Competitive Programming Notebook

As Meninas Superpoderosas

Contents

1 DS

1.1 Ordered Set

```cpp
// Ordered Set
//
// set roubado com mais operacoes
//
// para alterar para multiset
// trocar less para less_equal
//
// ordered_set<int> s
//
// order_of_key(k) // number of items strictly smaller than k -> int
// find_by_order(k) // k-th element in a set (counting from zero) -> iterator
//
// https://cses.fi/problemset/task/2169
//
// O(log N) para insert, erase (com iterator), order_of_key, find_by_order

using namespace __gnu_pbds;
template <typename T>
using ordered_set = tree<T,null_type,less<T>, rb_tree_tag,tree_order_statistics_node_update>;

void erase(ordered_set& a, int x){
    int r = a.order_of_key(x);
    auto it = a.find_by_order(r);
    a.erase(it);
}
```

1.2 Bigk

```cpp
struct SetSum {
    ll sum;
    multiset<ll> ms;

    SetSum() {}

    void add(ll x) {
        sum += x;
        ms.insert(x);
    }

    int rem(ll x) {
        auto it = ms.find(x);

        if (it == ms.end()) {
            return 0;
        }

        sum -= x;
        ms.erase(it);
        return 1;
    }

    ll getMin() { return *ms.begin(); }

    ll getMax() { return *ms.rbegin(); }

    ll getSum() { return sum; }

    int size() { return (int)ms.size(); }
};

struct BigK {
    int k;
    SetSum gt, mt;

    BigK(int k): k(k) {}

    void balance() {
        while (gt.size() > k) {
            ll mn = gt.getMin();
            gt.rem(mn);
            mt.add(mn);
        }

        while (gt.size() < k && mt.size() > 0) {
            ll mx = mt.getMax();
            mt.rem(mx);
            gt.add(mx);
        }
    }

    void add(ll x) {
        gt.add(x);
        balance();
    }

    void rem(ll x) {
        if (mt.rem(x) == 0) {
            gt.rem(x);
        }

        balance();
    }

    // be careful, O(abs(oldK - newk) * log)
    void setK(int _k) {
        k = _k;
        balance();
    }

    // O(log)
    void incK() { setK(k + 1); }

    // O(log)
    void decK() { setK(k - 1); }
};
```

1.3 Mex

```cpp
// Mex
//
// facilita queries de mex com update
//
// N eh o maior valor possÃŋvel do mex
// add(x) = adiciona x
// rem(x) = remove x
//
// O(log N) por insert
// O(1) por query

struct Mex {
    map<int, int> cnt;
    set<int> possible;

    Mex(int n) {
        for (int i = 0; i <= n + 1; i++) {
            possible.insert(i);
        }
    }

    void add(int x) {
        cnt[x]++;
        possible.erase(x);
    }

    void rem(int x) {
        cnt[x]--;

        if (cnt[x] == 0) {
            possible.insert(x);
        }
    }

    int query() {
        return *(possible.begin());
    }
};
```

1.4 Segtree Lazy Iterative

```
// Segtree iterativa com lazy
//
// https://codeforces.com/gym/103708/problem/C
//
// O(N * log(N)) build
// O(log(N)) update e query

const int MAX = 524288; // NEED TO BE POWER OF 2 !!!
const int LOG = 19; // LOG = ceil(log2(MAX))

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);
        }
    }

    void upd_lazy(int i, int tam) {
        if (lazy[i] && (2 * i + 1) < 2 * MAX) {
            poe(2*i, lazy[i], tam);
            poe(2*i+1, lazy[i], tam);
            lazy[i] = 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;
            upd_lazy(i, tam);
        }
    }

    void build(int n2) {
        n = n2;
        for (int i = 0; i < n; i++) seg[n+i] = 0;
        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);
    }

    int findkth(int x, int l, int r, ll k, int tam){
        int esq = x + x;
        int dir = x + x + 1;

        upd_lazy(x, tam);
        upd_lazy(esq, tam/2);
        upd_lazy(dir, tam/2);

        if(l == r){
            return l;
        } else {
            int mid = l + (r-l)/2;

            if(seg[esq] >= k){
                return findkth(esq,l,mid,k, tam/2);
            } else {
                return findkth(dir,mid+1, r, k - seg[esq], tam/2);
            }
        }
    }

    int findkth(ll k){
        // kth smallest, O(logN)
        // use position i to count how many times value ’i’ appear
        // merge must be the sum of nodes
        return findkth(1,0,n-1,k,(1 << (LOG-1)));
    }
};
```

1.5 Kruskal

```
struct Edge {
    int u, v;
    ll weight;

    Edge() {}

    Edge(int u, int v, ll weight) : u(u), v(v), weight(weight) {}

    bool operator<(Edge const& other) {
        return weight < other.weight;
    }
};

vector<Edge> kruskal(vector<Edge> edges, int n) {
    vector<Edge> result;
    ll cost = 0;

    sort(edges.begin(), edges.end());
    DSU dsu(n);

    for (auto e : edges) {
        if (!dsu.same(e.u, e.v)) {
            cost += e.weight;
            result.push_back(e);
            dsu.unite(e.u, e.v);
        }
    }

    return result;
}
```

1.6 Dsu

```
struct DSU {
    int n;
    vector<int> link, sizes;

    DSU(int n) {
        this->n = n;
        link.assign(n+1, 0);
        sizes.assign(n+1, 1);

        for (int i = 0; i <= n; i++)
            link[i] = i;
    }

    int find(int x) {
        while (x != link[x])
            x = link[x];

        return x;
    }

    bool same(int a, int b) {
        return find(a) == find(b);
    }

    void unite(int a, int b) {
        a = find(a);
        b = find(b);

        if (a == b) return;

        if (sizes[a] < sizes[b])
            swap(a, b);

        sizes[a] += sizes[b];
        link[b] = a;
    }
};
```

2 Graph

2.1 Dijkstra

```
const int INF = 1e9+17;
vector<vector<pair<int, int>>> adj; // {neighbor, weight}

void dijkstra(int s, vector<int> & d, vector<int> & p) {
    int n = adj.size();
    d.assign(n, INF);
    p.assign(n, -1);

    d[s] = 0;
    set<pair<int, int>> q;
    q.insert({0, s});
    while (!q.empty()) {
        int v = q.begin()->second;
        q.erase(q.begin());

        for (auto edge : adj[v]) {
            int to = edge.first;
            int len = edge.second;

            if (d[v] + len < d[to]) {
                q.erase({d[to], to});
                d[to] = d[v] + len;
                p[to] = v;
                q.insert({d[to], to});
            }
        }
    }
}
```

2.2 Ford Fulkerson

```
// Ford-Fulkerson
//
// max-flow / min-cut
//
// MAX nÃşs
//
// https://cses.fi/problemset/task/1694/
//
// O(m * max_flow)

using ll = long long;
const int MAX = 510;

struct Flow {
    int n;
    ll adj[MAX][MAX];
    bool used[MAX];

    Flow(int n) : n(n) {};

    void add_edge(int u, int v, ll c) {
        adj[u][v] += c;
        adj[v][u] = 0; // cuidado com isso
    }

    ll dfs(int x, int t, ll amount) {
        used[x] = true;

        if (x == t) return amount;

        for (int i = 1; i <= n; i++) {
            if (adj[x][i] > 0 && !used[i]) {
                ll sent = dfs(i, t, min(amount, adj[x][i]));

                if (sent > 0) {
                    adj[x][i] -= sent;
                    adj[i][x] += sent;

                    return sent;
                }
            }
        }

        return 0;
    }

    ll max_flow(int s, int t) { // source and sink
        ll total = 0;
        ll sent = -1;

        while (sent != 0) {
            memset(used, 0, sizeof(used));
            sent = dfs(s, t, INT_MAX);
            total += sent;
        }

        return total;
    }
};
```

2.3 2sat

```
// 2SAT
//
// verifica se existe e encontra soluÃğÃčo
// para fÃşrmulas booleanas da forma
// (a or b) and (!a or c) and (...)
//
// indexado em 0
// n(a) = 2*x e n(~a) = 2*x+1
// a = 2 ; n(a) = 4 ; n(~a) = 5 ; n(a)^1 = 5 ; n(~a)^1 = 4
//
// https://cses.fi/problemset/task/1684/
// https://codeforces.com/gym/104120/problem/E
// (add_eq, add_true, add_false e at_most_one nÃčo foram testadas)
//
// O(n + m)

struct sat {
    int n, tot;
    vector<vector<int>> adj, adjt; // grafo original, grafo transposto
    vector<int> vis, comp, ans;
    stack<int> topo; // ordem topolÃşgica

    sat() {}
    sat(int n_) : n(n_), tot(n), adj(2*n), adjt(2*n) {}

    void dfs(int x) {
        vis[x] = true;

        for (auto e : adj[x]) {
            if (!vis[e]) dfs(e);
        }

        topo.push(x);
    }

    void dfst(int x, int& id) {
        vis[x] = true;
        comp[x] = id;

        for (auto e : adjt[x]) {
            if (!vis[e]) dfst(e, id);
        }
    }

    void add_impl(int a, int b) { // a -> b = (!a or b)
        a = (a >= 0 ? 2*a : -2*a-1);
        b = (b >= 0 ? 2*b : -2*b-1);

        adj[a].push_back(b);
        adj[b^1].push_back(a^1);

        adjt[b].push_back(a);
        adjt[a^1].push_back(b^1);
    }

    void add_or(int a, int b) { // a or b
        add_impl(~a, b);
    }

    void add_nor(int a, int b) { // a nor b = !(a or b)
        add_or(~a, b), add_or(a, ~b), add_or(~a, ~b);
    }

    void add_and(int a, int b) { // a and b
        add_or(a, b), add_or(~a, b), add_or(a, ~b);
    }

    void add_nand(int a, int b) { // a nand b = !(a and b)
        add_or(~a, ~b);
    }

    void add_xor(int a, int b) { // a xor b = (a != b)
        add_or(a, b), add_or(~a, ~b);
    }

    void add_xnor(int a, int b) { // a xnor b = !(a xor b) = (a = b)
        add_xor(~a, b);
    }

    void add_true(int a) { // a = T
        add_or(a, ~a);
    }

    void add_false(int a) { // a = F
        add_and(a, ~a);
    }

    // magia - brunomaletta
    void add_true_old(int a) { // a = T (n sei se funciona)
        add_impl(~a, a);
    }

    void at_most_one(vector<int> v) { // no max um verdadeiro
        adj.resize(2*(tot+v.size()));
        for (int i = 0; i < v.size(); i++) {
            add_impl(tot+i, ~v[i]);
            if (i) {
                add_impl(tot+i, tot+i-1);
                add_impl(v[i], tot+i-1);
            }
        }
        tot += v.size();
    }

    pair<bool, vector<int>> solve() {
        ans.assign(n, -1);
        comp.assign(2*tot, -1);
        vis.assign(2*tot, 0);
        int id = 1;

        for (int i = 0; i < 2*tot; i++) if (!vis[i]) dfs(i);

        vis.assign(2*tot, 0);
        while (topo.size()) {
            auto x = topo.top();
            topo.pop();

            if (!vis[x]) {
                dfst(x, id);
                id++;
            }
        }

        for (int i = 0; i < tot; i++) {
            if (comp[2*i] == comp[2*i+1]) return {false, {}};
            ans[i] = (comp[2*i] > comp[2*i+1]);
        }

        return {true, ans};
    }
};
```

2.4 Floyd Warshall

```
const long long LLINF = 0x3f3f3f3f3f3f3f3fLL;

for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        adj[i][j] = 0;
    }
}

long long dist[MAX][MAX];
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        if (i == j)
            dist[i][j] = 0;
        else if (adj[i][j])
            dist[i][j] = adj[i][j];
        else
            dist[i][j] = LLINF;
    }
}

for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
        }
    }
}
```

2.5 Lca

```
// LCA
//
// lowest common ancestor between two nodes
//
// edit_distance(n, adj, root)
//
// https://cses.fi/problemset/task/1688
//
// O(log N)

struct LCA {
    const int MAXE = 31;
    vector<vector<int>> up;
    vector<int> dep;

    LCA(int n, vector<vector<int>>& adj, int root = 1) {
        up.assign(n+1, vector<int>(MAXE, -1));
        dep.assign(n+1, 0);

        dep[root] = 1;
        dfs(root, -1, adj);

        for (int j = 1; j < MAXE; j++) {
            for (int i = 1; i <= n; i++) {
                if (up[i][j-1] != -1)
                    up[i][j] = up[ up[i][j-1] ][j-1];
            }
        }
    }

    void dfs(int x, int p, vector<vector<int>>& adj) {
        up[x][0] = p;
        for (auto e : adj[x]) {
        if (e != p) {
            dep[e] = dep[x] + 1;
            dfs(e, x, adj);
        }
        }
    }

    int jump(int x, int k) { // jump from node x k times
        for (int i = 0; i < MAXE; i++) {
        if (k&(1 << i) && x != -1) x = up[x][i];
        }
        return x;
    }

    int lca(int a, int b) {
        if (dep[a] > dep[b]) swap(a, b);
        b = jump(b, dep[b] - dep[a]);

        if (a == b) return a;

        for (int i = MAXE-1; i >= 0; i--) {
        if (up[a][i] != up[b][i]) {
            a = up[a][i];
            b = up[b][i];
        }
        }

        return up[a][0];
    }

    int dist(int a, int b) {
        return dep[a] + dep[b] - 2 * dep[lca(a, b)];
    }
};
```

2.6 Bfs

```
vector<vector<int>> adj; // adjacency list representation
int n; // number of nodes
int s; // source vertex

queue<int> q;
vector<bool> used(n + 1);
vector<int> d(n + 1), p(n + 1);

q.push(s);
used[s] = true;
p[s] = -1;
while (!q.empty()) {
    int v = q.front();
    q.pop();
    for (int u : adj[v]) {
        if (!used[u]) {
            used[u] = true;
            q.push(u);
            d[u] = d[v] + 1;
            p[u] = v;
        }
    }
}

// restore path
if (!used[u]) {
    cout << "No path!";
} else {
    vector<int> path;

    for (int v = u; v != -1; v = p[v])
        path.push_back(v);

    reverse(path.begin(), path.end());

    cout << "Path: ";
    for (int v : path)
        cout << v << " ";
}
```

2.7 Min Cost Max Flow

```
// Min Cost Max Flow (brunomaletta)
//
// min_cost_flow(s, t, f) computa o par (fluxo, custo)
// com max(fluxo) <= f que tenha min(custo)
// min_cost_flow(s, t) -> Fluxo maximo de custo minimo de s pra t
// Se for um dag, da pra substituir o SPFA por uma DP pra nao
// pagar O(nm) no comeco
// Se nao tiver aresta com custo negativo, nao precisa do SPFA
//
// O(nm + f * m log n)

template<typename T> struct mcmf {
    struct edge {
        int to, rev, flow, cap; // para, id da reversa, fluxo, capacidade
        bool res; // se eh reversa
        T cost; // custo da unidade de fluxo
        edge() : to(0), rev(0), flow(0), cap(0), cost(0), res(false) {}
        edge(int to_, int rev_, int flow_, int cap_, T cost_, bool res_)
            : to(to_), rev(rev_), flow(flow_), cap(cap_), res(res_), cost(cost_) {}
    };

    vector<vector<edge>> g;
    vector<int> par_idx, par;
    T inf;
    vector<T> dist;

    mcmf(int n) : g(n), par_idx(n), par(n), inf(numeric_limits<T>::max()/3) {}

    void add(int u, int v, int w, T cost) { // de u pra v com cap w e custo cost
        edge a = edge(v, g[v].size(), 0, w, cost, false);
        edge b = edge(u, g[u].size(), 0, 0, -cost, true);

        g[u].push_back(a);
        g[v].push_back(b);
    }

    vector<T> spfa(int s) { // nao precisa se nao tiver custo negativo
        deque<int> q;
        vector<bool> is_inside(g.size(), 0);
        dist = vector<T>(g.size(), inf);

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

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

            for (int i = 0; i < g[v].size(); i++) {
                auto [to, rev, flow, cap, res, cost] = g[v][i];
                if (flow < cap and dist[v] + cost < dist[to]) {
                    dist[to] = dist[v] + cost;

                if (is_inside[to]) continue;
                if (!q.empty() and dist[to] > dist[q.front()]) q.push_back(to);
                else q.push_front(to);
                is_inside[to] = true;
            }
        }
    }
    return dist;
}
bool dijkstra(int s, int t, vector<T>& pot) {
    priority_queue<pair<T, int>, vector<pair<T, int>>, greater<>> q;
    dist = vector<T>(g.size(), inf);
    dist[s] = 0;
    q.emplace(0, s);
    while (q.size()) {
        auto [d, v] = q.top();
        q.pop();
        if (dist[v] < d) continue;
        for (int i = 0; i < g[v].size(); i++) {
            auto [to, rev, flow, cap, res, cost] = g[v][i];
            cost += pot[v] - pot[to];
            if (flow < cap and dist[v] + cost < dist[to]) {
                dist[to] = dist[v] + cost;
                q.emplace(dist[to], to);
                par_idx[to] = i, par[to] = v;
            }
        }
    }
    return dist[t] < inf;
}

pair<int, T> min_cost_flow(int s, int t, int flow = INF) {
    vector<T> pot(g.size(), 0);
    pot = spfa(s); // mudar algoritmo de caminho minimo aqui

    int f = 0;
    T ret = 0;
    while (f < flow and dijkstra(s, t, pot)) {
        for (int i = 0; i < g.size(); i++)
            if (dist[i] < inf) pot[i] += dist[i];

        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];
        }

        ret += pot[t] * mn_flow;

        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;
            u = par[u];
        }

        f += mn_flow;
    }

    return make_pair(f, ret);
}
```

```cpp
    // Opcional: retorna as arestas originais por onde passa flow = cap
    vector<pair<int,int>> recover() {
        vector<pair<int,int>> used;
        for (int i = 0; i < g.size(); i++) for (edge e : g[i])
            if(e.flow == e.cap && !e.res) used.push_back({i, e.to});
        return used;
    }
};
```

2.8 Has Negative Cycle

```cpp
// Edson

using edge = tuple<int, int, int>;

bool has_negative_cycle(int s, int N, const vector<edge>& edges)
{
    const int INF { 1e9+17 };

    vector<int> dist(N + 1, INF);
    dist[s] = 0;

    for (int i = 1; i <= N - 1; i++) {
        for (auto [u, v, w] : edges) {
            if (dist[u] < INF && dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
            }
        }
    }

    for (auto [u, v, w] : edges) {
        if (dist[u] < INF && dist[v] > dist[u] + w) {
            return true;
        }
    }

    return false;
}
```

2.9 Dinic

```cpp
// Dinic / Dinitz
//
// max-flow / min-cut
//
// https://cses.fi/problemset/task/1694/
//
// O(E * V^2)

using ll = long long;
const ll FLOW_INF = 1e18 + 7;

struct Edge {
    int from, to;
    ll cap, flow;
    Edge* residual; // a inversa da minha aresta

    Edge() {};

    Edge(int from, int to, ll cap) : from(from), to(to), cap(cap), flow(0) {};

    ll remaining_cap() {
        return cap - flow;
    }

    void augment(ll bottle_neck) {
        flow += bottle_neck;
        residual->flow -= bottle_neck;
    }

    bool is_residual() {
        return cap == 0;
    }
};

struct Dinic {
    int n;
    vector<vector<Edge*>> adj;
    vector<int> level, next;

    Dinic(int n): n(n) {
        adj.assign(n+1, vector<Edge*>());
        level.assign(n+1, -1);
        next.assign(n+1, 0);
    }

    void add_edge(int from, int to, ll cap) {
        auto e1 = new Edge(from, to, cap);
        auto e2 = new Edge(to, from, 0);

        e1->residual = e2;
        e2->residual = e1;

        adj[from].push_back(e1);
        adj[to].push_back(e2);
    }

    bool bfs(int s, int t) {
        fill(level.begin(), level.end(), -1);
        queue<int> q;

        q.push(s);
        level[s] = 1;

        while (q.size()) {
            int curr = q.front();
            q.pop();

            for (auto edge : adj[curr]) {
                if (edge->remaining_cap() > 0 && level[edge->to] == -1) {
                    level[edge->to] = level[curr] + 1;
                    q.push(edge->to);
                }
            }
        }

        return level[t] != -1;
    }

    ll dfs(int x, int t, ll flow) {
        if (x == t) return flow;

        for (int& cid = next[x]; cid < (int)adj[x].size(); cid++) {
            auto& edge = adj[x][cid];
            ll cap = edge->remaining_cap();

            if (cap > 0 && level[edge->to] == level[x] + 1) {
                ll sent = dfs(edge->to, t, min(flow, cap)); // bottle neck
                if (sent > 0) {
                    edge->augment(sent);
                    return sent;
                }
            }
        }

        return 0;
    }

    ll solve(int s, int t) {
        ll max_flow = 0;

        while (bfs(s, t)) {
            fill(next.begin(), next.end(), 0);

            while (ll sent = dfs(s, t, FLOW_INF)) {
                max_flow += sent;
            }
        }

        return max_flow;
    }

    // path recover
    vector<bool> vis;
    vector<int> curr;

    bool dfs2(int x, int& t) {
        vis[x] = true;
        bool arrived = false;

        if (x == t) {
            curr.push_back(x);
            return true;
        }

        for (auto e : adj[x]) {
            if (e->flow > 0 && !vis[e->to]) { // !e->is_residual() &&
                bool aux = dfs2(e->to, t);

                if (aux) {
                    arrived = true;
                    e->flow--;
                }
            }
        }

        if (arrived) curr.push_back(x);

        return arrived;
    }

    vector<vector<int>> get_paths(int s, int t) {
        vector<vector<int>> ans;

        while (true) {
            curr.clear();
            vis.assign(n+1, false);

            if (!dfs2(s, t)) break;

            reverse(curr.begin(), curr.end());
            ans.push_back(curr);
        }

        return ans;
    }
};
```

3 String

3.1 Trie Xor

```cpp
// TrieXOR
//
// adiciona, remove e verifica se existe strings binarias
// max_xor(x) = maximiza o xor de x com algum valor da trie
//
// raiz = 0
//
// https://codeforces.com/problemset/problem/706/D
//
// O(|s|) adicionar, remover e buscar

struct TrieXOR {
    int n, alph_sz, nxt;
    vector<vector<int>> trie;
    vector<int> finish, paths;

    TrieXOR() {}

    TrieXOR(int n, int alph_sz = 2) : n(n), alph_sz(alph_sz) {
        nxt = 1;
        trie.assign(n, vector<int>(alph_sz));
        finish.assign(n * alph_sz, 0);
        paths.assign(n * alph_sz, 0);
    }

    void add(int x) {
        int curr = 0;

        for (int i = 31; i >= 0; i--) {
            int b = ((x&(1 << i)) > 0);

            if (trie[curr][b] == 0)
                trie[curr][b] = nxt++;

            paths[curr]++;
            curr = trie[curr][b];
        }

        paths[curr]++;
        finish[curr]++;
    }

    void rem(int x) {
        int curr = 0;

        for (int i = 31; i >= 0; i--) {
            int b = ((x&(1 << i)) > 0);

            paths[curr]--;
            curr = trie[curr][b];
        }

        paths[curr]--;
        finish[curr]--;
    }

    int search(int x) {
        int curr = 0;

        for (int i = 31; i >= 0; i--) {
            int b = ((x&(1 << i)) > 0);

            if (trie[curr][b] == 0) return false;

            curr = trie[curr][b];
        }

        return (finish[curr] > 0);
    }

    int max_xor(int x) { // maximum xor with x and any number of trie
```

```
        int curr = 0, ans = 0;

        for (int i = 31; i >= 0; i--) {
            int b = ((x&(1 << i)) > 0);
            int want = b^1;

            if (trie[curr][want] == 0 || paths[trie[curr][want]] == 0) want ^= 1;
            if (trie[curr][want] == 0 || paths[trie[curr][want]] == 0) break;
            if (want != b) ans |= (1 << i);

            curr = trie[curr][want];
        }

        return ans;
    }
};
```

3.2 Split

```
vector<string> split(string s, char key=' ') {
    vector<string> ans;
    string aux = "";

    for (int i = 0; i < (int)s.size(); i++) {
        if (s[i] == key) {
            if (aux.size() > 0) {
                ans.push_back(aux);
                aux = "";
            }
        } else {
            aux += s[i];
        }
    }

    if ((int)aux.size() > 0) {
        ans.push_back(aux);
    }

    return ans;
}
```

3.3 Is Substring

```
// equivalente ao in do python

bool is_substring(string a, string b){ // verifica se a Ãľ substring de b
    for(int i = 0; i < b.size(); i++){
        int it = i, jt = 0; // b[it], a[jt]

        while(it < b.size() && jt < a.size()){
            if(b[it] != a[jt])
                break;

            it++;
            jt++;

            if(jt == a.size())
                return true;
        }
    }

    return false;
}
```

3.4 Hash

```
struct Hash {
    ll MOD, P;
    int n; string s;
    vector<ll> h, hi, p;
    Hash() {}
    Hash(string s, ll MOD, ll P = 31): s(s), MOD(MOD), P(P), n(s.size()), h(n), hi(n), p(n) {
        for (int i=0;i<n;i++) p[i] = (i ? P*p[i-1]:1) % MOD;
        for (int i=0;i<n;i++)
            h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
        for (int i=n-1;i>=0;i--)
            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P) % MOD;
    }
    int query(int l, int r) {
        ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
    int query_inv(int l, int r) {
        ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l+1] % MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
};

struct DoubleHash {
    const ll MOD1 = 90264469;
    const ll MOD2 = 25699183;

    Hash hash1, hash2;

    DoubleHash();

    DoubleHash(string s) : hash1(s, MOD1), hash2(s, MOD2) {}

    pair<int, int> query(int l, int r) {
        return { hash1.query(l, r), hash2.query(l, r) };
    }

    pair<int, int> query_inv(int l, int r) {
        return { hash1.query_inv(l, r), hash2.query_inv(l, r) };
    }
};

struct TripleHash {
    const ll MOD1 = 90264469;
    const ll MOD2 = 25699183;
    const ll MOD3 = 81249169;

    Hash hash1, hash2, hash3;

    TripleHash();

    TripleHash(string s) : hash1(s, MOD1), hash2(s, MOD2), hash3(s, MOD3) {}

    tuple<int, int, int> query(int l, int r) {
        return { hash1.query(l, r), hash2.query(l, r), hash3.query(l, r) };
    }

    tuple<int, int, int> query_inv(int l, int r) {
        return { hash1.query_inv(l, r), hash2.query_inv(l, r), hash3.query_inv(l, r) };
    }
};

struct HashK {
    vector<ll> primes; // more primes = more hashes
    vector<Hash> hash;

    HashK();

    HashK(string s, vector<ll> primes): primes(primes) {
        for (auto p : primes) {
            hash.push_back(Hash(s, p));
        }
    }

    vector<int> query(int l, int r) {
        vector<int> ans;

        for (auto h : hash) {
            ans.push_back(h.query(l, r));
        }

        return ans;
    }

    vector<int> query_inv(int l, int r) {
        vector<int> ans;

        for (auto h : hash) {
            ans.push_back(h.query_inv(l, r));
        }

        return ans;
    }
};
```

4 Math

4.1 Fft Quirino

```
// FFT
//
// boa em memÃşria e ok em tempo
//
// https://codeforces.com/group/YgJmumGtHD/contest/528947/problem/H (maratona mineira)

using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> &A, bool invert) {
  int N = size(A);

  for (int i = 1, j = 0; i < N; i++) {
    int bit = N >> 1;
    for (; j & bit; bit >>= 1)
      j ^= bit;
    j ^= bit;

    if (i < j)
      swap(A[i], A[j]);
  }

  for (int len = 2; len <= N; len <<= 1) {
    double ang = 2 * PI / len * (invert ? -1 : 1);
    cd wlen(cos(ang), sin(ang));
    for (int i = 0; i < N; i += len) {
      cd w(1);
      for (int j = 0; j < len/2; j++) {
        cd u = A[i+j], v = A[i+j+len/2] * w;
        A[i+j] = u + v;
        A[i+j+len/2] = u-v;
        w *= wlen;
      }
    }
  }

  if (invert) {
    for (auto &x : A)
      x /= N;
  }
}

vector<int> multiply(vector<int> const& A, vector<int> const& B) {
  vector<cd> fa(begin(A), end(A)), fb(begin(B), end(B));
  int N = 1;
  while (N < size(A) + size(B))
    N <<= 1;
  fa.resize(N);
  fb.resize(N);

  fft(fa, false);
  fft(fb, false);
  for (int i = 0; i < N; i++)
    fa[i] *= fb[i];
  fft(fa, true);

  vector<int> result(N);
  for (int i = 0; i < N; i++)
    result[i] = round(fa[i].real());
  return result;
}
```

4.2 Ceil

```
using ll = long long;

// avoid overflow
ll division_ceil(ll a, ll b) {
    return 1 + ((a - 1) / b); // if a != 0
}

int intceil(int a, int b) {
    return (a+b-1)/b;
}
```

4.3 Division Trick

```
for(int l = 1, r; l <= n; l = r + 1) {
    r = n / (n / l);
    // n / x yields the same value for l <= x <= r
}
for(int l, r = n; r > 0; r = l - 1) {
    int tmp = (n + r - 1) / r;
    l = (n + tmp - 1) / tmp;
    // (n+x-1) / x yields the same value for l <= x <= r
}
```

4.4 Fexp

```
using ll = long long;

ll fexp(ll base, ll exp, ll m) {
    ll ans = 1;
    base %= m;

    while (exp > 0) {
        if (exp % 2 == 1) {
            ans = (ans * base) % m;
        }

        base = (base * base) % m;
        exp /= 2;
    }

    return ans;
}
```

4.5 Sieve

```cpp
// nao "otimizado"

vector<bool> sieve(int lim=1e5+17) {
    vector<bool> isprime(lim+1, true);

    isprime[0] = isprime[1] = false;

    for (int i = 2; i*i < lim; i++) {
        if (isprime[i]) {
            for (int j = i+i; j < lim; j += i) {
                isprime[j] = false;
            }
        }
    }

    return isprime;
}
```

4.6 Divisors

```cpp
vector<ll> divisors(ll n) {
    vector<ll> ans;

    for (ll i = 1; i*i <= n; i++) {
        if (n%i == 0) {
            ll value = n/i;

            ans.push_back(i);
            if (value != i) {
                ans.push_back(value);
            }
        }
    }

    return ans;
}
```

4.7 Log Any Base

```cpp
int intlog(double base, double x) {
    return (int)(log(x) / log(base));
}
```

4.8 Generate Primes

```cpp
// crivo nao otimizado

vector<int> generate_primes(int lim=1e5+17) {
    vector<int> primes;
    vector<bool> isprime(lim+1, true);

    isprime[0] = isprime[1] = false;

    for (int i = 2; i*i < lim; i++) {
        if (isprime[i]) {
            primes.push_back(i);

            for (int j = i+i; j < lim; j += i) {
                isprime[j] = false;
            }
        }
    }

    return primes;
}
```

4.9 Factorization

```cpp
// nson

using ll = long long;

vector<pair<ll, int>> factorization(ll n) {
    vector<pair<ll, int>> ans;

    for (ll p = 2; p*p <= n; p++) {
        if (n%p == 0) {
            int expoente = 0;

            while (n%p == 0) {
                n /= p;
                expoente++;
            }

            ans.push_back({p, expoente});
        }
    }

    if (n > 1) {
        ans.push_back({n, 1});
    }

    return ans;
}
```

4.10 Is Prime

```cpp
bool is_prime(ll n) {
    if (n <= 1) return false;
    if (n == 2) return true;

    for (ll i = 2; i*i <= n; i++) {
        if (n % i == 0)
            return false;
    }

    return true;
}
```

5 Geometry

5.1 Convex Hull

```cpp
// Convex Hull - Monotone Chain
//
// Convex Hull is the subset of points that forms the smallest convex polygon
// which encloses all points in the set.
//
// https://cses.fi/problemset/task/2195/
// https://open.kattis.com/problems/convexhull (counterclockwise)
//
// O(n log(n))

typedef long long ftype;

struct Point {
    ftype x, y;

    Point() {};
    Point(ftype x, ftype y) : x(x), y(y) {};

    bool operator<(Point o) {
        if (x == o.x) return y < o.y;
        return x < o.x;
    }

    bool operator==(Point o) {
        return x == o.x && y == o.y;
    }
};

ftype cross(Point a, Point b, Point c) {
    // v: a -> c
    // w: a -> b

    // v: c.x - a.x, c.y - a.y
    // w: b.x - a.x, b.y - a.y

    return (c.x - a.x) * (b.y - a.y) - (c.y - a.y) * (b.x - a.x);
}

ftype dir(Point a, Point b, Point c) {
    // 0 -> colineares
    // -1 -> esquerda
    // 1 -> direita

    ftype cp = cross(a, b, c);

    if (cp == 0) return 0;
    else if (cp < 0) return -1;
    else return 1;
}

vector<Point> convex_hull(vector<Point> points) {
    sort(points.begin(), points.end());
    points.erase( unique(points.begin(), points.end()), points.end()); // somente pontos distintos
    int n = points.size();

    if (n == 1) return { points[0] };

    vector<Point> upper_hull = {points[0], points[1]};
    for (int i = 2; i < n; i++) {
        upper_hull.push_back(points[i]);

        int sz = upper_hull.size();

        while (sz >= 3 && dir(upper_hull[sz-3], upper_hull[sz-2], upper_hull[sz-1]) == -1) {
            upper_hull.pop_back();
            upper_hull.pop_back();
            upper_hull.push_back(points[i]);
            sz--;
        }
    }

    vector<Point> lower_hull = {points[n-1], points[n-2]};
    for (int i = n-3; i >= 0; i--) {
        lower_hull.push_back(points[i]);

        int sz = lower_hull.size();

        while (sz >= 3 && dir(lower_hull[sz-3], lower_hull[sz-2], lower_hull[sz-1]) == -1) {
            lower_hull.pop_back();
            lower_hull.pop_back();
            lower_hull.push_back(points[i]);
            sz--;
        }
    }

    // reverse(lower_hull.begin(), lower_hull.end()); // counterclockwise

    for (int i = (int)lower_hull.size() - 2; i > 0; i--) {
        upper_hull.push_back(lower_hull[i]);
    }

    return upper_hull;
}
```

6 DP

6.1 Edit Distance

```cpp
// Edit Distance / Levenshtein Distance
//
// numero minimo de operacoes
// para transformar
// uma string em outra
//
// tamanho da matriz da dp eh |a| x |b|
// edit_distance(a.size(), b.size(), a, b)
//
// https://cses.fi/problemset/task/1639
//
// O(n^2)

int tb[MAX][MAX];

int edit_distance(int i, int j, string &a, string &b) {
    if (i == 0) return j;
    if (j == 0) return i;

    int &ans = tb[i][j];

    if (ans != -1) return ans;

    ans = min({
        edit_distance(i-1, j, a, b) + 1,
        edit_distance(i, j-1, a, b) + 1,
        edit_distance(i-1, j-1, a, b) + (a[i-1] != b[j-1])
    });

    return ans;
}
```

6.2 Range Dp

```cpp
// Range DP 1: https://codeforces.com/problemset/problem/1132/F
//
// You may apply some operations to this string
// in one operation you can delete some contiguous substring of this string
// if all letters in the substring you delete are equal
// calculate the minimum number of operations to delete the whole string s

#include <bits/stdc++.h>

using namespace std;

const int MAX = 510;

int n, tb[MAX][MAX];
string s;

int dp(int left, int right) {
    if (left > right) return 0;

    int& mem = tb[left][right];
    if (mem != -1) return mem;

    mem = 1 + dp(left+1, right); // gastar uma operaÃğÃčo arrumando sÃş o cara atual
    for (int i = left+1; i <= right; i++) {
        if (s[left] == s[i]) {
            mem = min(mem, dp(left+1, i-1) + dp(i, right));
        }
    }

    return mem;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    cin >> n >> s;
    memset(tb, -1, sizeof(tb));
    cout << dp(0, n-1) << '\n';

    return 0;
}
```

6.3 Lis Segtree

```
int n, arr[MAX], aux[MAX]; cin >> n;
for (int i = 0; i < n; i++) {
    cin >> arr[i];
    aux[i] = arr[i];
}

sort(aux, aux+n);

Segtree st(n); // seg of maximum

int ans = 0;
for (int i = 0; i < n; i++) {
    int it = lower_bound(aux, aux+n, arr[i]) - aux;
    int lis = st.query(0, it) + 1;

    st.update(it, lis);

    ans = max(ans, lis);
}

cout << ans << '\n';
```

6.4 Digit Dp 2

```
// Digit DP 2: https://cses.fi/problemset/task/2220
//
// Number of integers between a and b
// where no two adjacents digits are the same

#include <bits/stdc++.h>

using namespace std;
using ll = long long;

const int MAX = 20; // 10^18

ll tb[MAX][MAX][2][2];

ll dp(string& number, int pos, int last_digit, bool under, bool started) {
    if (pos >= (int)number.size()) {
        return 1;
    }

    ll& mem = tb[pos][last_digit][under][started];
    if (mem != -1) return mem;
    mem = 0;

    int limit = 9;
    if (!under) limit = number[pos] - '0';

    for (int digit = 0; digit <= limit; digit++) {
        if (started && digit == last_digit) continue;

        bool is_under = under || (digit < limit);
        bool is_started = started || (digit != 0);

        mem += dp(number, pos+1, digit, is_under, is_started);
    }

    return mem;
}

ll solve(ll ubound) {
    memset(tb, -1, sizeof(tb));
    string number = to_string(ubound);
    return dp(number, 0, 10, 0, 0);
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    ll a, b; cin >> a >> b;
    cout << solve(b) - solve(a-1) << '\n';

    return 0;
}
```

6.5 Lis Binary Search

```
int lis(vector<int> arr) {
    vector<int> dp;

    for (auto e : arr) {
        int pos = lower_bound(dp.begin(), dp.end(), e) - dp.begin();

        if (pos == (int)dp.size()) {
            dp.push_back(e);
        } else {
            dp[pos] = e;
        }
    }

    return (int)dp.size();
}
```

6.6 Lcs

```
// LCS (Longest Common Subsequence)
//
// maior subsequencia comum entre duas strings
//
// tamanho da matriz da dp eh |a| x |b|
// lcs(a, b) = string da melhor resposta
// dp[a.size()][b.size()] = tamanho da melhor resposta
//
// https://atcoder.jp/contests/dp/tasks/dp_f
//
// O(n^2)

string lcs(string a, string b) {
    int n = a.size();
    int m = b.size();

    int dp[n+1][m+1];
    pair<int, int> p[n+1][m+1];

    memset(dp, 0, sizeof(dp));
    memset(p, -1, sizeof(p));

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (a[i-1] == b[j-1]) {
                dp[i][j] = dp[i-1][j-1] + 1;
                p[i][j] = {i-1, j-1};
            } else {
                if (dp[i-1][j] > dp[i][j-1]) {
                    dp[i][j] = dp[i-1][j];
                    p[i][j] = {i-1, j};
                } else {
                    dp[i][j] = dp[i][j-1];
                    p[i][j] = {i, j-1};
                }
            }
        }
    }

    // recuperar resposta

    string ans = "";
    pair<int, int> curr = {n, m};

    while (curr.first != 0 && curr.second != 0) {
        auto [i, j] = curr;

        if (a[i-1] == b[j-1]) {
            ans += a[i-1];
        }

        curr = p[i][j];
    }

    reverse(ans.begin(), ans.end());

    return ans;
}
```

6.7 Digit Dp

```
// Digit DP 1: https://atcoder.jp/contests/dp/tasks/dp_s
//
// find the number of integers between 1 and K (inclusive)
// where the sum of digits in base ten is a multiple of D

#include <bits/stdc++.h>

using namespace std;

const int MOD = 1e9+7;

string k;
int d;

int tb[10010][110][2];

int dp(int pos, int sum, bool under) {
    if (pos >= k.size()) return sum == 0;

    int& mem = tb[pos][sum][under];
    if (mem != -1) return mem;
    mem = 0;

    int limit = 9;
    if (!under) limit = k[pos] - '0';

    for (int digit = 0; digit <= limit; digit++) {
        mem += dp(pos+1, (sum + digit) % d, under | (digit < limit));
        mem %= MOD;
    }

    return mem;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    cin >> k >> d;

    memset(tb, -1, sizeof(tb));

    cout << (dp(0, 0, false) - 1 + MOD) % MOD << '\n';

    return 0;
}
```

7 General

7.1 Mix Hash

```
// magic hash function using mix

using ull = unsigned long long;
ull mix(ull o){
    o+=0x9e3779b97f4a7c15;
    o=(o^(o>>30))*0xbf58476d1ce4e5b9;
    o=(o^(o>>27))*0x94d049bb133111eb;
    return o^(o>>31);
}
ull hash(pii a) {return mix(a.first ^ mix(a.second));}
```

7.2 Xor 1 To N

```
// XOR sum from 1 to N
ll xor_1_to_n(ll n) {
    if (n % 4 == 0) {
        return n;
    } else if (n % 4 == 1) {
        return 1;
    } else if (n % 4 == 2) {
        return n + 1;
    }

    return 0;
}
```

7.3 Base Converter

```
const string digits = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";

ll tobase10(string number, int base) {
    map<char, int> val;
    for (int i = 0; i < digits.size(); i++) {
        val[digits[i]] = i;
    }

    ll ans = 0, pot = 1;

    for (int i = number.size() - 1; i >= 0; i--) {
        ans += val[number[i]] * pot;
        pot *= base;
    }
```

```cpp

    return ans;
}

string frombase10(ll number, int base) {
    if (number == 0) return "0";

    string ans = "";

    while (number > 0) {
        ans += digits[number % base];
        number /= base;
    }

    reverse(ans.begin(), ans.end());

    return ans;
}

// verifica se um nÃžmero estÃą na base especificada
bool verify_base(string num, int base) {
    map<char, int> val;
    for (int i = 0; i < digits.size(); i++) {
        val[digits[i]] = i;
    }

    for (auto digit : num) {
        if (val[digit] >= base) {
            return false;
        }
    }

    return true;
}
```

7.4 Min Priority Queue

```cpp
template<class T> using min_priority_queue = priority_queue<T, vector<T>, greater<T>>;
```

7.5 Input By File

```cpp
freopen("file.in", "r", stdin);
freopen("file.out", "w", stdout);
```

7.6 Template

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

using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);



    return 0;
}
```

7.7 First True

```cpp
// Binary Search (first_true)
//
// first_true(2, 10, [](int x) { return x * x >= 30; }); // outputs 6
//
// [l, r]
//
// if none of the values in the range work, return hi + 1
//
// f(4) = false
// f(5) = false
// f(6) = true
// f(7) = true

int first_true(int lo, int hi, function<bool(int)> f) {
    hi++;
    while (lo < hi) {
        int mid = lo + (hi - lo) / 2;

        if (f(mid)) {
            hi = mid;
        } else {
            lo = mid + 1;
        }
    }
    return lo;
}
```

7.8 Get Subsets Sum Iterative

```cpp
vector<ll> get_subset_sums(int l, int r, vector<ll>& arr) {
    vector<ll> ans;

    int len = r-l+1;
    for (int i = 0; i < (1 << len); i++) {
        ll sum = 0;

        for (int j = 0; j < len; j++) {
            if (i&(1 << j)) {
                sum += arr[l + j];
            }
        }

        ans.push_back(sum);
    }

    return ans;
}
```

7.9 Last True

```cpp
// Binary Search (last_true)

// last_true(2, 10, [](int x) { return x * x <= 30; }); // outputs 5
//
// [l, r]
//
// if none of the values in the range work, return lo - 1
//
// f(1) = true
// f(2) = true
// f(3) = true
// f(4) = true
// f(5) = true
// f(6) = false
// f(7) = false
// f(8) = false
//
// last_true(1, 8, f) = 5
// last_true(7, 8, f) = 6

int last_true(int lo, int hi, function<bool(int)> f) {
    lo--;
    while (lo < hi) {
        int mid = lo + (hi - lo + 1) / 2;

        if (f(mid)) {
            lo = mid;
        } else {
            hi = mid - 1;
        }
    }
    return lo;
}
```

7.10 Interactive

```
// you should use cout.flush() every cout
int query(int a) {
    cout << "? " << a << '\n';
    cout.flush();
    char res; cin >> res;
    return res;
}

// using endl you don't need
int query(int a) {
    cout << "? " << a << endl;
    char res; cin >> res;
    return res;
}
```

7.11 Next Permutation

```
// output: 1,2,3; 1,3,2; 2,1,3; 2,3,1; 3,1,2; 3,2,1;

vector<int> arr = {1, 2, 3};
int n = arr.size();

do {
    for (auto e : arr) {
        cout << e << ' ';
    }
    cout << '\n';
} while (next_permutation(arr.begin(), arr.end()));
```

7.12 Random

```
random_device dev;
mt19937 rng(dev());

uniform_int_distribution<mt19937::result_type> dist(1, 6); // distribution in range [1, 6]

int val = dist(rng);
```

8 Primitives