Q-1: You are given an undirected graph consisting of N nodes labeled from 0 to N – 1. The graph is represented by a 2D array arr[][], where arr[i] represents all the nodes that are connected to the ith node.

Your task is to determine whether it's possible to visit all the nodes in the graph starting from a given node X.

Note: You can visit a node and then move to any of its connected nodes.

Sample test case:

|  |
| --- |
| Input: { { 1, 2 }, { 0, 3, 2 }, { 0, 1 }, { 1 } } , N=4, X=0  Output: YES |

Solution:

#include <bits/stdc++.h>

using namespace std;

// Function to find if

// all nodes can be visited from X

bool canVisitAllNodes(vector<vector<int> >& arr,

int X, int n)

{

queue<int> q;

vector<int> visited(n, false);

q.push(X);

visited[X] = true;

int count = 0;

// Loop to implement BFS

while (q.size() > 0) {

int size = q.size();

for (int i = 0; i < size; i++) {

auto curr = q.front();

q.pop();

count++;

for (auto j : arr[curr]) {

if (visited[j] == false) {

q.push(j);

visited[j] = true;

}

}

}

}

// Check if all nodes are visited

if (count == n)

return true;

return false;

}

int main()

{

vector<vector<int> > arr

= { { 1, 2 }, { 0, 3, 2 }, { 0, 1 }, { 1 } };

int N = 4, X = 0;

// Function Call

if (canVisitAllNodes(arr, X, N)) {

cout << "YES" << endl;

}

else {

cout << "NO" << endl;

}

return 0;

}

Q-2: You are given an undirected graph with V vertices and E edges, along with a node X. Your task is to determine the level of node X in the undirected graph when traversing the graph starting from vertex 0. If node X is not present in the graph, return -1.

Sample test case:

|  |
| --- |
| Input: V = 5, Edges = {{0, 1}, {0, 2}, {1, 3}, {2, 4}}, X = 1  Output: 1 |

Solution:

#include <bits/stdc++.h>

using namespace std;

// Function to find the level of the given node

int findLevel(int N, vector<vector<int> >& edges, int X)

{

// Variable to store maximum vertex of graph

int maxVertex = 0;

for (auto it : edges) {

maxVertex = max({ maxVertex, it[0], it[1] });

}

// Creating adjacency list

vector<int> adj[maxVertex + 1];

for (int i = 0; i < edges.size(); i++) {

adj[edges[i][0]].push\_back(edges[i][1]);

adj[edges[i][1]].push\_back(edges[i][0]);

}

// If X is not present then return -1

if (X > maxVertex || adj[X].size() == 0)

return -1;

// Initialize a Queue for BFS traversal

queue<int> q;

q.push(0);

int level = 0;

// Visited array to mark the already visited nodes

vector<int> visited(maxVertex + 1, 0);

visited[0] = 1;

// BFS traversal

while (!q.empty()) {

int sz = q.size();

while (sz--) {

auto currentNode = q.front();

q.pop();

if (currentNode == X) {

return level;

}

for (auto it : adj[currentNode]) {

if (!visited[it]) {

q.push(it);

visited[it] = 1;

}

}

}

level++;

}

return -1;

}

int main()

{

int V = 5;

vector<vector<int> > edges= { { 0, 1 }, { 0, 2 }, { 1, 3 }, { 2, 4 } };

int X = 1;

// Function call

int level = findLevel(V, edges, X);

cout << level << endl;

return 0;

}

Q-3: A Hamiltonian path, is a path in an undirected graph that visits each vertex exactly once. Given an undirected graph, the task is to check if a Hamiltonian path is present in it or not. N (the number of vertices), M (Number of edges)

Sample test case:

|  |
| --- |
| Input:  N = 4, M = 4  Edges[][]= { {1,2}, {2,3}, {3,4}, {2,4} }  Output:  1 |

Solution:

#include<bits/stdc++.h>

using namespace std;

class Solution

{

public:

// Function to recursively check for Hamiltonian Path

bool checkHamiltonian(int node, int currVis, int n, int m, vector<int>& vis, vector<int> adj[]) {

// Base case: If all nodes are visited, a Hamiltonian Path is found

if (currVis == n)

return true;

vis[node] = 1; // Mark the current node as visited

for (auto nodes : adj[node]) {

if (!vis[nodes]) {

if (checkHamiltonian(nodes, currVis + 1, n, m, vis, adj))

return true;

}

}

vis[node] = 0; // Backtrack: Mark the current node as unvisited

return false;

}

// Function to check for Hamiltonian Path in the graph

bool check(int N, int M, vector<vector<int>> Edges) {

vector<int> adj[N + 1]; // Adjacency list for the graph

// Creating the adjacency list from the given edges

for (auto vec : Edges) {

int u = vec[0], v = vec[1];

adj[u].push\_back(v);

adj[v].push\_back(u); // Since the graph is undirected

}

// For each node in the graph, check if there's a Hamiltonian Path

for (int i = 1; i <= N; i++) {

vector<int> vis(N + 1, 0); // Initialize the visited array

if (checkHamiltonian(i, 1, N, M, vis, adj))

return true; // If Hamiltonian Path exists, return true

}

return false; // If no Hamiltonian Path is found, return false

}

};

int main()

{

int N, M, X, Y;

cin >> N >> M; // Input: Number of nodes (N) and edges (M)

vector<vector<int>> Edges; // Stores the edges of the graph

// Input the edges from the user

for (int i = 0; i < M; i++) {

cin >> X >> Y;

Edges.push\_back({ X, Y });

}

Solution obj; // Create an instance of the Solution class

if (obj.check(N, M, Edges)) {

cout << "1" << endl; // Output: If Hamiltonian Path exists, print 1

}

else {

cout << "0" << endl; // Output: If no Hamiltonian Path, print 0

}

return 0;

}