Notebook - Maratona de Programação

Lenhadoras de Segtree

Contents

1 DP

1.1 Substr Palindrome

```
// êvoc deve informar se a substring de S formada pelos elementos entre os indices i e j
// é um palindromo ou ãno.

char s[MAX];
int calculado[MAX][MAX]; // iniciado com false, ou 0
int tabela[MAX][MAX];

int is_palin(int i, int j){
  if(calculado[i][j]){
    return tabela[i][j];
  }
  if(i == j) return true;
  if(i + 1 == j) return s[i] == s[j];

  int ans = false;
  if(s[i] == s[j]){
    if(is_palin(i+1, j-1)){
      ans = true;
    }
  }
  calculado[i][j] = true;
  tabela[i][j] = ans;
  return ans;
}
```

1.2 Coins

```
int tb[1005];
int n;
vector<int> moedas;

int dp(int i){
  if(i >= n)
    return 0;
  if(tb[i] != -1)
    return tb[i];

  tb[i] = max(dp(i+1), dp(i+2) + moedas[i]);
  return tb[i];
}

int main(){
  memset(tb,-1,sizeof(tb));
}
```

1.3 Minimum Coin Change

```
int n;
vector<int> valores;

int tabela[1005];

int dp(int k){
  if(k == 0){
    return 0;
  }
  if(tabela[k] != -1)
    return tabela[k];
  int melhor = 1e9;
  for(int i = 0; i < n; i++){
    if(valores[i] <= k)
      melhor = min(melhor,1 + dp(k - valores[i]));
  }
  return tabela[k] = melhor;
}
```

1.4 Edit Distance

```
// Description:
// Minimum number of operations required to transform a string into another
// Operations allowed: add character, remove character, replace character

// Parameters:
// str1 - string to be transformed into str2
// str2 - string that str1 will be transformed into
// m - size of str1
// n - size of str2

// Problem:
// https://cses.fi/problemset/task/1639

// Complexity:
// O(m x n)

// How to use:
// memset(dp, -1, sizeof(dp));
// string a, b;
// edit_distance(a, b, (int)a.size(), (int)b.size());

// Notes:
// Size of dp matriz is m x n

int dp[MAX][MAX];

int edit_distance(string &str1, string &str2, int m, int n) {
    if (m == 0) return n;
    if (n == 0) return m;

    if (dp[m][n] != -1) return dp[m][n];

    if (str1[m - 1] == str2[n - 1]) return dp[m][n] = edit_distance(str1, str2, m - 1, n - 1);
    return dp[m][n] = 1 + min({edit_distance(str1, str2, m, n - 1), edit_distance(str1, str2, m - 1, n), edit_distance(str1, str2, m - 1, n - 1)});
}
```

1.5 Digits

```
// achar a quantidade de numeros menores que R que possuem no maximo 3 digitos nao nulos
// a ideia eh utilizar da ordem lexicografica para checar isso pois se temos por exemplo
// o numero 8500, a gente sabe que se pegarmos o numero 7... qualquer digito depois do 7
// sera necessariamente menor q 8500

string r;
int tab[20][2][5];

// i - digito de R
// menor - ja pegou um numero menor que um digito de R
// qt - quantidade de digitos nao nulos
int dp(int i, bool menor, int qt){
    if(qt > 3) return 0;
    if(i >= r.size()) return 1;
    if(tab[i][menor][qt] != -1) return tab[i][menor][qt];

    int dr = r[i]-’0’;
    int res = 0;

    for(int d = 0; d <= 9; d++) {
        int dnn = qt + (d > 0);
        if(menor == true) {
            res += dp(i+1, true, dnn);
        }
        else if(d < dr) {
            res += dp(i+1, true, dnn);
        }
        else if(d == dr) {
            res += dp(i+1, false, dnn);
        }
    }

    return tab[i][menor][qt] = res;
}
```

1.6 Knapsack With Index

```
void knapsack(int W, int wt[], int val[], int n) {
    int i, w;
    int K[n + 1][W + 1];

    for (i = 0; i <= n; i++) {
        for (w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
                K[i][w] = 0;
            else if (wt[i - 1] <= w)
                K[i][w] = max(val[i - 1] +
                    K[i - 1][w - wt[i - 1]], K[i - 1][w]);
            else
                K[i][w] = K[i - 1][w];
        }
    }

    int res = K[n][W];
    cout<< res << endl;

    w = W;
    for (i = n; i > 0 && res > 0; i--) {
        if (res == K[i - 1][w])
            continue;
        else {
            cout<<" "<<wt[i - 1] ;
            res = res - val[i - 1];
            w = w - wt[i - 1];
        }
    }
}

int main()
{
    int val[] = { 60, 100, 120 };
    int wt[] = { 10, 20, 30 };
    int W = 50;
    int n = sizeof(val) / sizeof(val[0]);

    knapsack(W, wt, val, n);

    return 0;
}
```

1.7 Kadane

```
// achar uma subsequencia continua no array que a soma seja a maior possivel
// nesse caso vc precisa multiplicar exatamente 1 elemento da subsequencia
// e achar a maior soma com isso

int n, x, arr[MAX], tab[MAX][2]; // tab[maior resposta no intervalo][foi multiplicado ou ãno]

int dp(int i, bool mult) {
    if (i == n-1) {
        if (!mult) return arr[n-1]*x;
        return arr[n-1];
    }
    if (tab[i][mult] != -1) return tab[i][mult];

    int res;

    if (mult) {
        res = max(arr[i], arr[i] + dp(i+1, 1));
    }
    else {
        res = max({
            arr[i]*x,
            arr[i]*x + dp(i+1, 1),
            arr[i] + dp(i+1, 0)
        });
    }

    return tab[i][mult] = res;
}

int main() {

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

    int ans = -oo;
    for (int i = 0; i < n; i++) {
        ans = max(ans, dp(i, 0));
    }

    return 0;
}



int ans = a[0], ans_l = 0, ans_r = 0;
int sum = 0, minus_pos = -1;

for (int r = 0; r < n; ++r) {
    sum += a[r];
    if (sum > ans) {
        ans = sum;
        ans_l = minus_pos + 1;
        ans_r = r;
    }
    if (sum < 0) {
        sum = 0;
        minus_pos = r;
    }
}
```

1.8 Knapsack

```
int val[MAXN], peso[MAXN], dp[MAXN][MAXS];

int knapsack(int n, int m){ // n Objetos | Peso max
    for(int i=0;i<=n;i++){
       for(int j=0;j<=m;j++){
            if(i==0 or j==0)
                dp[i][j] = 0;
            else if(peso[i-1]<=j)
                dp[i][j] = max(val[i-1]+dp[i-1][j-peso[i-1]], dp[i-1][j]);
            else
                dp[i][j] = dp[i-1][j];
       }
    }
    return dp[n][m];
}
```

2 Graphs

2.1 Kruskall

```
vector<int> parent, rank;

void make_set(int v) {
    parent[v] = v;
    rank[v] = 0;
}

int find_set(int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}

void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if (rank[a] < rank[b])
            swap(a, b);
        parent[b] = a;
        if (rank[a] == rank[b])
            rank[a]++;
    }
}

struct Edge {
    int u, v, weight;
    bool operator<(Edge const& other) {
        return weight < other.weight;
    }
};

int n;
vector<Edge> edges;

int cost = 0;
vector<Edge> result;
parent.resize(n);
rank.resize(n);
for (int i = 0; i < n; i++)
    make_set(i);

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

for (Edge e : edges) {
    if (find_set(e.u) != find_set(e.v)) {
        cost += e.weight;
        result.push_back(e);
        union_sets(e.u, e.v);
    }
}
```

2.2 Prim

```
int n;
vector<vector<int>> adj; // adjacency matrix of graph
const int INF = 1000000000; // weight INF means there is no edge

struct Edge {
    int w = INF, to = -1;
};

void prim() {
    int total_weight = 0;
    vector<bool> selected(n, false);
    vector<Edge> min_e(n);
    min_e[0].w = 0;

    for (int i=0; i<n; ++i) {
        int v = -1;
        for (int j = 0; j < n; ++j) {
            if (!selected[j] && (v == -1 || min_e[j].w < min_e[v].w))
                v = j;
        }

        if (min_e[v].w == INF) {
            cout << "No MST!" << endl;
            exit(0);
        }

        selected[v] = true;
        total_weight += min_e[v].w;
        if (min_e[v].to != -1)
            cout << v << " " << min_e[v].to << endl;

        for (int to = 0; to < n; ++to) {
            if (adj[v][to] < min_e[to].w)
                min_e[to] = {adj[v][to], v};
        }
    }

    cout << total_weight << endl;
}
```

2.3 Dijkstra

```
const int MAX = 2e5+7;
const int INF = 1000000000;
vector<vector<pair<int, int>>> adj(MAX);

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});
            }
        }
    }
}

vector<int> restore_path(int s, int t) {
    vector<int> path;

    for (int v = t; v != s; v = p[v])
        path.push_back(v);
    path.push_back(s);

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

int adj[MAX][MAX];
```

```cpp
int dist[MAX];
int minDistance(int dist[], bool sptSet[], int V) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

void dijkstra(int src, int V) {

    bool sptSet[V];
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet, V);

        sptSet[u] = true;


        for (int v = 0; v < V; v++)
            if (!sptSet[v] && adj[u][v]
                && dist[u] != INT_MAX
                && dist[u] + adj[u][v] < dist[v])
                dist[v] = dist[u] + adj[u][v];
    }
}
```

2.4 Bellman Ford

```cpp
struct edge
{
    int a, b, cost;
};

int n, m, v;
vector<edge> e;
const int INF = 1000000000;

void solve()
{
    vector<int> d (n, INF);
    d[v] = 0;
    for (int i=0; i<n-1; ++i)
        for (int j=0; j<m; ++j)
            if (d[e[j].a] < INF)
                d[e[j].b] = min (d[e[j].b], d[e[j].a] + e[j].cost);
}
```

2.5 Bipartite

```cpp
const int NONE = 0, BLUE = 1, RED = 2;
vector<vector<int>> graph(100005);
vector<bool> visited(100005);
int color[100005];

bool bfs(int s = 1){

    queue<int> q;
    q.push(s);
    color[s] = BLUE;

    while (not q.empty()){
        auto u = q.front(); q.pop();

        for (auto v : graph[u]){
            if (color[v] == NONE){
                color[v] = 3 - color[u];
                q.push(v);
            }
            else if (color[v] == color[u]){
                return false;
            }
        }
    }

    return true;
}

bool is_bipartite(int n){

    for (int i = 1; i<=n; i++)
        if (color[i] == NONE and not bfs(i))
            return false;

    return true;
}
```

2.6 Floyd Warshall

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

using namespace std;
using ll = long long;

const int MAX  = 507;
const long long INF = 0x3f3f3f3f3f3f3f3fLL;

ll dist[MAX][MAX];
int n;

void floyd_warshall() {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) dist[i][j] = 0;
            else if (!dist[i][j]) dist[i][j] = INF;
        }
    }

    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                // trata o caso no qual o grafo tem arestas com peso negativo
                if (dist[i][k] < INF && dist[k][j] < INF){
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
                }
            }
        }
    }
}
```

2.7 Tree Diameter

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

using namespace std;

const int MAX = 3e5+17;

vector<int> adj[MAX];
bool visited[MAX];

int max_depth = 0, max_node = 1;

void dfs (int v, int depth) {
    visited[v] = true;

    if (depth > max_depth) {
        max_depth = depth;
        max_node = v;
    }

    for (auto u : adj[v]) {
        if (!visited[u]) dfs(u, depth + 1);
    }
}

int tree_diameter() {
    dfs(1, 0);
    max_depth = 0;
    for (int i = 0; i < MAX; i++) visited[i] = false;
    dfs(max_node, 0);
    return max_depth;
}
```

2.8 Centroid Decomposition

```
int n;
vector<set<int>> adj;
vector<char> ans;

vector<bool> removed;

vector<int> subtree_size;

int dfs(int u, int p = 0) {
  subtree_size[u] = 1;

  for(int v : adj[u]) {
    if(v != p && !removed[v]) {
      subtree_size[u] += dfs(v, u);
        }
    }

  return subtree_size[u];
}

int get_centroid(int u, int sz, int p = 0) {
  for(int v : adj[u]) {
    if(v != p && !removed[v]) {
      if(subtree_size[v]*2 > sz) {
        return get_centroid(v, sz, u);
            }
        }
    }

  return u;
}

char get_next(char c) {
    if (c != 'Z') return c + 1;
    return '$';
}

bool flag = true;

void solve(int node, char c) {
  int center = get_centroid(node, dfs(node));
    ans[center] = c;
    removed[center] = true;

    for (auto u : adj[center]) {
        if (!removed[u]) {
            char next = get_next(c);
            if (next == '$') {
                flag = false;
                return;
            }
            solve(u, next);
        }
    }
}

int32_t main(){
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    cin >> n;
    adj.resize(n + 1);
    ans.resize(n + 1);
    removed.resize(n + 1);
    subtree_size.resize(n + 1);

    for (int i = 1; i <= n - 1; i++) {
        int u, v; cin >> u >> v;
        adj[u].insert(v);
        adj[v].insert(u);
    }

    solve(1, 'A');

    if (!flag) cout << "Impossible!\n";
    else {
        for (int i = 1; i <= n; i++) {
            cout << ans[i] << ' ';
        }
        cout << '\n';
    }

    return 0;
}
```

2.9 Lca

```
// Description:
// Find the lowest common ancestor between two nodes in a tree

// Problem:
// https://cses.fi/problemset/task/1688/

// Complexity:
// O(log n)

// How to use:
// preprocess(1);
// lca(a, b);

// Notes
// To calculate the distance between two nodes use the following formula
// dist[a] + dist[b] - 2*dist[lca(a, b)]

const int MAX = 2e5+17;

const int BITS = 32;

vector<int> adj[MAX];
// vector<pair<int, int>> adj[MAX];
// int dist[MAX];

int timer;
vector<int> tin, tout;
vector<vector<int>> up;

void dfs(int v, int p)
{
    tin[v] = ++timer;
    up[v][0] = p;

    for (int i = 1; i <= BITS; ++i) {
        up[v][i] = up[up[v][i-1]][i-1];
    }

    for (auto u : adj[v]) {
        if (u != p) {
            dfs(u, v);
        }
    }

    /*for (auto [u, peso] : adj[v]) {
        if (u != p) {
            dist[u] = dist[v] + peso;
            dfs(u, v);
        }
    }*/

    tout[v] = ++timer;
}

bool is_ancestor(int u, int v)
{
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int u, int v)
{
    if (is_ancestor(u, v))
        return u;
    if (is_ancestor(v, u))
        return v;
    for (int i = BITS; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}

void preprocess(int root) {
    tin.resize(MAX);
    tout.resize(MAX);
    timer = 0;
    up.assign(MAX, vector<int>(BITS + 1));
    dfs(root, root);
}
```

2.10 Ford Fulkerson Edmonds Karp

```
// Description:
// Obtains the maximum possible flow rate given a network. A network is a graph with a single source vertex and a single sink vertex in which each edge has a capacity

// Complexity:
// O(V * E^2) where V is the number of vertex and E is the number of edges

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

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

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

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

    return 0;
}

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

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

    return flow;
}
```

2.11 Dinic

```
// Description:
// Obtains the maximum possible flow rate given a network. A network is a graph with a single source vertex and a single sink vertex in which each edge has a capacity

// Problem:
// https://codeforces.com/gym/103708/problem/J

// Complexity:
// O(V^2 * E) where V is the number of vertex and E is the number of edges

// Unit network
// A unit network is a network in which for any vertex except source and sink either incoming or outgoing edge is unique and has unit capacity (matching problem).
// Complexity on unit networks: O(E * sqrt(V))

// Unity capacity networks
// A more generic settings when all edges have unit capacities, but the number of incoming and outgoing edges is unbounded
// Complexity on unity capacity networks: O(E * sqrt(E))

// How to use:
// Dinic dinic = Dinic(num_vertex, source, sink);
// dinic.add_edge(vertex1, vertex2, capacity);
// cout << dinic.flow() << '\n';

struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u] || edges[id].cap - edges[id].flow < 1)
                continue;
            long long tr = dfs(u, min(pushed, edges[id].cap - edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

2.12 Find Cycle

```
bitset<MAX> visited;
vector<int> path;
vector<int> adj[MAX];

bool dfs(int u, int p){

    if (visited[u]) return false;

    path.pb(u);
    visited[u] = true;

    for (auto v : adj[u]){
        if (visited[v] and u != v and p != v){
            path.pb(v); return true;
        }

        if (dfs(v, u)) return true;
    }

    path.pop_back();
    return false;
}

bool has_cycle(int N){

    visited.reset();

    for (int u = 1; u <= N; ++u){
        path.clear();
        if (not visited[u] and dfs(u,-1))
            return true;

    }

    return false;
}
```

2.13 Tarjan Bridge

```
// Description:
// Find a bridge in a connected unidirected graph
// A bridge is an edge so that if you remove that edge the graph is no longer connected

// Problem:
// https://cses.fi/problemset/task/2177/

// Complexity:
// O(V + E) where V is the number of vertices and E is the number of edges

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

vector<bool> visited;
vector<int> tin, low;
int timer;

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

void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i, -1);
    }
}
```

2.14 Cycle Path Recovery

```
int n;
vector<vector<int>> adj;
vector<char> color;
vector<int> parent;
int cycle_start, cycle_end;

bool dfs(int v) {
    color[v] = 1;
    for (int u : adj[v]) {
        if (color[u] == 0) {
            parent[u] = v;
            if (dfs(u))
                return true;
        } else if (color[u] == 1) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
    }
    color[v] = 2;
    return false;
}

void find_cycle() {
    color.assign(n, 0);
    parent.assign(n, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (color[v] == 0 && dfs(v))
            break;
    }

    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        reverse(cycle.begin(), cycle.end());

        cout << "Cycle found: ";
        for (int v : cycle)
            cout << v << " ";
        cout << endl;
    }
}
```

2.15 Centroid Find

```
// Description:
// Indexed at zero
// Find a centroid, that is a node such that when it is appointed the root of the tree,
// each subtree has at most floor(n/2) nodes.

// Problem:
// https://cses.fi/problemset/task/2079/

// Complexity:
// O(n)

// How to use:
// get_subtree_size(0);
// cout << get_centroid(0) + 1 << endl;

int n;
vector<int> adj[MAX];
int subtree_size[MAX];

int get_subtree_size(int node, int par = -1) {
  int &res = subtree_size[node];
  res = 1;
  for (int i : adj[node]) {
    if (i == par) continue;
    res += get_subtree_size(i, node);
  }
  return res;
}

int get_centroid(int node, int par = -1) {
  for (int i : adj[node]) {
    if (i == par) continue;

    if (subtree_size[i] * 2 > n) { return get_centroid(i, node); }
  }
  return node;
}

int main() {
  cin >> n;
  for (int i = 0; i < n - 1; i++) {
    int u, v; cin >> u >> v;
    u--; v--;
    adj[u].push_back(v);
    adj[v].push_back(u);
  }

  get_subtree_size(0);
  cout << get_centroid(0) + 1 << endl;
}
```

2.16 Small To Large

```
// Problem:
// https://codeforces.com/contest/600/problem/E

void process_colors(int curr, int parent) {

  for (int n : adj[curr]) {
    if (n != parent) {
      process_colors(n, curr);

```

```
            if (colors[curr].size() < colors[n].size()) {
                sum_num[curr] = sum_num[n];
                vmax[curr] = vmax[n];
        swap(colors[curr], colors[n]);
      }

      for (auto [item,vzs] : colors[n]) {
                if(colors[curr][item]+vzs > vmax[curr]){
                    vmax[curr] = colors[curr][item] + vzs;
                    sum_num[curr] = item;
                }
                else if(colors[curr][item]+vzs == vmax[curr]){
                    sum_num[curr] += item;
                }

                colors[curr][item] += vzs;
      }
    }
  }

}


int32_t main() {

  int n; cin >> n;

  for (int i = 1; i <= n; i++) {
    int a; cin >> a;
    colors[i][a] = 1;
        vmax[i] = 1;
        sum_num[i] = a;
  }

  for (int i = 1; i < n; i++) {
    int a, b; cin >> a >> b;

    adj[a].push_back(b);
    adj[b].push_back(a);
  }

  process_colors(1, 0);

  for (int i = 1; i <= n; i++) {
    cout << sum_num[i] << (i < n ? " " : "\n");
  }

    return 0;

}

```

3 Data Structures

3.1 Ordered Set

```
// Description:
// insert(k) - add element k to the ordered set
// erase(k) - remove element k from the ordered set
// erase(it) - remove element it points to from the ordered set
// order_of_key(k) - returns number of elements strictly smaller than k
// find_by_order(n) - return an iterator pointing to the k-th element in the ordered set (counting from zero).

// Problem:
// https://cses.fi/problemset/task/2169/

// Complexity:
// O(log n) for all operations

// How to use:
// ordered_set<int> os;
// cout << os.order_of_key(1) << ’\n;
// cout << os.find_by_order(1) << ’\n;

// Notes
// The ordered set only contains different elements
// By using less_equal<T> instead of less<T> on using ordered_set declaration
// The ordered_set becomes an ordered_multiset
// So the set can contain elements that are equal

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

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

3.2 Priority Queue

```
// Description:
// Keeps the largest (by default) element at the top of the queue

// Problem:
// https://cses.fi/problemset/task/1164/

// Complexity:
// O(log n) for push and pop
// O (1) for looking at the element at the top

// How to use:
// prioriy_queue<int> pq;
// pq.push(1);
// pq.top();
// pq.pop()

// Notes
// To use the priority queue keeping the smallest element at the top

priority_queue<int, vector<int>, greater<int>> pq;
```

3.3 Dsu

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

using namespace std;

const int MAX = 1e6+17;

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;
    }

    int size(int x) {
        return sizes[x];
    }
};

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

    int cities, roads; cin >> cities >> roads;
    vector<int> final_roads;
    int ans = 0;
    DSU dsu = DSU(cities);
    for (int i = 0, a, b; i < roads; i++) {
        cin >> a >> b;
        dsu.unite(a, b);
    }

    for (int i = 2; i <= cities; i++) {
        if (!dsu.same(1, i)) {
            ans++;
            final_roads.push_back(i);
            dsu.unite(1,i);
        }
    }

    cout << ans << '\n';
    for (auto e : final_roads) {
        cout << "1 " << e << '\n';
    }

}
```

3.4 Persistent

```
// Description:
// Persistent segtree allows for you to save the different versions of the segtree between each update
// Indexed at one
// Query - get sum of elements from range (l, r) inclusive
// Update - update element at position id to a value val

// Problem:
// https://cses.fi/problemset/task/1737/

// Complexity:
// O(log n) for both query and update

// How to use:
// vector<int> raiz(MAX); // vector to store the roots of each version
// Segtree seg = Segtree(INF);
// raiz[0] = seg.create(); // null node
// curr = 1; // keep track of the last version

// raiz[k] = seg.update(raiz[k], idx, val); // updating version k
// seg.query(raiz[k], l, r) // querying version k
// raiz[++curr] = raiz[k]; // create a new version based on version k

const int MAX = 2e5+17;
const int INF = 1e9+17;

typedef long long ftype;

struct Segtree {
    vector<ftype> seg, d, e;
    const ftype NEUTRAL = 0;
    int n;

    Segtree(int n) {
        this->n = n;
    }

    ftype f(ftype a, ftype b) {
        return a + b;
    }

    ftype create() {
        seg.push_back(0);
        e.push_back(0);
        d.push_back(0);
        return seg.size() - 1;
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        if (q < ini || p > fim) return NEUTRAL;
        if (pos == 0) return 0;
        if (p <= ini && fim <= q) return seg[pos];
        int m = (ini + fim) >> 1;
        return f(query(e[pos], ini, m, p, q), query(d[pos], m + 1, fim, p, q));
    }

    int update(int pos, int ini, int fim, int id, int val) {
        int novo = create();

        seg[novo] = seg[pos];
        e[novo] = e[pos];
        d[novo] = d[pos];

        if (ini == fim) {
            seg[novo] = val;
            return novo;
        }

        int m = (ini + fim) >> 1;

        if (id <= m) e[novo] = update(e[novo], ini, m, id, val);
        else d[novo] = update(d[novo], m + 1, fim, id, val);

        seg[novo] = f(seg[e[novo]], seg[d[novo]]);

        return novo;
    }

    ftype query(int pos, int p, int q) {
        return query(pos, 1, n, p, q);
    }

    int update(int pos, int id, int val) {
        return update(pos, 1, n, id, val);
    }
};
```

3.5 Minimum And Amount

```
// Description:
// Query - get minimum element in a range (l, r) inclusive
// and also the number of times it appears in that range
// Update - update element at position id to a value val

// Problem:
// https://codeforces.com/edu/course/2/lesson/4/1/practice/contest/273169/problem/C

// Complexity:
// O(log n) for both query and update

// How to use:
// Segtree seg = Segtree(n);
// seg.build(v);

#define pii pair<int, int>
#define mp make_pair
#define ff first
#define ss second

const int INF = 1e9+17;

typedef pii ftype;

struct Segtree {
    vector<ftype> seg;
    int n;
    const ftype NEUTRAL = mp(INF, 0);

    Segtree(int n) {
        int sz = 1;
        while (sz < n) sz *= 2;
        this->n = sz;

        seg.assign(2*sz, NEUTRAL);
    }

    ftype f(ftype a, ftype b) {
        if (a.ff < b.ff) return a;
        if (b.ff < a.ff) return b;

        return mp(a.ff, a.ss + b.ss);
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        if (ini >= p && fim <= q) {
            return seg[pos];
        }

        if (q < ini || p > fim) {
            return NEUTRAL;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        return f(query(e, ini, m, p, q), query(d, m + 1, fim, p, q));
    }

    void update(int pos, int ini, int fim, int id, int val) {
        if (ini > id || fim < id) {
            return;
        }

        if (ini == id && fim == id) {
            seg[pos] = mp(val, 1);

            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        update(e, ini, m, id, val);
        update(d, m + 1, fim, id, val);

        seg[pos] = f(seg[e], seg[d]);
    }

    void build(int pos, int ini, int fim, vector<int> &v) {
        if (ini == fim) {
            if (ini < (int)v.size()) {
                seg[pos] = mp(v[ini], 1);
            }
            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        build(e, ini, m, v);
        build(d, m + 1, fim, v);

        seg[pos] = f(seg[e], seg[d]);
    }

    ftype query(int p, int q) {
        return query(0, 0, n - 1, p, q);
    }

    void update(int id, int val) {
        update(0, 0, n - 1, id, val);
    }

    void build(vector<int> &v) {
        build(0, 0, n - 1, v);
    }

    void debug() {
        for (auto e : seg) {
            cout << e.ff << ' ' << e.ss << '\n';
        }
        cout << '\n';
    }
};
```

3.6 Range Query Point Update

```
// Description:
// Indexed at zero
// Query - get sum of elements from range (l, r) inclusive
// Update - update element at position id to a value val

// Problem:
// https://codeforces.com/edu/course/2/lesson/4/1/practice/contest/273169/problem/B

// Complexity:
// O(log n) for both query and update

// How to use:
// Segtree seg = Segtree(n);
// seg.build(v);

// Notes
// Change neutral element and f function to perform a different operation

// If you want to change the operations to point query and range update
// Use the same segtree, but perform the following operations
// Query - seg.query(0, id);
// Update - seg.update(l, v); seg.update(r + 1, -v);

typedef long long ftype;

struct Segtree {
    vector<ftype> seg;
    int n;
    const ftype NEUTRAL = 0;

    Segtree(int n) {
        int sz = 1;
        while (sz < n) sz *= 2;
        this->n = sz;

        seg.assign(2*sz, NEUTRAL);
    }

    ftype f(ftype a, ftype b) {
        return a + b;
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        if (ini >= p && fim <= q) {
            return seg[pos];
        }

        if (q < ini || p > fim) {
            return NEUTRAL;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        return f(query(e, ini, m, p, q), query(d, m + 1, fim, p, q));
    }

    void update(int pos, int ini, int fim, int id, int val) {
        if (ini > id || fim < id) {
            return;
        }

        if (ini == id && fim == id) {
            seg[pos] = val;

            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        update(e, ini, m, id, val);
        update(d, m + 1, fim, id, val);

        seg[pos] = f(seg[e], seg[d]);
    }

    void build(int pos, int ini, int fim, vector<int> &v) {
        if (ini == fim) {
            if (ini < (int)v.size()) {
                seg[pos] = v[ini];
            }
            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        build(e, ini, m, v);
        build(d, m + 1, fim, v);

        seg[pos] = f(seg[e], seg[d]);
    }

    ftype query(int p, int q) {
        return query(0, 0, n - 1, p, q);
    }

    void update(int id, int val) {
        update(0, 0, n - 1, id, val);
    }

    void build(vector<int> &v) {
        build(0, 0, n - 1, v);
    }

    void debug() {
        for (auto e : seg) {
            cout << e << ' ';
        }
        cout << '\n';
    }
};
```

3.7 Segment With Maximum Sum

```
// Description:
// Query - get sum of segment that is maximum among all segments
// E.g
// Array: 5 -4 4 3 -5
// Maximum segment sum: 8 because 5 + (-4) + 4 = 8
// Update - update element at position id to a value val

// Problem:
// https://codeforces.com/edu/course/2/lesson/4/2/practice/contest/273278/problem/A

// Complexity:
// O(log n) for both query and update

// How to use:
// Segtree seg = Segtree(n);
// seg.build(v);

// Notes
```

```cpp
// The maximum segment sum can be a negative number
// In that case, taking zero elements is the best choice
// So we need to take the maximum between 0 and the query
// max(0LL, seg.query(0, n).max_seg)

using ll = long long;

typedef ll ftype_node;

struct Node {
    ftype_node max_seg;
    ftype_node pref;
    ftype_node suf;
    ftype_node sum;

    Node(ftype_node max_seg, ftype_node pref, ftype_node suf, ftype_node sum) : max_seg(max_seg), pref(pref), suf(suf), sum(sum) {};
};

typedef Node ftype;

struct Segtree {
    vector<ftype> seg;
    int n;
    const ftype NEUTRAL = Node(0, 0, 0, 0);

    Segtree(int n) {
        int sz = 1;
        // potencia de dois mais proxima
        while (sz < n) sz *= 2;
        this->n = sz;

        // numero de nos da seg
        seg.assign(2*sz, NEUTRAL);
    }

    ftype f(ftype a, ftype b) {
        ftype_node max_seg = max({a.max_seg, b.max_seg, a.suf + b.pref});
        ftype_node pref = max(a.pref, a.sum + b.pref);
        ftype_node suf = max(b.suf, b.sum + a.suf);
        ftype_node sum = a.sum + b.sum;

        return Node(max_seg, pref, suf, sum);
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        if (ini >= p && fim <= q) {
            return seg[pos];
        }

        if (q < ini || p > fim) {
            return NEUTRAL;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        return f(query(e, ini, m, p, q), query(d, m + 1, fim, p, q));
    }

    void update(int pos, int ini, int fim, int id, int val) {
        if (ini > id || fim < id) {
            return;
        }

        if (ini == id && fim == id) {
            seg[pos] = Node(val, val, val, val);

            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        update(e, ini, m, id, val);
        update(d, m + 1, fim, id, val);

        seg[pos] = f(seg[e], seg[d]);
    }

    void build(int pos, int ini, int fim, vector<int> &v) {
        if (ini == fim) {
            // se a çãposio existir no array original
            // seg tamanho potencia de dois
            if (ini < (int)v.size()) {
                seg[pos] = Node(v[ini], v[ini], v[ini], v[ini]);
            }
            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        build(e, ini, m, v);
        build(d, m + 1, fim, v);

        seg[pos] = f(seg[e], seg[d]);
    }

    ftype query(int p, int q) {
        return query(0, 0, n - 1, p, q);
    }

    void update(int id, int val) {
        update(0, 0, n - 1, id, val);
    }

    void build(vector<int> &v) {
        build(0, 0, n - 1, v);
    }

    void debug() {
        for (auto e : seg) {
            cout << e.max_seg << ' ' << e.pref << ' ' << e.suf << ' ' << e.sum << '\n';
        }
        cout << '\n';
    }
};
```

3.8 Dynamic Implicit Sparse

```cpp
// Description:
// Indexed at one

// When the indexes of the nodes are too big to be stored in an array
// and the queries need to be answered online so we can't sort the nodes and compress them
// we create nodes only when they are needed so there'll be (Q*log(MAX)) nodes
// where Q is the number of queries and MAX is the maximum index a node can assume

// Query - get sum of elements from range (l, r) inclusive
// Update - update element at position id to a value val

// Problem:
// https://cses.fi/problemset/task/1648

// Complexity:
// O(log n) for both query and update

// How to use:
// MAX is the maximum index a node can assume
// Create a default null node
// Create a node to be the root of the segtree

// Segtree seg = Segtree(MAX);
// seg.create();
// seg.create();

typedef long long ftype;

const int MAX = 1e9+17;

struct Segtree {
    vector<ftype> seg, d, e, lazy;
    const ftype NEUTRAL = 0;
    int n;

    Segtree(int n) {
        this->n = n;
    }

    ftype f(ftype a, ftype b) {
        return a + b;
    }

    ftype create() {
        seg.push_back(0);
        e.push_back(0);
        d.push_back(0);
        return seg.size() - 1;
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        if (q < ini || p > fim) return NEUTRAL;
        if (pos == 0) return 0;
        if (p <= ini && fim <= q) return seg[pos];
        int m = (ini + fim) >> 1;
        return f(query(e[pos], ini, m, p, q), query(d[pos], m + 1, fim, p, q));
    }

    void update(int pos, int ini, int fim, int id, int val) {
        if (ini > id || fim < id) {
            return;
        }

        if (ini == fim) {
            seg[pos] = val;

            return;
        }

        int m = (ini + fim) >> 1;

        if (id <= m) {
            if (e[pos] == 0) e[pos] = create();
            update(e[pos], ini, m, id, val);
        } else {
            if (d[pos] == 0) d[pos] = create();
            update(d[pos], m + 1, fim, id, val);
        }

        seg[pos] = f(seg[e[pos]], seg[d[pos]]);
    }

    ftype query(int p, int q) {
        return query(1, 1, n, p, q);
    }

    void update(int id, int val) {
        update(1, 1, n, id, val);
    }
};
```

3.9 Lazy

```cpp
// Description:
// Query - get sum of elements from range (l, r) inclusive
// Update - add a value val to elementos from range (l, r) inclusive

// Problem:
// https://codeforces.com/edu/course/2/lesson/5/1/practice/contest/279634/problem/A

// Complexity:
// O(log n) for both query and update

// How to use:
// Segtree seg = Segtree(n);
// seg.build(v);

// Notes
// Change neutral element and f function to perform a different operation

typedef long long ftype;

struct Segtree {
    vector<ftype> seg;
    vector<ftype> lazy;
    int n;
    const ftype NEUTRAL = 0;
    const ftype NEUTRAL_LAZY = -1;

    Segtree(int n) {
        int sz = 1;
        while (sz < n) sz *= 2;
        this->n = sz;

        seg.assign(2*sz, NEUTRAL);
        lazy.assign(2*sz, NEUTRAL_LAZY);
    }

    ftype apply_lazy(ftype a, ftype b, int len) {
        if (b == NEUTRAL_LAZY) return a;
        if (a == NEUTRAL_LAZY) return b * len;
        else return a + b * len;
    }

    void propagate(int pos, int ini, int fim) {
        if (ini == fim) {
            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
        lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);

        seg[e] = apply_lazy(seg[e], lazy[pos], m - ini + 1);
        seg[d] = apply_lazy(seg[d], lazy[pos], fim - m);

        lazy[pos] = NEUTRAL_LAZY;
    }

    ftype f(ftype a, ftype b) {
        return a + b;
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        propagate(pos, ini, fim);

        if (ini >= p && fim <= q) {
            return seg[pos];
        }

        if (q < ini || p > fim) {
            return NEUTRAL;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        return f(query(e, ini, m, p, q), query(d, m + 1, fim, p, q));
    }

    void update(int pos, int ini, int fim, int p, int q, int val) {
        propagate(pos, ini, fim);

        if (ini > q || fim < p) {
            return;
        }

        if (ini >= p && fim <= q) {
            lazy[pos] = apply_lazy(lazy[pos], val, 1);
            seg[pos] = apply_lazy(seg[pos], val, fim - ini + 1);

            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        update(e, ini, m, p, q, val);
        update(d, m + 1, fim, p, q, val);

        seg[pos] = f(seg[e], seg[d]);
    }

    void build(int pos, int ini, int fim, vector<int> &v) {
        if (ini == fim) {
            if (ini < (int)v.size()) {
                seg[pos] = v[ini];
            }
            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        build(e, ini, m, v);
        build(d, m + 1, fim, v);

        seg[pos] = f(seg[e], seg[d]);
    }

    ftype query(int p, int q) {
        return query(0, 0, n - 1, p, q);
    }

    void update(int p, int q, int val) {
        update(0, 0, n - 1, p, q, val);
    }

    void build(vector<int> &v) {
        build(0, 0, n - 1, v);
    }

    void debug() {
        for (auto e : seg) {
            cout << e << ' ';
        }
        cout << '\n';
        for (auto e : lazy) {
            cout << e << ' ';
        }
        cout << '\n';
        cout << '\n';
    }
};
```

3.10 Lazy Dynamic Implicit Sparse

```cpp
// Description:
// Indexed at one

// When the indexes of the nodes are too big to be stored in an array
// and the queries need to be answered online so we can't sort the nodes and compress them
// we create nodes only when they are needed so there'll be (Q*log(MAX)) nodes
// where Q is the number of queries and MAX is the maximum index a node can assume

// Query - get sum of elements from range (l, r) inclusive
// Update - update element at position id to a value val

// Problem:
// https://oj.uz/problem/view/IZhO12_apple

// Complexity:
// O(log n) for both query and update

// How to use:
// MAX is the maximum index a node can assume
// Create a default null node
// Create a node to be the root of the segtree

// Segtree seg = Segtree(MAX);
// seg.create();
// seg.create();

typedef long long ftype;

const int MAX = 1e9+17;

typedef long long ftype;

const int MAX = 1e9+17;
```

```cpp

struct Segtree {
    vector<ftype> seg, d, e, lazy;
    const ftype NEUTRAL = 0;
    const ftype NEUTRAL_LAZY = -1;
    int n;

    Segtree(int n) {
        this->n = n;
    }

    ftype apply_lazy(ftype a, ftype b, int len) {
        if (b == NEUTRAL_LAZY) return a;
        else return b * len;
    }

    void propagate(int pos, int ini, int fim) {
        if (seg[pos] == 0) return;

        if (ini == fim) {
            return;
        }

        int m = (ini + fim) >> 1;

        if (e[pos] == 0) e[pos] = create();
        if (d[pos] == 0) d[pos] = create();

        lazy[e[pos]] = apply_lazy(lazy[e[pos]], lazy[pos], 1);
        lazy[d[pos]] = apply_lazy(lazy[d[pos]], lazy[pos], 1);

        seg[e[pos]] = apply_lazy(seg[e[pos]], lazy[pos], m - ini + 1);
        seg[d[pos]] = apply_lazy(seg[d[pos]], lazy[pos], fim - m);

        lazy[pos] = NEUTRAL_LAZY;
    }

    ftype f(ftype a, ftype b) {
        return a + b;
    }

    ftype create() {
        seg.push_back(0);
        e.push_back(0);
        d.push_back(0);
        lazy.push_back(-1);
        return seg.size() - 1;
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        propagate(pos, ini, fim);
        if (q < ini || p > fim) return NEUTRAL;
        if (pos == 0) return 0;
        if (p <= ini && fim <= q) return seg[pos];
        int m = (ini + fim) >> 1;
        return f(query(e[pos], ini, m, p, q), query(d[pos], m + 1, fim, p, q));
    }

    void update(int pos, int ini, int fim, int p, int q, int val) {
        propagate(pos, ini, fim);
        if (ini > q || fim < p) {
            return;
        }

        if (ini >= p && fim <= q) {
            lazy[pos] = apply_lazy(lazy[pos], val, 1);
            seg[pos] = apply_lazy(seg[pos], val, fim - ini + 1);

            return;
        }

        int m = (ini + fim) >> 1;

        if (e[pos] == 0) e[pos] = create();
        update(e[pos], ini, m, p, q, val);

        if (d[pos] == 0) d[pos] = create();
        update(d[pos], m + 1, fim, p, q, val);

        seg[pos] = f(seg[e[pos]], seg[d[pos]]);
    }

    ftype query(int p, int q) {
        return query(1, 1, n, p, q);
    }

    void update(int p, int q, int val) {
        update(1, 1, n, p, q, val);
    }
};
```

3.11 Segtree2d

```cpp
// Description:
// Indexed at zero
// Given a N x M grid, where i represents the row and j the column, perform the following operations
// update(j, i) - update the value of grid[i][j]
// query(j1, j2, i1, i2) - return the sum of values inside the rectangle
// defined by grid[i1][j1] and grid[i2][j2] inclusive

// Problem:
// https://cses.fi/problemset/task/1739/

// Complexity:
// Time complexity:
// O(log N * log M) for both query and update
// O(N * M) for build
// Memory complexity:
// 4 * M * N

// How to use:
// Segtree2D seg = Segtree2D(n, n);
// vector<vector<int>> v(n, vector<int>(n));
// seg.build(v);

// Notes
// Indexed at zero

struct Segtree2D {
    const int MAXN = 1025;
    int N, M;

    vector<vector<int>> seg;

    Segtree2D(int N, int M) {
        this->N = N;
        this->M = M;
        seg.resize(2*MAXN, vector<int>(2*MAXN));
    }

    void buildY(int noX, int lX, int rX, int noY, int lY, int rY, vector<vector<int>> &v){
        if(lY == rY){
            if(lX == rX){
                seg[noX][noY] = v[rX][rY];
            }else{
                seg[noX][noY] = seg[2*noX+1][noY] + seg[2*noX+2][noY];
            }
        }else{
            int m = (lY+rY)/2;

            buildY(noX, lX, rX, 2*noY+1, lY, m, v);
            buildY(noX, lX, rX, 2*noY+2, m+1, rY, v);

            seg[noX][noY] = seg[noX][2*noY+1] + seg[noX][2*noY+2];
        }
    }

    void buildX(int noX, int lX, int rX, vector<vector<int>> &v){
        if(lX != rX){
            int m = (lX+rX)/2;

            buildX(2*noX+1, lX, m, v);
            buildX(2*noX+2, m+1, rX, v);
        }

        buildY(noX, lX, rX, 0, 0, M - 1, v);
    }

    void updateY(int noX, int lX, int rX, int noY, int lY, int rY, int y){
        if(lY == rY){
            if(lX == rX){
                seg[noX][noY] = !seg[noX][noY];
            }else{
                seg[noX][noY] = seg[2*noX+1][noY] + seg[2*noX+2][noY];
            }
        }else{
            int m = (lY+rY)/2;

            if(y <= m){
                updateY(noX, lX, rX, 2*noY+1,lY, m, y);
            }else if(m < y){
                updateY(noX, lX, rX, 2*noY+2, m+1, rY, y);
            }

            seg[noX][noY] = seg[noX][2*noY+1] + seg[noX][2*noY+2];
        }
    }

    void updateX(int noX, int lX, int rX, int x, int y){
        int m = (lX+rX)/2;

        if(lX != rX){
            if(x <= m){
                updateX(2*noX+1, lX, m, x, y);
            }else if(m < x){
                updateX(2*noX+2, m+1, rX, x, y);
            }
        }

        updateY(noX, lX, rX, 0, 0, M - 1, y);
    }

    int queryY(int noX, int noY, int lY, int rY, int aY, int bY){
        if(aY <= lY && rY <= bY) return seg[noX][noY];

        int m = (lY+rY)/2;

        if(bY <= m) return queryY(noX, 2*noY+1, lY, m, aY, bY);
        if(m < aY) return queryY(noX, 2*noY+2, m+1, rY, aY, bY);

        return queryY(noX, 2*noY+1, lY, m, aY, bY) + queryY(noX, 2*noY+2, m+1, rY, aY, bY);
    }

    int queryX(int noX, int lX, int rX, int aX, int bX, int aY, int bY){
        if(aX <= lX && rX <= bX) return queryY(noX, 0, 0, M - 1, aY, bY);

        int m = (lX+rX)/2;

        if(bX <= m) return queryX(2*noX+1, lX, m, aX, bX, aY, bY);
        if(m < aX) return queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);

        return queryX(2*noX+1, lX, m, aX, bX, aY, bY) + queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);
    }

    void build(vector<vector<int>> &v) {
        buildX(0, 0, N - 1, v);
    }

    int query(int aX, int bX, int aY, int bY) {
        return queryX(0, 0, N - 1, aX, bX, aY, bY);
    }

    void update(int x, int y) {
        updateX(0, 0, N - 1, x, y);
    }
};
```

4 Strings

4.1 Lcs

```
// Description:
// Finds the longest common subsquence between two string

// Problem:
// https://codeforces.com/gym/103134/problem/B

// Complexity:
// O(mn) where m and n are the length of the strings

string lcsAlgo(string s1, string s2, int m, int n) {
  int LCS_table[m + 1][n + 1];

  for (int i = 0; i <= m; i++) {
    for (int j = 0; j <= n; j++) {
      if (i == 0 || j == 0)
        LCS_table[i][j] = 0;
      else if (s1[i - 1] == s2[j - 1])
        LCS_table[i][j] = LCS_table[i - 1][j - 1] + 1;
      else
        LCS_table[i][j] = max(LCS_table[i - 1][j], LCS_table[i][j - 1]);
    }
  }

  int index = LCS_table[m][n];
  char lcsAlgo[index + 1];
  lcsAlgo[index] = '\0';

  int i = m, j = n;
  while (i > 0 && j > 0) {
    if (s1[i - 1] == s2[j - 1]) {
      lcsAlgo[index - 1] = s1[i - 1];
      i--;
      j--;
      index--;
    }

    else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])
      i--;
    else
      j--;
  }

  return lcsAlgo;
}
```

4.2 Kmp

```
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

4.3 Z-function

```
vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}
```

4.4 Generate All Sequences Length K

```
// gera todas as ípossveis êsequncias usando as letras em set (de comprimento n) e que tenham tamanho k
// sequence = ""
vector<string> generate_sequences(char set[], string sequence, int n, int k) {
   if (k == 0){
       return { sequence };
   }

   vector<string> ans;
   for (int i = 0; i < n; i++) {
        auto aux = generate_sequences(set, sequence + set[i], n, k - 1);
        ans.insert(ans.end(), aux.begin(), aux.end());
        // for (auto e : aux) ans.push_back(e);
   }

   return ans;
}
```

4.5 Generate All Permutations

```
vector<string> generate_permutations(string s) {
    int n = s.size();
    vector<string> ans;

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

    do {
        ans.push_back(s);
    } while (next_permutation(s.begin(), s.end()));

    return ans;
}
```

5 Algorithms

5.1 Binary Search Last True

```
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;
}
```

5.2 Ternary Search

```
double ternary_search(double l, double r) {
    double eps = 1e-9;              //set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);      //evaluates the function at m1
        double f2 = f(m2);      //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);                    //return the maximum of f(x) in [l, r]
}
```

5.3 Delta-encoding

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

int main(){
    int n, q;
    cin >> n >> q;
    int [n];
    int delta[n+2];

    while(q--){
        int l, r, x;
        cin >> l >> r >> x;
        delta[l] += x;
        delta[r+1] -= x;
    }

    int curr = 0;
    for(int i=0; i < n; i++){
        curr += delta[i];
        v[i] = curr;
    }

    for(int i=0; i< n; i++){
        cout << v[i] << ' ';
    }
    cout << '\n';

    return 0;
}
```

5.4 Lis

```
int lis(vector<int> const& a) {
    int n = a.size();
    vector<int> d(n, 1);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            if (a[j] < a[i])
                d[i] = max(d[i], d[j] + 1);
        }
    }

    int ans = d[0];
    for (int i = 1; i < n; i++) {
        ans = max(ans, d[i]);
    }
    return ans;
}
```

5.5 Binary Search First 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;
}
```

6 Math

6.1 Ceil

```
long long division_ceil(long long a, long long b) {
    return 1 + ((a - 1) / b); // if a != 0
}
```

6.2 Sieve Of Eratosthenes

```
int n;
vector<bool> is_prime(n+1, true);
is_prime[0] = is_prime[1] = false;
for (int i = 2; i <= n; i++) {
    if (is_prime[i] && (long long)i * i <= n) {
        for (int j = i * i; j <= n; j += i)
            is_prime[j] = false;
    }
}
```

6.3 Crt

```
ll crt(const vector<pair<ll, ll>> &vet){
    ll ans = 0, lcm = 1;
    ll a, b, g, x, y;
    for(const auto &p : vet) {
        tie(a, b) = p;
        tie(g, x, y) = gcd(lcm, b);
        if((a - ans) % g != 0) return -1; // no solution
        ans = ans + x * ((a - ans) / g) % (b / g) * lcm;
        lcm = lcm * (b / g);
        ans = (ans % lcm + lcm) % lcm;
    }
    return ans;
}
```

6.4 Check If Bit Is On

```
// msb de 0 é undefined
#define msb(n) (32 - __builtin_clz(n))
// #define msb(n) (64 - __builtin_clzll(n) )
// popcount
// turn bit off

bool bit_on(int n, int bit) {
    if(1 & (n >> bit)) return true;
    else return false;
}
```

6.5 Matrix Exponentiation

```
// Description:
// Calculate the nth term of a linear recursion

// Example Fibonacci:
// Given a linear recurrence, for example fibonacci
// F(n) = n, x <= 1
// F(n) = F(n - 1) + F(n - 2), x > 1

// The recurrence has two terms, so we can build a matrix 2 x 1 so that
// n + 1 = transition * n

// (2 x 1) = (2 x 2) * (2 x 1)
// F(n)     = a b * F(n - 1)
// F(n - 1)   c d   F(n - 2)

// Another Example:
// Given a grid 3 x n, you want to color it using 3 distinct colors so that
// no adjacent place has the same color. In how many different ways can you do that?
// There are 6 ways for the first column to be colored using 3 distinct colors
// ans 6 ways using 2 equal colors and 1 distinct one

// Adding another column, there are:
// 3 ways to go from 2 equal to 2 equal
// 2 ways to go from 2 equal to 3 distinct
// 2 ways to go from 3 distinct to 2 equal
// 2 ways to go from 3 distinct to 3 distinct

// So we star with matrix 6 6 and multiply it by the transition 3 2 and get 18 12
//                       6 6
                   2 2          12 12
```

```
// the we can exponentiate this matrix to find the nth column

// Problem:
// https://cses.fi/problemset/task/1722/

// Complexity:
// O(log n)

// How to use:
// vector<vector<ll>> v = {{1, 1}, {1, 0}};
// Matriz transition = Matriz(v);
// cout << fexp(transition, n)[0][1] << '\n';

using ll = long long;

const int MOD = 1e9+7;

struct Matriz{
    vector<vector<ll>> mat;
    int rows, columns;

    vector<ll> operator[](int i){
        return mat[i];
    }

    Matriz(vector<vector<ll>>& matriz){
        mat = matriz;
        rows = mat.size();
        columns = mat[0].size();
    }

    Matriz(int row, int column, bool identity=false){
        rows = row;  columns = column;
        mat.assign(rows, vector<ll>(columns, 0));
        if(identity) {
            for(int i = 0; i < min(rows,columns); i++) {
                mat[i][i] = 1;
            }
        }
    }

    Matriz operator * (Matriz a) {
        assert(columns == a.rows);
        vector<vector<ll>> resp(rows, vector<ll>(a.columns, 0));

        for(int i = 0; i < rows; i++){
            for(int j = 0; j < a.columns; j++){
                for(int k = 0; k < a.rows; k++){
                    resp[i][j] = (resp[i][j] + (mat[i][k] * 1LL * a[k][j]) % MOD) % MOD;
                }
            }
        }
        return Matriz(resp);
    }

    Matriz operator + (Matriz a) {
        assert(rows == a.rows && columns == a.columns);
        vector<vector<ll>> resp(rows, vector<ll>(columns,0));
        for(int i = 0; i < rows; i++){
            for(int j = 0; j < columns; j++){
                resp[i][j] = (resp[i][j] + mat[i][j] + a[i][j]) % MOD;
            }
        }
        return Matriz(resp);
    }
};

Matriz fexp(Matriz base, ll exponent){
    Matriz result = Matriz(base.rows, base.rows, 1);
    while(exponent > 0){
        if(exponent & 1LL) result = result * base;
        base = base * base;
        exponent  = exponent >> 1;
    }
    return result;
}
```

6.6 Fast Exponentiation

```
ll fexp(ll b, ll e, ll mod) {
    ll res = 1;
    b %= mod;
    while(e){
        if(e & 1LL)
            res = (res * b) % mod;
        e = e >> 1LL;
        b = (b * b) % mod;
    }
    return res;
}
```

6.7 Divisors

```
vector<long long> all_divisors(long long n) {
  vector<long long> ans;
  for(long long a = 1; a*a <= n; a++){
    if(n % a == 0) {
      long long b = n / a;
      ans.push_back(a);
      if(a != b) ans.push_back(b);
    }
  }
  sort(ans.begin(), ans.end());
  return ans;
}
```

6.8 Binary To Decimal

```
int binary_to_decimal(long long n) {
  int dec = 0, i = 0, rem;

  while (n!=0) {
    rem = n % 10;
    n /= 10;
    dec += rem * pow(2, i);
    ++i;
  }

  return dec;
}

long long decimal_to_binary(int n) {
  long long bin = 0;
  int rem, i = 1;

  while (n!=0) {
    rem = n % 2;
    n /= 2;
    bin += rem * i;
    i *= 10;
  }

  return bin;
}
```

6.9 Multiplicative Inverse

```
ll extend_euclid(ll a, ll b, ll &x, ll &y) {
    if (a == 0)
    {
        x = 0; y = 1;
        return b;
    }
    ll x1, y1;
    ll d = extend_euclid(b%a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return d;
}

// gcd(a, m) = 1 para existir solucao
// ax + my = 1, ou a*x = 1 (mod m)
ll inv_gcd(ll a, ll m) { // com gcd
  ll x, y;
  extend_euclid(a, m, x, y);
  return (((x % m) +m) %m);
}

ll inv(ll a, ll phim) { // com phi(m), se m for primo entao phi(m) = p-1
  ll e = phim-1;
  return fexp(a, e, MOD);
}
```

6.10 Prime Factors

```
vector<pair<long long, int>> fatora(long long n) {
  vector<pair<long long, int>> ans;
  for(long long p = 2; p*p <= n; p++) {
    if(n % p == 0) {
      int expoente = 0;
      while(n % p == 0) {
        n /= p;
        expoente++;
      }
      ans.emplace_back(p, expoente);
    }
  }
  if(n > 1) ans.emplace_back(n, 1);
  return ans;
}
```

6.11 Linear Diophantine Equation

```
// int a, b, c, x1, x2, y1, y2; cin >> a >> b >> c >> x1 >> x2 >> y1 >> y2;
// int ans = -1;
// if (a == 0 && b == 0) {
//     if (c != 0) ans = 0;
//     else ans = (x2 - x1 + 1) * (y2 - y1 + 1);
// }
// else if (a == 0) {
//     if (c % b == 0 && y1 <= c / b && y2 >= c / b) ans = (x2 - x1 + 1);
//     else ans = 0;
// }
// else if (b == 0) {
//     if (c % a == 0 && x1 <= c / a && x2 >= c / a) ans = (y2 - y1 + 1);
//     else ans = 0;
// }

// Careful when a or b are negative or zero

// if (ans == -1) ans =  find_all_solutions(a, b, c, x1, x2, y1, y2);
// cout << ans << '\n';

// Problems:
// https://www.spoj.com/problems/CEQU/
// http://codeforces.com/problemsets/acmsguru/problem/99999/106

// consider trivial case a or b is 0
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

// x and y are one solution and g is the gcd, all passed as reference
// minx <= x <= maxx miny <= y <= maxy
bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

void shift_solution(int & x, int & y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}

// return number of solutions in the interval
int find_all_solutions(int a, int b, int c, int minx, int maxx, int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;

    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

7 Misc

7.1 Split

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

    string palTemp = "";
    for(int i = 0; i < txt.size(); i++){

        if(txt[i] == key){
            if(palTemp.size() > 0){
                ans.push_back(palTemp);
                palTemp = "";
            }
        } else{
            palTemp += txt[i];
        }

    }

    if(palTemp.size() > 0)
        ans.push_back(palTemp);

    return ans;
}
```

7.2 Int128

```
__int128 read() {
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}
void print(__int128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x > 9) print(x / 10);
    putchar(x % 10 + '0');
}
```

8 Template

8.1 Template Clean

```
// Notes:
// Compile and execute
// g++ teste.cpp -o teste -std=c++17
// ./teste < teste.txt

// Print with precision
// cout << fixed << setprecision(12) << value << endl;

// File as input and output
// freopen("input.txt", "r", stdin);
// freopen("output.txt", "w", stdout);

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

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



    return 0;
}
```

8.2 Template

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

#define int long long
#define optimize std::ios::sync_with_stdio(false); cin.tie(NULL);
#define vi vector<int>
#define ll long long
#define pb push_back
#define mp make_pair
#define ff first
#define ss second
#define pii pair<int, int>
#define MOD 1000000007
#define sqr(x) ((x) * (x))
#define all(x) (x).begin(), (x).end()
#define FOR(i, j, n) for (int i = j; i < n; i++)
#define qle(i, n) (i == n ? "\n" : " ")
#define endl "\n"
const int oo = 1e9;
const int MAX = 1e6;

int32_t main(){ optimize;

    return 0;
}
```