
            if (I[m] >= v[i]) r = m;
            else l = m + 1;
        }

        // ultimo elemento com tamanho l eh v[i]
        I[l] = v[i];
        // tamanho da LIS terminando com o
        // elemento v[i] eh l
        L[i] = l;
    }

    // reconstroi LIS
    vector<int> ret;
    int m = -INF, p;
    for (int i = 0; i < n; i++) if (L[i] > m) {
        m = L[i];
        p = i;
    }
    ret.push_back(v[p]);
    int last = m;
    while (p--) if (L[p] == m - 1) {
        ret.push_back(v[p]);
        m = L[p];
    }

    reverse(ret.begin(), ret.end());
    return ret;
}
```

## 6.11 Mo algorithm - distinct values

```
// O(s*n*f + q*(n/s)*f) optimize over s, insert/erase = O(f)
// for s = sqrt(n), O((n+q)*sqrt(n)*f)
```

```
const int MAX = 3e4+10;
const int SQ = sqrt(MAX);
int v[MAX];

int ans, freq[MAX];

void insert(int p){
    int o = v[p];
    freq[o]++;
    ans += (freq[o] == 1);
}
void erase(int p){
    int o = v[p];
    ans -= (freq[o] == 1);
    freq[o]--;
}

vector<int> MO(vector<ii> &q){
    ans = 0;
    memset(freq, 0, sizeof freq);
    int m = q.size();
    vector<int> ord(m), ret(m);
    iota(ord.begin(), ord.end(), 0);
    sort(ord.begin(), ord.end(), [&](int l, int r){
        int sl = q[l].first/SQ;
        int sr = q[r].first/SQ;
        if (sl != sr) return sl < sr;
        return q[l].second < q[r].second;
    });
    int l = 0, r = 0;
    insert(0);

    for (int i : ord){
        int ql, qr;
        tie(ql, qr) = q[i];
        while (r < qr) insert(++r);
        while (l > ql) insert(--l);
        while (l < ql) erase(l++);
        while (r > qr) erase(r--);
        ret[i] = ans;
    }
    return ret;
}
```

## 6.12 Distinct Range Query

```
// build - O(n (log n + log(sigma)))
// query - O(log(sigma))

int v[MAX], n, nxt[MAX];

namespace wav {
    vector<vector<int> > esq(4*(1+MAXN-MINN));

    void build(int b = 0, int e = n, int p = 1, int l =
       MINN, int r = MAXN) {
        if (l == r) return;
        int m = (l+r)/2; esq[p].push_back(0);
        for (int i = b; i < e; i++)
           esq[p].push_back(esq[p].back()+(nxt[i]<=m));
        int m2 = stable_partition(nxt+b, nxt+e, [=](int
           i){return i <= m;}) - nxt;
        build(b, m2, 2*p, l, m), build(m2, e, 2*p+1, m+1, r);
    }

    int count(int i, int j, int x, int y, int p = 1, int l =
       MINN, int r = MAXN) {
        if (y < l or r < x) return 0;
        if (x <= l and r <= y) return j-i;
        int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
        return count(ei, ej, x, y, 2*p, l, m)+count(i-ei,
           j-ej, x, y, 2*p+1, m+1, r);
    }
}

void build() {
    for (int i = 0; i < n; i++) nxt[i] = MAXN+1;
    vector<ii> t;
    for (int i = 0; i < n; i++) t.push_back({v[i], i});
    sort(t.begin(), t.end());
    for (int i = 0; i < n-1; i++) if (t[i].f == t[i+1].f)
       nxt[t[i].s] = t[i+1].s;
```

```
    wav::build();
}

int query(int a, int b) {
    return wav::count(a, b+1, b+1, MAXN+1);
}
```

## 6.13 Area da Uniao de Retangulos

```
// O(n log(n))

const int MAX = 1e5+10;
namespace seg {
    pair<int, ll> seg[4*MAX];
    ll lazy[4*MAX], *v;
    int n;

    pair<int, ll> merge(pair<int, ll> l, pair<int, ll> r){
        if (l.second == r.second) return {l.first+r.first,
           l.second};
        else if (l.second < r.second) return l;
        else return r;
    }

    pair<int, ll> build(int p=1, int l=0, int r=n-1) {
        lazy[p] = 0;
        if (l == r) return seg[p] = {1, v[l]};
        int m = (l+r)/2;
        return seg[p] = merge(build(2*p, l, m), build(2*p+1,
           m+1, r));
    }
    void build(int n2, ll* v2) {
        n = n2, v = v2;
        build();
    }
    void prop(int p, int l, int r) {
        seg[p].second += lazy[p];
        if (l != r) lazy[2*p] += lazy[p], lazy[2*p+1] +=
           lazy[p];
        lazy[p] = 0;
    }
    pair<int, ll> query(int a, int b, int p=1, int l=0, int
        r=n-1) {
        prop(p, l, r);
        if (a <= l and r <= b) return seg[p];
        if (b < l or r < a) return {0, LINF};
        int m = (l+r)/2;
        return merge(query(a, b, 2*p, l, m), query(a, b,
           2*p+1, m+1, r));
    }
    pair<int, ll> update(int a, int b, int x, int p=1, int
       l=0, int r=n-1) {
        prop(p, l, r);
        if (a <= l and r <= b) {
            lazy[p] += x;
            prop(p, l, r);
            return seg[p];
        }
        if (b < l or r < a) return seg[p];
        int m = (l+r)/2;
        return seg[p] = merge(update(a, b, x, 2*p, l, m),
                update(a, b, x, 2*p+1, m+1, r));
    }
};

ll seg_vec[MAX];

ll area_sq(vector<pair<ii, ii>> &sq){
    vector<pair<ii, ii>> up;
    for (auto it : sq){
        int x1, y1, x2, y2;
        tie(x1, y1) = it.first;
        tie(x2, y2) = it.second;
        up.push_back({{x1+1,  1}, {y1, y2}});
        up.push_back({{x2+1, -1}, {y1, y2}});
    }
    sort(up.begin(), up.end());
    memset(seg_vec, 0, sizeof seg_vec);
    ll H_MAX = MAX;
    seg::build(H_MAX-1, seg_vec);
    auto it = up.begin();
    ll ans = 0;
    while (it != up.end()){
        ll L = (*it).first.first;
```

```
        while (it != up.end() && (*it).first.first == L){
            int x, inc, y1, y2;
            tie(x, inc) = it->first;
            tie(y1, y2) = it->second;
            seg::update(y1+1, y2, inc);
            it++;
        }
        if (it == up.end()) break;
        ll R = (*it).first.first;

        ll W = R-L;
        auto jt = seg::query(0, H_MAX-1);
        ll H = H_MAX - 1;
        if (jt.second == 0) H -= jt.first;
        ans += W*H;
    }
    return ans;
}
```

## 6.14 Area Maxima de Histograma

```
// Assume que todas as barras tem largura 1,
// e altura dada no vetor v
//
// O(n)

typedef long long ll;

ll area(vector<int> v) {
    ll ret = 0;
    stack<int> s;
    // valores iniciais pra dar tudo certo
    v.insert(v.begin(), -1);
    v.insert(v.end(), -1);
    s.push(0);

    for(int i = 0; i < (int) v.size(); i++) {
        while (v[s.top()] > v[i]) {
            ll h = v[s.top()]; s.pop();
            ret = max(ret, h * (i - s.top() - 1));
        }
        s.push(i);
    }

    return ret;
}
```

## 6.15 Mo algorithm - DQUERY path on trees

```
// https://codeforces.com/blog/entry/43230
// https://www.spoj.com/problems/COT2/
//
// (s*2*n*f + q*(2*n/s)*f) optimize over s, insert/erase =
   O(f)
// for s = sqrt(n), O((n+q)*sqrt(n)*f)

vector<int> g[MAX];
namespace LCA { ... }

const int MAX = 40010;
const int SQ = 316;

int w[MAX];
int st[MAX], en[MAX], hst[2*MAX];

int v[2*MAX];

int ans, freq[MAX], freqv[MAX];

void dfs(int i, int p, int &t){
    v[t] = i;
    st[i] = t++;
    for (int j : g[i]){
        if (j == p) continue;
        dfs(j, i, t);
    }

    v[t] = i;
    en[i] = t++;
}

void update(int o){
    if (freqv[o] == 1){//insert w[o]
        ans += (freq[w[o]] == 0);
```

```
        freq[w[o]]++;
    }
    if (freqv[o] != 1){//erase w[o]
        ans -= (freq[w[o]] == 1);
        freq[w[o]]--;
    }
}

void insert(int p){
    int o = v[p];
    freqv[o]++;
    update(o);
}

void erase(int p){
    int o = v[p];
    freqv[o]--;
    update(o);
}

vector<tuple<int, int, int>> make_queries(vector<ii> &q_){
    LCA::build(0);//any LCA alg works
    vector<tuple<int, int, int>> q;
    for (auto &it : q_){
        int l, r;
        tie(l, r) = it;
        if (st[r] < st[l]) swap(l, r);
        int p = LCA::lca(l, r);
        int init = (p == l) ? st[l] : en[l];
        q.push_back({init, st[r], st[p]});
    }
    return q;
}

vector<int> MO(vector<ii> &q_){
    int t = 0;
    dfs(0, -1, t);
    auto q = make_queries(q_);
    ans = 0;
    memset(freq, 0, sizeof freq);
    int m = q.size();
    vector<int> ord(m), ret(m);
    iota(ord.begin(), ord.end(), 0);
    sort(ord.begin(), ord.end(), [&](int l, int r){
        int sl = get<0>(q[l])/SQ;
        int sr = get<0>(q[r])/SQ;
        if (sl != sr) return sl < sr;
        return get<1>(q[l]) < get<1>(q[r]);
    });

    int l = 0, r = 0;
    insert(0);

    for (int i : ord){
        int ql, qr, qp;
        tie(ql, qr, qp) = q[i];
        while (r < qr) insert(++r);
        while (l > ql) insert(--l);
        while (l < ql) erase(l++);
        while (r > qr) erase(r--);

        if (qp < l || qp > r){
            //lca out of range
            insert(qp);
            ret[i] = ans;
            erase(qp);
        }
        else ret[i] = ans;
    }
    return ret;
}
```

## 6.16 Mininum Enclosing Circle

```
// O(n) com alta probabilidade

const double EPS = 1e-12;
mt19937 rng((int)
   chrono::steady_clock::now().time_since_epoch().count());

struct pt {
    double x, y;
    pt(double x_ = 0, double y_ = 0) : x(x_), y(y_) {}
    pt operator + (const pt& p) const { return pt(x+p.x,
```

```
        y+p.y); }
    pt operator - (const pt& p) const { return pt(x-p.x,
        y-p.y); }
    pt operator * (double c) const { return pt(x*c, y*c); }
    pt operator / (double c) const { return pt(x/c, y/c); }
};

double dot(pt p, pt q) { return p.x*q.x+p.y*q.y; }
double cross(pt p, pt q) { return p.x*q.y-p.y*q.x; }
double dist(pt p, pt q) { return sqrt(dot(p-q, p-q)); }

pt center(pt p, pt q, pt r) {
    pt a = p-r, b = q-r;
    pt c = pt(dot(a, p+r)/2, dot(b, q+r)/2);
    return pt(cross(c, pt(a.y, b.y)), cross(pt(a.x, b.x),
        c)) / cross(a, b);
}

struct circle {
    pt cen;
    double r;
    circle(pt cen_, double r_) : cen(cen_), r(r_) {}
    circle(pt a, pt b, pt c) {
        cen = center(a, b, c);
        r = dist(cen, a);
    }
    bool inside(pt p) { return dist(p, cen) < r+EPS; }
};

circle minCirc(vector<pt> v) {
    shuffle(v.begin(), v.end(), rng);
    circle ret = circle(pt(0, 0), 0);
    for (int i = 0; i < v.size(); i++) if
        (!ret.inside(v[i])) {
        ret = circle(v[i], 0);
        for (int j = 0; j < i; j++) if (!ret.inside(v[j])) {
            ret = circle((v[i]+v[j])/2, dist(v[i], v[j])/2);
            for (int k = 0; k < j; k++) if
                (!ret.inside(v[k]))
                ret = circle(v[i], v[j], v[k]);
        }
    }
    return ret;
}
```

## 6.17 Min fixed range

```
// https://codeforces.com/contest/1195/problem/E
//
// O(n)
// ans[i] = min_{0 <= j < k} v[i+j]

vector<int> min_k(vector<int> &v, int k){
    int n = v.size();
    deque<int> d;
    auto put = [&](int i){
        while (!d.empty() && v[d.back()] > v[i])
            d.pop_back();
        d.push_back(i);
    };
    for (int i = 0; i < k-1; i++)
        put(i);
    vector<int> ans(n-k+1);
    for (int i = 0; i < n-k+1; i++){
        put(i+k-1);
        while (i > d.front()) d.pop_front();
        ans[i] = v[d.front()];
    }
    return ans;
}
```

## 6.18 Conectividade Dinamica

```
// Offline com Divide and Conquer e
// DSU com rollback
// O(n log^2(n))

typedef pair<int, int> T;

namespace data {
    int n, ans;
    int p[MAX], sz[MAX];
    stack<int> S;
```

```
    void build(int n2) {
        n = n2;
        for (int i = 0; i < n; i++) p[i] = i, sz[i] = 1;
        ans = n;
    }
    int find(int k) {
        while (p[k] != k) k = p[k];
        return k;
    }
    void add(T x) {
        int a = x.first, b = x.second;
        a = find(a), b = find(b);
        if (a == b) return S.push(-1);
        ans--;
        if (sz[a] > sz[b]) swap(a, b);
        S.push(a);
        sz[b] += sz[a];
        p[a] = b;
    }
    int query() {
        return ans;
    }
    void rollback() {
        int u = S.top(); S.pop();
        if (u == -1) return;
        sz[p[u]] -= sz[u];
        p[u] = u;
        ans++;
    }
};

int ponta[MAX]; // outra ponta do intervalo ou -1 se for
    query
int ans[MAX], n, q;
T qu[MAX];

void solve(int l = 0, int r = q-1) {
    if (l >= r) {
        ans[l] = data::query(); // agora a estrutura ta certa
        return;
    }
    int m = (l+r)/2, qnt = 1;
    for (int i = m+1; i <= r; i++) if (ponta[i]+1 and
        ponta[i] < l)
        data::add(qu[i]), qnt++;
    solve(l, m);
    while (--qnt) data::rollback();
    for (int i = l; i <= m; i++) if (ponta[i]+1 and ponta[i]
        > r)
        data::add(qu[i]), qnt++;
    solve(m+1, r);
    while (qnt--) data::rollback();
}
```

## 6.19 Points Inside Polygon

```
// Encontra quais pontos estao
// dentro de um poligono simples nao convexo
// o poligono tem lados paralelos aos eixos
// Pontos na borda estao dentro
// Pontos podem estar em ordem horaria ou anti-horaria
//
// O(n log(n))

#define f first
#define s second
#define pb push_back

typedef long long ll;
typedef pair<int, int> ii;

const ll N = 1e9+10;
const int MAX = 1e5+10;
int ta[MAX];

namespace seg {
    unordered_map<ll, int> seg;
    int query(int a, int b, ll p, ll l, ll r) {
        if (b < l or r < a) return 0;
        if (a <= l and r <= b) return seg[p];
        ll m = (l+r)/2;
        return query(a, b, 2*p, l, m)+query(a, b, 2*p+1,
            m+1, r);
```

```
    }
    int query(ll p) {
        return query(0, p+N, 1, 0, 2*N);
    }
    int update(ll i, int x, ll p, ll l, ll r) {
        if (i < l or r < i) return seg[p];
        if (l == r) return seg[p] += x;
        ll m = (l+r)/2;
        return seg[p] = update(i, x, 2*p, l, m)+update(i, x,
           2*p+1, m+1, r);
    }
    void update(ll a, ll b, int x) {
        if (a > b) return;
        update(a+N, x, 1, 0, 2*N);
        update(b+N+1, -x, 1, 0, 2*N);
    }
};

void pointsInsidePol(vector<ii>& pol, vector<ii>& v) {
    vector<pair<int, pair<int, ii> > > ev; // {x, {tipo, {a,
       b}}}
    // -1: poe ; id: query ; 1e9: tira
    for (int i = 0; i < v.size(); i++)
        ev.pb({v[i].f, {i, {v[i].s, v[i].s}}});
    for (int i = 0; i < pol.size(); i++) {
        ii u = pol[i], v = pol[(i+1)%pol.size()];
        if (u.s == v.s) {
            ev.pb({min(u.f, v.f), {-1, {u.s, u.s}}});
            ev.pb({max(u.f, v.f), {N, {u.s, u.s}}});
            continue;
        }
        int t = N;
        if (u.s > v.s) t = -1;
        ev.pb({u.f, {t, {min(u.s, v.s)+1, max(u.s, v.s)}}});
    }

    sort(ev.begin(), ev.end());
    for (int i = 0; i < v.size(); i++) ta[i] = 0;
    for (auto i : ev) {
        pair<int, ii> j = i.s;
        if (j.f == -1) seg::update(j.s.f, j.s.s, 1);
        else if (j.f == N) seg::update(j.s.f, j.s.s, -1);
        else if (seg::query(j.s.f)) ta[j.f] = 1; // ta dentro
    }
}
```

# 7 Strings

## 7.1 Algoritmo Z

```
// Complexidades:
// z - O(|s|)
// match - O(|s| + |p|)

vector<int> get_z(string s) {
    int n = s.size();
    vector<int> z(n, 0);

    // intervalo da ultima substring valida
    int l = 0, r = 0;
    for (int i = 1; i < n; i++) {
        // estimativa pra z[i]
        if (i <= r) z[i] = min(r - i + 1, z[i - l]);
        // calcula valor correto
        while (i + z[i] < n and s[z[i]] == s[i + z[i]])
            z[i]++;
        // atualiza [l, r]
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }

    return z;
}

// quantas vezes p aparece em s
int match(string s, string p) {
    int n = s.size(), m = p.size();
    vector<int> z = get_z(p + s);

    int ret = 0;
    for (int i = m; i < n + m; i++)
        if (z[i] >= m) ret++;
```

```
    return ret;
}
```

## 7.2 String hashing

```
// String deve ter valores [1, x]
// p deve ser o menor primo maior que x
// Para evitar colisao: testar mais de um
// mod; so comparar strings do mesmo tamanho
// ex : str_hash<31, 1e9+7> h(s);
//      ll val = h(10, 20);
//
// Complexidades:
// build - O(|s|)
// get_hash - O(1)

typedef long long ll;

template<int P, int MOD> struct str_hash {
    int n;
    string s;
    vector<ll> h, power;
    str_hash(string s_): n(s_.size()), s(s_), h(n), power(n){
        power[0] = 1;
        for (int i = 1; i < n; i++) power[i] = power[i-1]*P
            % MOD;
        h[0] = s[0];
        for (int i = 1; i < n; i++) h[i] = (h[i-1]*P + s[i])
            % MOD;
    }
    ll operator()(int i, int j){
        if (!i) return h[j];
        return (h[j] - h[i-1]*power[j-i+1] % MOD + MOD) %
            MOD;
    }
};
```

## 7.3 Automato de Sufixo

```
// Automato que aceita os sufixos de uma string
// Todas as funcoes sao lineares

namespace sam {
    int cur, sz, len[2*MAX], link[2*MAX], acc[2*MAX];
    int nxt[2*MAX][26];

    void add(int c) {
        int at = cur;
        len[sz] = len[cur]+1, cur = sz++;
        while (at != -1 and !nxt[at][c]) nxt[at][c] = cur,
            at = link[at];
        if (at == -1) { link[cur] = 0; return; }
        int q = nxt[at][c];
        if (len[q] == len[at]+1) { link[cur] = q; return; }
        int qq = sz++;
        len[qq] = len[at]+1, link[qq] = link[q];
        for (int i = 0; i < 26; i++) nxt[qq][i] = nxt[q][i];
        while (at != -1 and nxt[at][c] == q) nxt[at][c] =
            qq, at = link[at];
        link[cur] = link[q] = qq;
    }
    void build(string& s) {
        len[0] = 0, link[0] = -1, sz++;
        for (auto i : s) add(i-'a');
        int at = cur;
        while (at) acc[at] = 1, at = link[at];
    }

    // coisas que da pra fazer:
    ll distinct_substrings() {
        ll ans = 0;
        for (int i = 1; i < sz; i++) ans += len[i] -
            len[link[i]];
        return ans;
    }
    string longest_common_substring(string& S, string& T) {
        build(S);
        int at = 0, l = 0, ans = 0, pos = -1;
        for (int i = 0; i < T.size(); i++) {
            while (at and !nxt[at][T[i]-'a']) at = link[at],
                l = len[at];
            if (nxt[at][T[i]-'a']) at = nxt[at][T[i]-'a'],
                l++;
```

```
            else at = 0, l = 0;
            if (l > ans) ans = l, pos = i;
        }
        return T.substr(pos-ans+1, ans);
    }
    ll dp[2*MAX];
    ll paths(int i) {
        auto& x = dp[i];
        if (x) return x;
        x = 1;
        for (int j = 0; j < 26; j++) if (nxt[i][j]) x +=
            paths(nxt[i][j]);
        return x;
    }
    void kth_substring(int k, int at=0) { // k=1 : menor
        substring lexicog.
        for (int i = 0; i < 26; i++) if (k and nxt[at][i]) {
            if (paths(nxt[at][i]) >= k) {
                cout << char('a'+i);
                kth_substring(k-1, nxt[at][i]);
                return;
            }
            k -= paths(nxt[at][i]);
        }
    }
};
```

## 7.4 Suffix Array

```
// kasai recebe o suffix array e calcula lcp[i],
// o lcp entre s[sa[i],...,n-1] e s[sa[i+1],..,n-1]
//
// Complexidades:
// suffix_array - O(n log(n))
// kasai - O(n)

vector<int> suffix_array(string s) {
    s += "$";
    int n = s.size(), N = max(n, 260);
    vector<int> sa(n), ra(n);
    for(int i = 0; i < n; i++) sa[i] = i, ra[i] = s[i];
    for(int k = 0; k < n; k ? k *= 2 : k++) {
        vector<int> nsa(sa), nra(n), cnt(N);

        for(int i = 0; i < n; i++) nsa[i] = (nsa[i]-k+n)%n,
            cnt[ra[i]]++;
        for(int i = 1; i < N; i++) cnt[i] += cnt[i-1];
        for(int i = n-1; i+1; i--) sa[--cnt[ra[nsa[i]]]] =
            nsa[i];

        for(int i = 1, r = 0; i < n; i++) nra[sa[i]] = r +=
            ra[sa[i]] !=
             ra[sa[i-1]] or ra[(sa[i]+k)%n] !=
                ra[(sa[i-1]+k)%n];
        ra = nra;
    }
    return vector<int>(sa.begin()+1, sa.end());
}

vector<int> kasai(string s, vector<int> sa) {
    int n = s.size(), k = 0;
    vector<int> ra(n), lcp(n);
    for (int i = 0; i < n; i++) ra[sa[i]] = i;

    for (int i = 0; i < n; i++, k -= !!k) {
        if (ra[i] == n-1) { k = 0; continue; }
        int j = sa[ra[i]+1];
        while (i+k < n and j+k < n and s[i+k] == s[j+k]) k++;
        lcp[ra[i]] = k;
    }
    return lcp;
}
```

## 7.5 Suffix Array Rafael

```
// O(n log^2(n))

struct suffix_array{
    string &s;
    int n;
    vector<int> p, r, aux, lcp;
    seg_tree<int, min_el> st;
    suffix_array(string &s):
```

```
    s(s), n(s.size()), p(n), r(n), aux(n), lcp(n){
        for (int i = 0; i < n; i++){
            p[i] = i;
            r[i] = s[i];
        }
        auto rank = [&](int i){
            if (i >= n) return -i;
            return r[i];
        };
        for (int d = 1; d < n; d *= 2){
            auto t = [&](int i){
                return make_pair(rank(i), rank(i+d));
            };
            sort(p.begin(), p.end(),
                    [&](int &i, int &j){
                    return t(i) < t(j);
                    }
                );
            aux[p[0]] = 0;
            for (int i = 1; i < n; i++)
                aux[p[i]] = aux[p[i-1]] + (t(p[i]) >
                   t(p[i-1]));
            for (int j = 0; j < n; j++) r[j] = aux[j];
            if (aux[p[n-1]] == n-1) break;
        }

        int h = 0;
        for (int i = 0; i < n; i++){
            if (r[i] == n-1){
                lcp[r[i]] = 0;
                continue;
            }
            int j = p[r[i] + 1];
            while (i + h < n && j + h < n && s[i+h] ==
               s[j+h]) h++;
            lcp[r[i]] = h;
            h = max(0, h-1);
        }
        st = seg_tree<int, min_el>(&lcp);
    }
int query(int l, int r){
    return st.query(l, r);
    }
    ll distinct_substrings(){
        ll ans = p[0] + 1;
        for (int i = 1; i < n; i++)
            ans += p[i] - lcp[i-1] + 1;
        return ans;
    }
};
```

## 7.6 KMP

```
// Primeiro chama a funcao process com o padrao
// Depois chama match com (texto, padrao)
// Vai retornar o numero de ocorrencias do padrao
// p eh 1-based
//
// Complexidades:
// process - O(m)
// match - O(n + m)
// n = |texto| e m = |padrao|

int p[MAX];

void process(string& s) {
    int i = 0, j = -1;
    p[0] = -1;
    while (i < s.size()) {
        while (j >= 0 and s[i] != s[j]) j = p[j];
        i++, j++;
        p[i] = j;
    }
}

int match(string& s, string& t) {
    process(t);
    int i = 0, j = 0, ans = 0;
    while (i < s.size()) {
        while (j >= 0 and s[i] != t[j]) j = p[j];
        i++, j++;
        if (j == t.size()) j = p[j], ans++;
    }
    return ans;
```

```
}
```

# 8 Extra

## 8.1 makefile

```
CXX = g++
CXXFLAGS = -fsanitize=address -O1 -fno-omit-frame-pointer -g
   -Wall -Wshadow -std=c++14 -Wno-unused-result
   -Wno-sign-compare

CXXFLAGS = -fsanitize=address,undefined
   -fno-sanitize-recover=all -D_GLIBCXX_DEBUG -O1
   -fno-omit-frame-pointer -g -Wall -Wshadow -Wconversion
   -std=c++14 -Wno-unused-result -Wno-sign-compare
```

## 8.2 template.cpp

```
#include <bits/stdc++.h>

using namespace std;

#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'
#define f first
#define s second
#define pb push_back

typedef long long ll;
typedef pair<int, int> ii;

const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;

int main(){ _
    exit(0);
}
```

## 8.3 vimrc

```
set ts=4 si ai sw=4 number mouse=a
syntax on
```

## 8.4 stress.sh

```
make a a2 gen
for ((i = 1; ; i++)) do
    ./gen $i > in
    ./a < in > out
    ./a2 < in > out2
    if (! cmp -s out out2) then
        echo "--> entrada:"
        cat in
        echo "--> saida1:"
        cat out
        echo "--> saida2:"
        cat out2
        break;
    fi
done
```