****

ADA Lab

Assignment - 8

Dated: 31-03-2021

Sub Code: CSE-228

**Vivek Kumar Ahirwar**

**191112419**

**CSE - 3**

Department of

Computer Science and Engineering

Subject Coordinator:

Prof. Manish Pandey

Contents

[Problem 1: Graph BFS DFS 2](#_Toc68100399)

[Design, Develop and Implement a Program in your preferred language for the following operations on Graph(G) of Cities (Take graph of your choice) 2](#_Toc68100400)

[a) Create a Graph of N cities using Adjacency Matrix. 2](#_Toc68100401)

[b) Print all the nodes reachable from a given starting node in a digraph using BFS method. 2](#_Toc68100402)

[c) Check whether a given graph is connected or not using DFS method 2](#_Toc68100403)

[Code 2](#_Toc68100404)

[Output 4](#_Toc68100405)

[Analysis 5](#_Toc68100406)

# Problem 1: Graph BFS DFS

### Design, Develop and Implement a Program in your preferred language for the following operations on Graph(G) of Cities (Take graph of your choice)

### a) Create a Graph of N cities using Adjacency Matrix.

### b) Print all the nodes reachable from a given starting node in a digraph using BFS method.

### c) Check whether a given graph is connected or not using DFS method

## Code

// Keep Changing....@Vi

#include <iostream>

using namespace std;

class Graph

{

public:

    bool \*\*adjMatrix;

    int numVertices;

    // Initialize the matrix to zero

    Graph(int numVertices)

    {

        this->numVertices = numVertices;

        adjMatrix = new bool \*[numVertices];

        for (int i = 0; i < numVertices; i++)

        {

            adjMatrix[i] = new bool[numVertices];

            for (int j = 0; j < numVertices; j++)

                adjMatrix[i][j] = false;

        }

    }

    void addEdge(int i, int j)

    {

        adjMatrix[i][j] = true;

        adjMatrix[j][i] = true;

    }

    void bfs(int vertex)

    {

        int reach[numVertices];

        bool vis[numVertices];

        for (int i = 0; i < numVertices; i++)

        {

            reach[i] = -1;

            vis[i] = false;

        }

        int start = 0, end = 0;

        reach[start] = vertex;

        vis[vertex] = true;

        while (start <= end)

        {

            int u = reach[start];

            start++;

            for (int i = 0; i < numVertices; i++)

            {

                if (vis[i] == false && adjMatrix[u][i] == true)

                {

                    reach[++end] = i;

                    vis[i] = true;

                }

            }

        }

        cout << "vertex reachable from vertex " << vertex << " are : ";

        for (int i = 1; i <= end; i++)

        {

            cout << reach[i] << " ";

        }

    }

    void dfs(int vertex, bool vis[], int &cnt)

    {

        vis[vertex] = true;

        cnt++;

        for (int i = 0; i < numVertices; i++)

        {

            if (vis[i] == false && adjMatrix[vertex][i] == true)

            {

                dfs(i, vis, cnt);

            }

        }

    }

    void connected()

    {

        bool vis[numVertices];

        for (int i = 0; i < numVertices; i++)

        {

            vis[i] = false;

        }

        int cnt = 0;

        dfs(0, vis, cnt);

        if (cnt == numVertices)

        {

            cout << "\nGraph is connected\n";

        }

        else

        {

            cout << "\nGraph is not connected\n";

        }

    }

};

int main()

{

    Graph g(6);

    g.addEdge(0, 1);

    g.addEdge(1, 2);

    g.addEdge(4, 0);

    g.addEdge(4, 1);

    g.addEdge(2, 3);

    g.addEdge(3, 1);

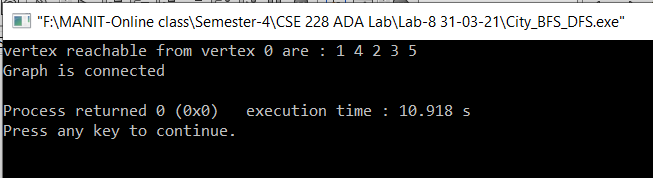
    g.addEdge(3, 5);

    g.bfs(0);

    g.connected();

}

## Output



## Analysis

**Time Complexity**: O(N).

For BFS/DFS

**Auxiliary Space**: O(N\*N).

For storing adjacency matrix