B1

#include <iostream>

#include <vector>

#include <unordered\_map>

using namespace std;

void dfs(int *node*, unordered\_map<int, vector<int>>& *graph*, unordered\_map<int, bool>& *visited*) {

*visited*[*node*] = true;

    for (int neighbor : *graph*[*node*]) {

        if (!*visited*[neighbor]) {

            dfs(neighbor, *graph*, *visited*);

        }

    }

}

int findConnectedComponents(int *n*, vector<pair<int, int>>& *edges*) {

    unordered\_map<int, vector<int>> graph;

    unordered\_map<int, bool> visited;

    int components = 0;

    for (const auto& edge : *edges*) {

        int u = edge.first;

        int v = edge.second;

        graph[u].push\_back(v);

        graph[v].push\_back(u);

    }

    for (int node = 1; node <= *n*; ++node) {

        if (!visited[node]) {

            dfs(node, graph, visited);

            components++;

        }

    }

    return components;

}

int main() {

    int n, m;

    cin >> n >> m;

    vector<pair<int, int>> edges;

    for (int i = 0; i < m; i++) {

        int x, y;

        cin >> x >> y;

        edges.push\_back({x, y});

    }

    int result = findConnectedComponents(n, edges);

    cout << result << endl;

    return 0;

}

B2

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

int findShortestPath(int *n*, int *m*, int *X*, int *Y*, vector<pair<int, int>>& *edges*) {

    vector<vector<int>> graph(*n* + 1);

    vector<bool> visited(*n* + 1, false);

    queue<pair<int, int>> q;

    for (const auto& edge : *edges*) {

        int u = edge.first;

        int v = edge.second;

        graph[u].push\_back(v);

    }

    q.push({*X*, 0});

    visited[*X*] = true;

    while (!q.empty()) {

        int node = q.front().first;

        int distance = q.front().second;

        q.pop();

        if (node == *Y*) {

            return distance;

        }

        for (int neighbor : graph[node]) {

            if (!visited[neighbor]) {

                q.push({neighbor, distance + 1});

                visited[neighbor] = true;

            }

        }

    }

    return -1;

}

int main() {

    int n, m, X, Y;

    cin >> n >> m >> X >> Y;

    vector<pair<int, int>> edges;

    for (int i = 0; i < m; i++) {

        int x, y;

        cin >> x >> y;

        edges.push\_back({x, y});

    }

    int result = findShortestPath(n, m, X, Y, edges);

    cout << result << endl;

    return 0;

}