Competitive Programming Notebook

As Meninas Superpoderosas

Contents

1 Graph

1.1 Bfs

```cpp
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 << " ";
}
```

1.2 Floyd Warshall

```cpp
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]);
        }
    }
}
```

1.3 Min Cost Max Flow

```cpp
// 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);
    }

    // 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;
    }
};
```

1.4 2sat

```cpp
// 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};
    }
};
```

1.5 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)];
    }
};
```

1.6 Dinic

```
// 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;
    }
};
```

1.7 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});
            }
        }
    }
}
```

1.8 3sat

```
// We are given a CNF, e.g. phi(x) = (x_1 or ~x_2) and (x_3 or ~x_4 or ~x_5) and ... .
// SAT finds an assignment x for phi(x) = true.
// Davis-Putnum-Logemann-Loveland Algorithm (youknowwho code)
// Complexity: O(2^n) in worst case.
// This implementation is practical for n <= 1000 or more. lmao.

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

const int N = 3e5 + 9;

// positive literal  x in [0,n),
// negative literal ~x in [-n,0)
// 0 indexed
struct SAT_GOD {
  int n;
  vector<int> occ, pos, neg;
  vector<vector<int>> g, lit;
  SAT_GOD(int n) : n(n), g(2*n), occ(2*n) { }
  void add_clause(const vector<int> &c) {
    for(auto u: c) {
      g[u+n].push_back(lit.size());
      occ[u+n] += 1;
    }
    lit.push_back(c);
  }
  //(!u | v | !w) -> (u, 0, v, 1, w, 0)
  void add(int u, int af, int v = 1e9, int bf = 0, int w = 1e9, int cf = 0) {
    vector<int> a;
    if(!af) u = ~u;
    a.push_back(u);
    if(v != 1e9) {
      if(!bf) v = ~v;
      a.push_back(v);
    }
    if(w != 1e9) {
      if(!cf) w = ~w;
      a.push_back(w);
    }
    add_clause(a);
  }
  vector<bool> x;
  vector<vector<int>> decision_stack;
  vector<int> unit_stack, pure_stack;
  void push(int u) {
    x[u + n] = 1;
    decision_stack.back().push_back(u);
    for (auto i: g[u + n]) if (pos[i]++ == 0) {
        for (auto u: lit[i]) --occ[u+n];
      }
    for (auto i: g[~u + n]) {
      ++neg[i];
      if (pos[i] == 0) unit_stack.push_back(i);
    }
  }
  void pop() {
    int u = decision_stack.back().back();
    decision_stack.back().pop_back();
    x[u + n] = 0;
    for (auto i: g[u + n]) if (--pos[i] == 0) {
        for (auto u: lit[i]) ++occ[u + n];
      }
    for (auto i: g[~u+n]) --neg[i];
  }
  bool reduction() {
    while(!unit_stack.empty() || !pure_stack.empty()) {
      if(!pure_stack.empty()) {  // pure literal elimination
        int u = pure_stack.back();
        pure_stack.pop_back();
        if (occ[u + n] == 1 && occ[~u + n] == 0) push(u);
      } else {                     // unit propagation
        int i = unit_stack.back();
        unit_stack.pop_back();
        if(pos[i] > 0) continue;
        if(neg[i]     == lit[i].size()) return false;
        if(neg[i] + 1 == lit[i].size()) {
          int w = n;
          for (int u: lit[i]) if (!x[u + n] && !x[~u + n]) w = u;
          if (x[~w + n]) return false;
          push(w);
        }
      }
    }
    return true;
  }
  bool ok() {
    x.assign(2*n,0);
    pos = neg = vector<int>(lit.size());
    decision_stack.assign(1, {});
    while(1) {
      if(reduction()) {
        int s = 0;
        for(int u = 0; u < n; ++u) if(occ[s + n] + occ[~s + n] < occ[u + n] + occ[~u + n]) s = u;
        if(occ[s + n] + occ[~s + n] == 0) return true;
        decision_stack.push_back({});
        push(s);
      } else {
        int s = decision_stack.back()[0];
        while(!decision_stack.back().empty()) pop();
        decision_stack.pop_back();
        if (decision_stack.empty()) return false;
        push(~s);
      }
    }
  }
};

int32_t main() {
  int n = 9;
  SAT_GOD t(n);
  t.add(0, 0, 1, 1);
  t.add(1, 0);
  t.add(1, 0, 3, 1, 5, 1);
  cout << t.ok() << endl;
}
```

1.9 Ford Fulkerson

```
// Ford-Fulkerson
//
// max-flow / min-cut
//
// MAX nÃ§s
//
// 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;
    }
};
```

1.10 Has Negative Cycle

```
// 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 Primitives

3 Geometry

3.1 Convex Hull

```
// 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;
}
```

4 Math

4.1 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
}
```

4.2 Log Any Base

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

4.3 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);
```

```cpp

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;
}
```

4.4 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;
}
```

4.5 Sieve

```cpp
vector<int> sieve(int MAXN){
    //list of prime numbers up to MAXN
    vector<int> primes;
    bitset<(int)1e7> not_prime;
    not_prime[0] = 1;
    not_prime[1] = 1;
    for(int i = 2; i <= MAXN; i++){
        if(!not_prime[i]){
            primes.push_back(i);
            for(ll j = 1LL * i * i; j <= MAXN; j += i){
                not_prime[(int)j] = 1;
            }
        }
    }
    return primes;
}
```

4.6 Ceil

```cpp
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;
}
```

4.7 Fexp

```cpp
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;
}
```

4.8 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;
}
```

4.9 Divisors

```
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;
}
```

4.10 Number Sum Product Of Divisors

```
// CSES - Divisor Analysis
// Print the number, sum and product of the divisors.
// Since the input number may be large, it is given as a prime factorization.
//
// Input:
// The first line has an integer n: the number of parts in the prime factorization.
// After this, there are n lines that describe the factorization. Each line has two numbers x and k where x is a prime and k is its power.
//
// Output:
// Print three integers modulo 10^9+7: the number, sum and product of the divisors.
//
// Constraints:
// (1 <= n <= 1e5) ; (2 <= x <= 1e6) ; (1 <= k <= 1e9) ; each x is a distinct prime

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

const ll MOD = 1e9 + 7;

ll expo(ll base, ll pow) {
    ll ans = 1;
    while (pow) {
        if (pow & 1) ans = ans * base % MOD;
        base = base * base % MOD;
        pow >>= 1;
    }
    return ans;
}

ll p[100001], k[100001];

int main() {
    cin.tie(0)->sync_with_stdio(0);
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) cin >> p[i] >> k[i];
    ll div_cnt = 1, div_sum = 1, div_prod = 1, div_cnt2 = 1;
    for (int i = 0; i < n; i++) {
        div_cnt = div_cnt * (k[i] + 1) % MOD;
        div_sum = div_sum * (expo(p[i], k[i] + 1) - 1) % MOD *
                  expo(p[i] - 1, MOD - 2) % MOD;
        div_prod = expo(div_prod, k[i] + 1) *
                   expo(expo(p[i], (k[i] * (k[i] + 1) / 2)), div_cnt2) % MOD;
        div_cnt2 = div_cnt2 * (k[i] + 1) % (MOD - 1);
    }
    cout << div_cnt << ' ' << div_sum << ' ' << div_prod;
    return 0;
}
```

5 General

5.1 Kosaraju

```
// https://codeforces.com/blog/entry/125435
#ifdef MAXWELL_LOCAL_DEBUG
#include "debug/debug_template.cpp"
#define dbg debug
#else
#define debug(...)
#define dbg debug
#define debugArr(arr, n)
#endif

#include <bits/stdc++.h>
#define ff first
#define ss second

using namespace std;
using ll = long long;
using ld = long double;
using pii = pair<int,int>;
using vi = vector<int>;

using tii = tuple<int,int,int>;
// auto [a,b,c] = ...
// .insert({a,b,c})

const int oo = (int)1e9 + 5; //INF to INT
const ll OO = 0x3f3f3f3f3f3f3f3fLL; //INF to LL

struct Kosaraju {

    int N;
    int cntComps;

    vector<vector<int>> g;
    vector<vector<int>> gi;

    stack<int> S;
    vector<int> vis;
    vector<int> comp;

    Kosaraju(vector<vector<int>>& arr) {
        N = (int)arr.size();
        cntComps = 0;

        g.resize(N);
        gi.resize(N);
        vis.resize(N);
        comp.resize(N);

        for(int i = 0; i < (int)arr.size(); i++) {
            for(auto &v : arr[i]) {
                g[i].push_back(v);
                gi[v].push_back(i);
            }
```

```cpp
        }

        run();
    }

    void dfs(int u) {
        vis[u] = 1;
        for(auto &v : g[u]) if(!vis[v]) {
            dfs(v);
        }
        S.push(u);
    }

    void scc(int u, int c) {
        vis[u] = 1;
        comp[u] = c;
        for(auto &v : gi[u]) if(!vis[v]) {
            scc(v, c);
        }
    }

    void run() {
        vis.assign(N, 0);

        for(int i = 0; i < N; i++) if(!vis[i]) {
            dfs(i);
        }

        vis.assign(N, 0);

        while((int)S.size()) {
            int u = S.top();
            S.pop();
            if(!vis[u]) {
                scc(u, cntComps++);
            }
        }

    }

};

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

    int t = 1;

    while(t--) {
        solve();
    }

}
```

5.2 Min Priority Queue

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

5.3 Random

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

    //mt19937 rng(chrono::steady_clock::now().time_since_epoch().count()); //gerar int
    mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count()); //gerar ll

    /*usar rng() pra gerar numeros aleatÃşrios.*/
    /*usar rng() % x pra gerar numeros em [0, x-1]*/
    for(int i = 0; i < 10; i++){
        cout << rng() << endl;
    }
    vector<ll> arr = {1,2,3,4,5,6,7,8,9};
    /*dÃą pra usar no shuffle de vector tambÃľm*/
    shuffle(arr.begin(), arr.end(),rng);
    for(auto &x: arr)
        cout << x << endl;

}
```

5.4 Next Permutation

```cpp
// 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()));
```

5.5 Base Converter

```cpp
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;
    }

    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;
}
```

5.6 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;
}
```

5.7 Flags

```
// g++ -std=c++17 -Wall -Wshadow -fsanitize=address -O2 -D -o cod a.cpp
```

5.8 Get Subsets Sum Iterative

```
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;
}
```

5.9 Last True

```
// 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;
}
```

5.10 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;
}
```

5.11 Input By File

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

5.12 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));}
```

5.13 Template

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

using namespace std;

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



    return 0;
}
```

5.14 Overflow

```
// Signatures of some built-in functions to perform arithmetic operations with overflow check
// Source: https://gcc.gnu.org/onlinedocs/gcc/Integer-Overflow-Builtins.html
//
// you can also check overflow by performing the operation with double
// and checking if the result it's greater than the maximum value supported by the variable

bool __builtin_add_overflow (type1 a, type2 b, type3 *res)
bool __builtin_sadd_overflow (int a, int b, int *res)
bool __builtin_saddl_overflow (long int a, long int b, long int *res)
bool __builtin_saddll_overflow (long long int a, long long int b, long long int *res)
bool __builtin_uadd_overflow (unsigned int a, unsigned int b, unsigned int *res)
bool __builtin_uaddl_overflow (unsigned long int a, unsigned long int b, unsigned long int *res)
bool __builtin_uaddll_overflow (unsigned long long int a, unsigned long long int b, unsigned long long int *res)

bool __builtin_sub_overflow (type1 a, type2 b, type3 *res)
bool __builtin_ssub_overflow (int a, int b, int *res)
bool __builtin_ssubl_overflow (long int a, long int b, long int *res)
bool __builtin_ssubll_overflow (long long int a, long long int b, long long int *res)
bool __builtin_usub_overflow (unsigned int a, unsigned int b, unsigned int *res)
bool __builtin_usubl_overflow (unsigned long int a, unsigned long int b, unsigned long int *res)
bool __builtin_usubll_overflow (unsigned long long int a, unsigned long long int b, unsigned long long int *res)

bool __builtin_mul_overflow (type1 a, type2 b, type3 *res)
bool __builtin_smul_overflow (int a, int b, int *res)
bool __builtin_smull_overflow (long int a, long int b, long int *res)
bool __builtin_smulll_overflow (long long int a, long long int b, long long int *res)
bool __builtin_umul_overflow (unsigned int a, unsigned int b, unsigned int *res)
bool __builtin_umull_overflow (unsigned long int a, unsigned long int b, unsigned long int *res)
bool __builtin_umulll_overflow (unsigned long long int a, unsigned long long int b, unsigned long long int *res)

bool __builtin_add_overflow_p (type1 a, type2 b, type3 c)
bool __builtin_sub_overflow_p (type1 a, type2 b, type3 c)
bool __builtin_mul_overflow_p (type1 a, type2 b, type3 c)
```

5.15 First True

```
// 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;
}
```

5.16 Xor Basis

```
// XOR Basis
// You are given a set of $N$ integer values. You should find the minimum number of values that you need to add to the set such that the following will hold true:
// For every two integers $A$ and $B$ in the set, their bitwise xor $A \oplus B$ is also in the set.

vector<ll> basis;

void add(ll x) {
    for (int i = 0; i < (int)basis.size(); i++) {
        // reduce x using the current basis vectors
        x = min(x, x ^ basis[i]);
    }

    if (x != 0) { basis.push_back(x); }
}

ll res = (1LL << (int)basis.size()) - n;
```

6 String

6.1 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;
}
```

6.2 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;
    }
};
```

6.3 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;
}
```

6.4 Trie Xor

```
// 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;
    }
};
```

7 DP

7.1 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;
}
```

7.2 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;
}
```

7.3 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();
}
```

7.4 Knapsack

```
//Submeter em c++ 64bits otimiza o long long
ll knapsack(vector<ll>& weight, vector<ll>& value, int W) {
    //Usar essa knapsack se sÃş precisar do resultado final.
    //O(W) em memÃşria
    vector<vector<ll>> table(2, vector<ll>(W + 1, 0));
    int n = (int)value.size();

    for(int k = 1; k <= n; k++) {
        for(int i = 0; i <= W; i++) {
            if(i - weight[k - 1] >= 0) {
                table[k % 2][i] = max(table[ (k - 1) % 2 ][i],
                    value[k - 1] + table[(k - 1) % 2][i - weight[k - 1]]);
            } else {
                table[k % 2][i] = max(table[(k - 1) % 2][i], table[k % 2][i]);
            }
        }
    }

    return table[n % 2][W];
}

ll knapsack(vector<ll>& weight, vector<ll>& value, int W) {
    //Usar essa knapsack se, em algum momento, precisar recuperar os indices
    //O(NW) em memÃşria

    int n = (int)value.size();
    vector<vector<ll>> table(W + 1, vector<ll>(n + 1, 0));

    for(int k = 1; k <= n; k++) {
        for(int i = 0; i <= W; i++) {
            if(i - weight[k - 1] >= 0) {
                table[i][k] = max(table[i][k - 1],
                    value[k - 1] + table[i - weight[k - 1]][k - 1]);
            } else {
                table[i][k] = max(table[i][k - 1], table[i][k]);
            }
        }
    }

    /*
    int per = W;
    vector<int> idx;
    for(int k = n; k > 0; k--) {
        if(table[per][k] == table[per][k - 1]){
            continue;
        } else {
            idx.push_back(k - 1);
            per -= weight[k - 1];
        }
    }
    */

    return table[W][n];
}


const int MOD = 998244353;

struct Knapsack {

    int S; // max value
    vector<ll> dp;

    Knapsack(int S_) {
        S = S_ + 5;
        dp.assign(S, 0);
        dp[0] = 1;
    }

    void Add(int val) {
        if(val <= 0 || val >= S) return;
        for(int i = S - 1; i >= val; i--) {
            dp[i] += dp[i - val];
            dp[i] %= MOD;
        }
    }

    void Rem(int val) {
        if(val <= 0 || val >= S) return;
        for(int i = val; i < S; i++) {
            dp[i] += MOD - dp[i - val];
            dp[i] %= MOD;
        }
    }

    int Query(int val) {
        // # of ways to select a subset of numbers with sum = val
        if(val <= 0 || val >= S) return 0;
        return dp[val];
    }

};


void solve() {

    int n, w;
    cin >> n >> w;
    vector<ll> weight(n), value(n);
    for(int i = 0; i < n; i++) {
        cin >> weight[i] >> value[i];
    }
    cout << knapsack(weight, value, w) << "\n";
}
```

7.5 Edit Distance

```
// 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;
}
```

7.6 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;
}
```

7.7 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';
```

7.8 Range Dp

```
// 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;
}
```

8 DS

8.1 Range Color Update

```
// Range color update (brunomaletta)
//
// update(l, r, c) colore o range [l, r] com a cor c,
// e retorna os ranges que foram coloridos {l, r, cor}
// query(i) returna a cor da posicao i
//
// Complexidades (para q operacoes):
// update - O(log(q)) amortizado
// query - O(log(q))

template<typename T> struct color {
    set<tuple<int, int, T>> se;

    vector<tuple<int, int, T>> update(int l, int r, T val) {
        auto it = se.upper_bound({r, INF, val});
        if (it != se.begin() and get<1>(*prev(it)) > r) {
            auto [L, R, V] = *--it;
            se.erase(it);
            se.emplace(L, r, V), se.emplace(r+1, R, V);
        }
        it = se.lower_bound({l, -INF, val});
        if (it != se.begin() and get<1>(*prev(it)) >= l) {
            auto [L, R, V] = *--it;
            se.erase(it);
            se.emplace(L, l-1, V), it = se.emplace(l, R, V).first;
        }
        vector<tuple<int, int, T>> ret;
        for (; it != se.end() and get<0>(*it) <= r; it = se.erase(it))
            ret.push_back(*it);
        se.emplace(l, r, val);
        return ret;
    }
    T query(int i) {
        auto it = se.upper_bound({i, INF, T()});
        if (it == se.begin() or get<1>(*--it) < i) return -1; // nao tem
        return get<2>(*it);
    }
};
```

8.2 Trie Old

```
struct Trie {

    int nxt = 1, sz, maxLet = 26; //tamanho do alfabeto
    vector< vector<int> > trie;
    bitset<(int)1e7> finish; //modificar esse valor pra ser >= n
    //garantir que vai submeter em cpp 64

    Trie(int n){
        sz = n;
        trie.assign(sz, vector<int>(maxLet,0));
    }

    void add(string &s){
        int cur = 0;
        for(auto c: s){
            //alterar esse azinho dependendo da entrada!!
            if(trie[cur][c-'a'] == 0){
                trie[cur][c-'a'] = nxt++;
                cur = trie[cur][c-'a'];
            } else {
                cur = trie[cur][c-'a'];
            }
        }
        finish[cur] = 1;
    }

    int search(string& s){
        int cur = 0;
        for(auto c: s){
            if(trie[cur][c - 'a'] == 0){
                return 0;
            }
            cur = trie[cur][c-'a'];
        }
        return finish[cur];
    }

};
```

8.3 Sparse

```
struct Sparse {

    vector<vector<int>> arr;

    int op(int& a, int& b){ //min, max, gcd, lcm, and, or
        return min(a,b);
        //return __gcd(a,b);
        //return max(a,b);
    }

    Sparse(vector<int>& v){ //ConstrÃşi a tabela
        int n = v.size(), logn = 0;
        while((1<<logn) <= n) logn++;
        arr.assign(n, vector<int>(logn, 0));
        for(int i = 0; i < n; i++)
            arr[i][0] = v[i];
        for(int k = 1; k < logn; k++){
            for(int i = 0; i < n; i++){
                if(i + ( 1 << k) -1 >= n)
                    break;
                int p = i+( 1 << (k-1) );
                arr[i][k] = op( arr[i][ k-1 ] , arr[p][k-1]  );
            }
        }
    }

    int query(int l, int r){
        int pot = 31 - __builtin_clz(r-l+1); //r-l+1 sÃčo INTEIROS, nÃčo ll
        int k = (1 << pot) ;
        return op(  arr[l][pot] , arr[  r - (k-1)  ][pot]   );
    }

};
```

8.4 Mex

```
// 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());
    }
};
```

8.5 Bit

```
struct BIT {
    int n, LOGN = 0;
    vector<ll> bit;

    BIT(int nn){
        n = nn + 10;
        bit.resize(n + 10, 0);
        while( (1LL << LOGN) <= n ) LOGN++;
    }

    ll query(int x){
        x++;
        ll ans = 0;
        while(x > 0){
            ans += bit[x];
            x -= (x & (-x));
        }
        return ans;
    }

    void update(int x, ll val){
        x++;
        while(x < (int)bit.size()){
            bit[x] += val;
            x += (x & (-x));
        }
    }

    int findkth(int k){
        //kth smallest, O(logN)
        //use position i to count how many times value 'i' appear
        int sum = 0, pos = 0;
        for(int i = LOGN; i >= 0; i--){
            if(pos + (1LL << i) < n && sum + bit[pos + (1LL << i)] < k){
                sum += bit[pos + (1LL << i)];
                pos += (1LL << i);
            }
        }
        return pos;
    }
/*
    int findkth(int k){
        //kth smallest, O(log^2(N))
        //use position i to count how many times value ’i’ appear
        int x = 0, mx = 200;
        for(int b = n; b > 0 && mx > 0; b /= 2){
            while( x+b < n && query(x+b) < k && mx-- > 0 ){
                x += b;
            }
        }
        return x+1;
    }
*/
};
```

8.6 Maxqueue

```
struct MaxQueue {
    stack< pair<ll,ll> > in, out;

    void add(ll x){
        if(in.size())
            in.push( { x, max(x, in.top().ss)  } );
        else
            in.push( {x, x} );
    }


    ll get_max(){
        if(in.size() > 0 && out.size() > 0)
            return max(in.top().ss, out.top().ss);
        else if(in.size() > 0) return in.top().ss;
        else if(out.size() > 0) return out.top().ss;
        else return INF;
    }


    void rem(){

        if(out.size() == 0){
            while(in.size()){
                ll temp = in.top().ff, ma;
                if(out.size() == 0) ma = temp;
                else ma = max(temp, out.top().ss);
                out.push({temp, ma});
                in.pop();
            }
        }
        //removendo o topo de out
        out.pop();
    }

    ll size(){
        return in.size() + out.size();
    }

};
```

8.7 Dsu

```
// DSU
//
// https://judge.yosupo.jp/submission/126864

struct DSU {
    int n = 0, components = 0;
    vector<int> parent;
    vector<int> size;

    DSU(int nn){
        n = nn;
        components = n;
        size.assign(n + 5, 1);
        parent.assign(n + 5, 0);
        iota(parent.begin(), parent.end(), 0);
    }

    int find(int x){
        if(x == parent[x]) {
            return x;
        }
        //path compression
        return parent[x] = find(parent[x]);
    }

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

        if(a == b) {
            return;
        }

        if(size[a] < size[b]) {
            swap(a, b);
        }

        parent[b] = a;
        size[a] += size[b];
        components -= 1;
    }

    int sameSet(int a, int b) {
        a = find(a);
        b = find(b);
        return a == b;
    }
};
```

8.8 Segtree

```
struct Segtree {

    int n; //size do array que a seg vai ser criada em cima
    vector<ll> seg;

    Segtree(vector<ll>& s){
        n = (int)s.size();
        seg.resize(n+n+n+n, 0);
        seg_build(1,0,n-1,s);
    }

    ll merge(ll a, ll b){
        //return a+b;
        if(!a) a = OO;
        if(!b) b = OO;
        return min(a,b);
    }

    void seg_build(int x, int l, int r, vector<ll>& s){
        if(r < l) return;
        if(l == r){
            seg[x] = s[l];
        } else {
            int mid = l + (r-l)/2;
            seg_build(x+x, l, mid, s);
            seg_build(x+x+1, mid+1, r, s);
            seg[x] = merge(seg[x+x], seg[x+x+1]);
        }
    }

    //nÃş atual, intervalo na Ãąrvore e intervalo pedido
    ll q(int x, int l, int r, int i, int j){
        if(r < i || l > j ) return 0;
        if(l >= i && r <= j ) return seg[x];
        int mid = l + (r-l)/2;
        return merge(q(x+x,l,mid,i,j), q(x+x+1,mid+1,r,i,j));
    }

    //att posi pra val
    void att(int x, int l, int r, int posi, ll val){
        if(l == r){
            seg[x] = val;
        } else {
            int mid = l + (r-l)/2;
            if(posi <= mid)att(x+x,l,mid,posi,val);
            else att(x+x+1,mid+1,r,posi,val);
            seg[x] = merge(seg[x+x], seg[x+x+1]);
        }
    }

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

    ll query(int l, int r){
        return q(1, 0, n-1, l, r);
    }

    void update(int posi, ll val){ //alterar em posi pra val
        att(1, 0, n-1, posi, val);
    }

    int findkth(int 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);
    }

};
```

8.9 Seglazystructnode

```
struct Node {

    int l, r;

    int pref0, suf0, best0;
    int pref1, suf1, best1;

    Node(){
        pref0 = 0; suf0 = 0; best0 = 0;
        pref1 = 0; suf1 = 0; best1 = 0;
        l = -1; r = -1;
    };

    void Init(int val_, int l_, int r_) {
        best0 = !val_;
        pref0 = !val_;
        suf0 = !val_;

        best1 = val_;
        pref1 = val_;
        suf1 = val_;

        l = l_;
        r = r_;
    }


    bool AllZero() {
        return r - l + 1 == best0;
    }

    bool AllOne() {
        return r - l + 1 == best1;
    }

    void Reverse() {
        swap(pref0, pref1);
        swap(suf0, suf1);
        swap(best0, best1);
    }

};

Node Merge(Node a, Node b) {

    if(a.l == -1 || a.r == -1) {
        return b;
    }

    if(b.l == -1 || b.r == -1) {
        return a;
    }

    auto ans = Node();

    ans.l = a.l;
    ans.r = b.r;

    // --------------------------------------------- //


    if(a.AllZero()) {
        ans.pref0 = a.pref0 + b.pref0;
    } else {
        ans.pref0 = a.pref0;
    }

    if(b.AllZero()) {
        ans.suf0 = b.suf0 + a.suf0;
    } else {
        ans.suf0 = b.suf0;
    }

    ans.best0 = max({
        a.best0,
        b.best0,
        a.suf0 + b.pref0
    });

    // --------------------------------------------- //


    if(a.AllOne()) {
        ans.pref1 = a.pref1 + b.pref1;
```

```cpp
    } else {
        ans.pref1 = a.pref1;
    }

    if(b.AllOne()) {
        ans.suf1 = b.suf1 + a.suf1;
    } else {
        ans.suf1 = b.suf1;
    }

    ans.best1 = max({
        a.best1,
        b.best1,
        a.suf1 + b.pref1
    });

    // --------------------------------------------- //

    return ans;
}


struct SegLazy {

    private:

        int n;
        vector<Node> seg;
        vector<bool> lazy; // precisa reverter ou nao


        void build(ll x, int l, int r, string& s){
            if(l == r){
                int val = s[l] - '0';
                seg[x].Init(val, l, r);
            } else {
                int mid = l + (r-l)/2;
                build(x+x, l, mid, s);
                build(x+x+1, mid+1, r, s);
                seg[x] = Merge(seg[x+x], seg[x+x+1]);
            }
        }

        void upd_lazy(ll node, ll l, ll r){

            if(lazy[node]) {
                seg[node].Reverse();
            }

            ll esq = node + node, dir = esq + 1;

            if(dir < (int)seg.size() && lazy[node]){
                lazy[esq] = !lazy[esq];
                lazy[dir] = !lazy[dir];
            }

            lazy[node] = 0;
        }

        Node q(ll x, int l, int r, int i, int j){
            upd_lazy(x,l,r);

            if(r < i || l > j)
                return Node();

            if(l >= i && r <= j )
                return seg[x];

            int mid = l + (r-l)/2;
            return Merge(q(x+x,l,mid,i,j), q(x+x+1,mid+1,r,i,j));
        }

        void upd(ll x, int l, int r, int i, int j){
            upd_lazy(x,l,r);
            if(r < i || l > j) return;
            if(l >= i && r <= j){
                lazy[x] = !lazy[x];
                upd_lazy(x,l,r);
            } else {
                int mid = l + (r-l)/2;
                upd(x+x,l,mid,i,j);
                upd(x+x+1,mid+1,r,i,j);
                seg[x] = Merge(seg[x+x], seg[x+x+1]);
            }
        }


    public:

        SegLazy(string& s){
            n = (int)s.size();
            seg.assign(n+n+n+n, Node());
            lazy.assign(n+n+n+n, 0);
            build(1,0,n-1,s);
        }


        void update(int l){
            upd(1,0,n-1,l,l);
        }

        void update_range(int l, int r){
            upd(1,0,n-1,l,r);
        }

        Node query(int l){
            return q(1, 0, n-1, l, l);
        }

        Node query(int l, int r){
            return q(1, 0, n-1, l, r);
        }

};

void solve() {

    int n, q;
    string s;

    cin >> n >> q >> s;

    SegLazy seg(s);

    while(q--) {
        int c, l, r;
        cin >> c >> l >> r;

        if(c == 1) {
            // inverte l...r
            seg.update_range(l - 1, r - 1);
        } else {
            // query l...r
            auto node = seg.query(l - 1, r - 1);
            cout << node.best1 << "\n";
        }

    }

}
```

8.10 Mergesorttree

```cpp
//const int MAXN = 3e5 + 10;
```

```cpp
//vector<int> seg[ 4 * MAXN + 10];

struct MergeSortTree {

    int n; //size do array que a seg vai ser criada em cima
    vector< vector<int> > seg;
    //vector< vector<ll> > ps; //prefix sum

    MergeSortTree(vector<int>& s){
        //se o input for grande (ou o tempo mt puxado), coloca a seg com size
        //maximo de forma global
        n = (int)s.size();
        seg.resize(4 * n + 10);
        //ps.resize(4 * n + 10);
        seg_build(1,0,n-1,s);
    }

    vector<int> merge(vi& a, vi& b){
        int i = 0, j = 0, p = 0;
        vi ans(a.size() + b.size());
        while(i < (int)a.size() && j < (int)b.size()){
            if(a[i] < b[j]){
                ans[p++] = a[i++];
            } else {
                ans[p++] = b[j++];
            }
        }
        while(i < (int)a.size()){
            ans[p++] = a[i++];
        }
        while(j < (int)b.size()){
            ans[p++] = b[j++];
        }
        return ans;
    }

    vector<ll> calc(vi& s) {
        ll sum = 0;
        vector<ll> tmp;
        for(auto &x : s) {
            sum += x;
            tmp.push_back(sum);
        }
        return tmp;
    }

    void seg_build(int x, int l, int r, vector<int>& s){
        if(r < l) return;
        if(l == r){
            seg[x].push_back(s[l]);
            //ps[x] = {s[l]};
        } else {
            int mid = l + (r-l)/2;
            seg_build(x+x, l, mid, s);
            seg_build(x+x+1, mid+1, r, s);
            seg[x] = merge(seg[x+x], seg[x+x+1]);
            //ps[x] = calc(seg[x]);
        }
    }

    //nÃş atual, intervalo na Ãąrvore e intervalo pedido
    // retorna a quantidade de numeros <= val em [l, r]

    ll q(int x, int l, int r, int i, int j, int val){
        if(r < i || l > j ) return 0;
        if(l >= i && r <= j ){
            return (lower_bound(seg[x].begin(), seg[x].end(), val)  - seg[x].begin());
        }
        int mid = l + (r-l)/2;
        return q(x+x,l,mid,i,j, val) + q(x+x+1,mid+1,r,i,j, val);
    }


    // retorna a soma dos numeros <= val em [l, r]
    // nÃş atual, intervalo na Ãąrvore e intervalo pedido
    /*
    ll q(int x, int l, int r, int i, int j, ll val){
        if(r < i || l > j ) return 0;
        if(l >= i && r <= j ){
            auto it = upper_bound(seg[x].begin(), seg[x].end(), val) - seg[x].begin();

            if(val > seg[x].back()) {
                return ps[x].back();
            }

            if(val < seg[x][0]) {
                return 0;
            }

            return ps[x][it - 1];

        }

        int mid = l + (r-l)/2;
        return q(x+x,l,mid,i,j, val) + q(x+x+1,mid+1,r,i,j, val);
    }
    */

    ll query(int l, int r, ll val){
        return q(1, 0, n-1, l, r, val);
    }

};
```

8.11 Seghash

```cpp
template<typename T> //use as SegtreeHash<int> h or SegtreeHash<char>
struct SegtreeHash {

    int n; //size do array que a seg vai ser criada em cima

    // P = 31, 53, 59, 73 .... (prime > number of different characters)
    // M = 578398229, 895201859, 1e9 + 7, 1e9 + 9 (big prime)
    int p, m;

    vector<ll> seg, pot;

    ll minValue = 0; // menor valor possÃŋvel que pode estar na estrutura
                     // isso ÃĽ pra evitar que a hash de ’0’ seja igual a de ’0000...’

    SegtreeHash(vector<T>& s, ll P = 31, ll MOD = (ll)1e9 + 7){
        n = (int)s.size();
        p = P; m = MOD;
        seg.resize(4 * n, -1);
        pot.resize(4 * n);
        pot[0] = 1;
        for(int i = 1; i < (int)pot.size(); i++) {
            pot[i] = (pot[i - 1] * P) % MOD;
        }
        seg_build(1, 0, n - 1, s);
    }

    ll merge(ll a, ll b, int tam){
        if(a == -1) return b;
        if(b == -1) return a;
        return (a + b * pot[tam]) % m;
    }

    void seg_build(int x, int l, int r, vector<T>& s) {
        if(r < l) return;
        if(l == r){
            seg[x] = (int)s[l] - minValue + 1;
        } else {
            int mid = l + (r-l)/2;
            seg_build(x+x, l, mid, s);
            seg_build(x+x+1, mid+1, r, s);
            seg[x] = merge(seg[x+x], seg[x+x+1], mid - l + 1);
        }
    }

    //nÃş atual, intervalo na Ãąrvore e intervalo pedido
    ll q(int x, int l, int r, int i, int j){
        if(r < i || l > j ) return -1;
        if(l >= i && r <= j ) return seg[x];
        int mid = l + (r-l)/2;
        return merge(q(x+x,l,mid,i,j), q(x+x+1,mid+1,r,i,j), mid - max(i, l) + 1);
    }

    //att posi pra val
    void att(int x, int l, int r, int posi, T val){
        if(l == r){
            seg[x] = (int)val - minValue + 1;
        } else {
            int mid = l + (r-l)/2;
            if(posi <= mid)att(x+x,l,mid,posi,val);
            else att(x+x+1,mid+1,r,posi,val);
            seg[x] = merge(seg[x+x], seg[x+x+1], mid - l + 1);
        }
    }

    ll query(int l, int r){
        return q(1, 0, n-1, l, r);
    }

    void update(int posi, T val){ //alterar em posi pra val
        att(1, 0, n-1, posi, val);
    }

};
```

8.12 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)));
    }
};
```

8.13 Seglazy

```
struct SegLazy {

    int n;
    vector<ll> seg;
    vector<ll> lazy;

    SegLazy(vector<ll>& arr){
        n = (int)arr.size();
        seg.assign(n+n+n+n, 0);
        lazy.assign(n+n+n+n, 0);
        build(1,0,n-1,arr);
    }

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

    void build(ll x, int l, int r, vector<ll>& arr){
        if(l == r){
            seg[x] = 1LL * arr[l];
        } else {
            int mid = l + (r-l)/2;
            build(x+x, l, mid, arr);
            build(x+x+1, mid+1, r, arr);
            seg[x] = merge(seg[x+x], seg[x+x+1]);
        }
    }

    void upd_lazy(ll node, ll l, ll r){
        seg[node] += (ll)(r-l+1) * lazy[node];
        ll esq = node + node, dir = esq + 1;

        if(dir < (int)seg.size()){
            lazy[esq] += lazy[node];
            lazy[dir] += lazy[node];
        }

        lazy[node] = 0;
    }

    ll q(ll x, int l, int r, int i, int j){
        upd_lazy(x,l,r);

        if(r < i || l > j)
            return 0;

        if(l >= i && r <= j )
            return seg[x];

        int mid = l + (r-l)/2;
        return merge(q(x+x,l,mid,i,j), q(x+x+1,mid+1,r,i,j));
    }

    ll query(int l, int r){ //valor em uma posi especÃŋfica -> query de [l,l];
        return q(1, 0, n-1, l, r);
    }

    void upd(ll x, int l, int r, int i, int j, ll u){
        upd_lazy(x,l,r);
        if(r < i || l > j) return;
        if(l >= i && r <= j){
            lazy[x] += u;
            upd_lazy(x,l,r);
        } else {
            int mid = l + (r-l)/2;
            upd(x+x,l,mid,i,j,u);
            upd(x+x+1,mid+1,r,i,j,u);
            seg[x] = merge(seg[x+x], seg[x+x+1]);
        }
    }

    void upd_range(int l, int r, ll u){ //intervalo e valor
        upd(1,0,n-1,l,r,u);
    }

};
```

8.14 Bit2d

```
struct BIT2D {

    int n, m;
    vector<vector<int>> bit;

    BIT2D(int nn, int mm) {
        //use as 0-indexed, but inside here I will use 1-indexed positions
        n = nn + 2;
        m = mm + 2;
        bit.assign(n, vector<int>(m));
    }

    void update(int x, int y, int p) {
        x++; y++;
        assert(x > 0 && y > 0 && x <= n && y <= m);
        for(; x < n; x += (x&(-x)))
            for(int j = y; j < m; j += (j&(-j)))
                bit[x][j] += p;
    }

    int sum(int x, int y) {
        int ans = 0;
        for(; x > 0; x -= (x & (-x)))
            for(int j = y; j > 0; j -= (j&(-j)))
                ans += bit[x][j];
        return ans;
    }

    int query(int x, int y, int p, int q) {
        //x...p on line, y...q on column
        //sum from [x][y] to [p][q];
        x++; y++; p++; q++;
        assert(x > 0 && y > 0 && x <= n && y <= m);
        assert(p > 0 && q > 0 && p <= n && q <= m);
        return sum(p, q) - sum(x - 1, q) - sum(p, y - 1) + sum(x - 1, y - 1);
    }


};
```

8.15 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);
}
```

8.16 Cht

```cpp
// CHT (tiagodfs)

const ll is_query = -LLINF;
struct Line{
    ll m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const{
        if(rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if(!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct Cht : public multiset<Line>{ // maintain max m*x+b
    bool bad(iterator y){
        auto z = next(y);
        if(y == begin()){
            if(z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if(z == end()) return y->m == x->m && y->b <= x->b;
        return (ld)(x->b - y->b)*(z->m - y->m) >= (ld)(y->b - z->b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b){ // min -> insert (-m,-b) -> -eval()
        auto y = insert({ m, b });
        y->succ = [=]{ return next(y) == end() ? 0 : &*next(y); };
        if(bad(y)){ erase(y); return; }
        while(next(y) != end() && bad(next(y))) erase(next(y));
        while(y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x){
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

8.17 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); }
};
```

8.18 Querytree

```
struct QueryTree {
    int n, t = 0, l = 3, build = 0, euler = 0;
    vector<ll> dist;
    vector<int> in, out, d;
    vector<vector<int>> sobe;
    vector<vector<pair<int,ll>>> arr;
    vector<vector<ll>> table_max; //max edge
    vector<vector<ll>> table_min; //min edge

    QueryTree(int nn) {
        n = nn + 5;
        arr.resize(n);
        in.resize(n);
        out.resize(n);
        d.resize(n);
        dist.resize(n);
        while( (1 << l) < n ) l++;
        sobe.assign(n + 5, vector<int>(++l));
        table_max.assign(n + 5, vector<ll>(l));
        table_min.assign(n + 5, vector<ll>(l));
    }

    void add_edge(int u, int v, ll w){ // bidirectional edge with weight w
        arr[u].push_back({v, w});
        arr[v].push_back({u, w});
    }

    //assert the root of tree is node 1 or change the 'last' in the next function
    void Euler_Tour(int u, int last = 1, ll we = 0, int depth = 0, ll sum = 0){ //euler tour
        euler = 1; //remember to use this function before the queries
        in[u] = t++;
        d[u] = depth;
        dist[u] = sum; //sum = sum of the values in edges from root to node u
        sobe[u][0] = last; //parent of u. parent of 1 is 1
        table_max[u][0] = we;
        table_min[u][0] = we;
        for(auto v: arr[u]) if(v.ff != last){
            Euler_Tour(v.ff, u, v.ss, depth + 1, sum + v.ss);
        }
        out[u] = t++;
    }

    void build_table(){ //binary lifting
        assert(euler);
        build = 1; //remeber use this function before queries
        for(int k = 1; k < l; k++){
            for(int i = 1; i <= n; i++){
                sobe[i][k] = sobe[sobe[i][k-1]][k-1];
                table_max[i][k] = max(table_max[i][k - 1], table_max[sobe[i][k-1]][k-1]);
                table_min[i][k] = min(table_min[i][k - 1], table_min[sobe[i][k-1]][k-1]);
            }
        }
    }

    int is_ancestor(int u, int v){ // return 1 if u is ancestor of v
        assert(euler);
        return in[u] <= in[v] && out[u] >= out[v];
    }

    int lca(int u, int v){ //return lca of u and v
        assert(build && euler);
        if(is_ancestor(u,v)) return u;
        if(is_ancestor(v,u)) return v;
        int lca = u;
        for(int k = l - 1; k >= 0; k--){
            int tmp = sobe[lca][k];
            if(!is_ancestor(tmp, v)){
                lca = tmp;
            }
        }
        return sobe[lca][0];
    }

    int lca(int u, int v, int root) { //return lca of u and v when tree is rooted at 'root'
        return lca(u, v) ^ lca(v, root) ^ lca(root, u); //magic
    }

    int up_k(int u, int qt){ //return node k levels higher starting from u
        assert(build && euler);
        for(int b = 0; b < l; b++){
            if(qt%2) u = sobe[u][b];
            qt >>= 1;
        }
        return u;
    }

    ll goUpMax(int u, int to){ //return the max weigth of a edge going from u to 'to'
        assert(build);
        if(u == to) return 0;
        ll mx = table_max[u][0];
        for(int k = l - 1; k >= 0; k--){
            int tmp = sobe[u][k];
            if( !is_ancestor(tmp, to) ){
                mx = max(mx, table_max[u][k]);
                u = tmp;
            }
        }
        return max(mx, table_max[u][0]);
    }

    ll max_edge(int u, int v){ //return the max weight of a edge in the simple path from u to v
        assert(build);
        int ancestor = lca(u, v);
        ll a = goUpMax(u, ancestor), b = goUpMax(v, ancestor);
        if(ancestor == u) return b;
        else if(ancestor == v) return a;
        return max(a,b);
    }

    ll goUpMin(int u, int to){ //return the min weight of a edge going from u to 'to'
        assert(build);
```

```cpp
        if(u == to) return oo;
        ll mx = table_min[u][0];
        for(int k = l - 1; k >= 0; k--){
            int tmp = sobe[u][k];
            if( !is_ancestor(tmp, to) ){
                mx = min(mx, table_min[u][k]);
                u = tmp;
            }
        }
        return min(mx, table_min[u][0]);
    }

    ll min_edge(int u, int v){ //return the min weight of a edge in the simple path from u to v
        assert(build);
        int ancestor = lca(u, v);
        ll a = goUpMin(u, ancestor), b = goUpMin(v, ancestor);
        if(ancestor == u) return b;
        else if(ancestor == v) return a;
        return min(a,b);
    }

    ll query_dist(int u, int v){ //distance of nodes u and v
        int x = lca(u, v);
        return dist[u] - dist[x] + dist[v] - dist[x];
    }

    int kth_between(int u, int v, int k){ //kth node in the simple path from u to v; if k = 1, ans = u
        k--;
        int x = lca(u, v);
        if( k > d[u] - d[x] ){
            k -= (d[u] - d[x]);
            return up_k(v, d[v]-d[x]-k);
        }
        return up_k(u, k);
    }

};

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

    int t = 1, n, u, v, w, k;
    string s;
    cin >> t;
    while(t--){
        cin >> n;
        QueryTree arr(n);
        for(int i = 1; i < n; i++){
            cin >> u >> v >> w;
            arr.add_edge(u,v,w);
        }
        arr.Euler_Tour(1);
        arr.build_table();
        while(cin >> s, s != "DONE"){
            cin >> u >> v;
            if(s == "DIST") {
                cout << arr.query_dist(u, v) << "\n";
            } else {
                cin >> k;
                cout << arr.kth_between(u,v,k) << "\n";
            }
        }
        cout << "\n";
    }

}
```

8.19 Trie

```cpp
struct Trie {

    struct Node {
        map<char, Node> adj; // dÃą pra trocar por vector(26)
        ll finishHere;

        Node() {
            finishHere = 0;
        }

        bool find(char c) {
            return adj.find(c) != adj.end();
        }

    };

    Node mainNode;

    Trie(){
        mainNode = Node();
    }

    void add(string &s) {
        Node *curNode = &mainNode;

        for(auto &c : s) {

            if(!curNode->find(c)) {
                curNode->adj[c] = Node();
            }

            curNode = &curNode->adj[c];
        }

        curNode->finishHere += 1;
    }

    void dfs(Node& node) {
        for(auto &v : node.adj) {
            dfs(v.ss);
            // faz alguma coisa
        }
    }

    void dfs() {
        return dfs(mainNode);
    }

    bool search(string &s) {
        Node* curNode = &mainNode;

        for(auto &c : s) {
            if(!curNode->find(c))
                return false;

            curNode = &curNode->adj[c];
        }

        return curNode->finishHere > 0;
    }

    void debugRec(Node node, int depth) {
        for(auto &x : node.adj) {
            cout << string(3 * depth, ' ') << x.ff << " " << x.ss.finishHere << "\n";
            debugRec(x.ss, depth + 1);
        }
    }

    void debug() {
        debugRec(mainNode, 0);
    }

};
```

8.20 Triexor

```
struct Trie {

    int nxt = 1, sz, maxLet = 2;
    vector< vector<int> > trie;
    vector<int> finish, paths;

    Trie(int n){
        sz = n;
        trie.assign(sz + 10, vector<int>(maxLet,0));
        finish.resize(sz + 10);
        paths.resize(sz+10);
    }

    void add(int x){
        int cur = 0;
        for(int i = 31; i >= 0; i--){
            int b = ( (x & (1 << i)) > 0);
            if(trie[cur][b] == 0)
                trie[cur][b] = nxt++;
            cur = trie[cur][b];
            paths[cur]++;
        }
        paths[cur]++;
    }

    void rem(int x){
       int cur = 0;
        for(int i = 31; i >= 0; i--){
            int b = ( (x & (1 << i)) > 0);
            cur = trie[cur][b];
            paths[cur]--;
        }
        finish[cur]--;
        paths[cur]--;
    }

    int query(int x){ //return the max xor with x
        int ans = 0, cur = 0;

        for(int i = 31; i >= 0; i--){
            int b = ( (x & (1 << i)) > 0);
            int bz = trie[cur][0];
            int bo = trie[cur][1];

            if(bz > 0 && bo > 0 && paths[bz] > 0 && paths[bo] > 0){
                //cout << "Optimal" << endl;
                cur = trie[cur][b ^ 1];
                ans += (1 << i);
            } else if(bz > 0 && paths[bz] > 0){
                //cout << "Zero" << endl;
                cur = trie[cur][0];
                if(b) ans += (1 << i);
            } else if(bo > 0 && paths[bo] > 0){
                //cout << "One" << endl;
                cur = trie[cur][1];
                if(!b) ans += (1 << i);
            } else {
                break;
            }
        }

        return ans;
    }

};
```

8.21 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;
}
```

8.22 Psum2d

```
struct PSum {

    vector<vi> arr;
    int n, m, initialized = 0;

    PSum(int _n, int _m) {
        n = _n;
        m = _m;
        arr.resize(n + 2);
        arr.assign(n + 2, vector<int>(m + 2, 0));
    }

    void add(int a, int b, int c) {
        //a and b are 0-indexed
        arr[a + 1][b + 1] += c;
    }

    void init() {
        for(int i = 1; i <= n; i++) {
            for(int j = 1; j <= m; j++) {
                arr[i][j] += arr[i][j - 1];
                arr[i][j] += arr[i - 1][j];
                arr[i][j] -= arr[i - 1][j - 1];
            }
        }
        initialized = 1;
    }

    int query(int a, int b, int c, int d) {
        // sum of a...c and b...d
        // a, b, c and d are 0-indexed
        assert(initialized);
        return arr[c + 1][d + 1] - arr[a][d + 1] - arr[c + 1][b] + arr[a][b];
    }

};
```