**Algorithm Analysis and Data Structures**

**CS 5343.502(Spring 2020)**

**Assignment 4**

**Divya Birla**

**2021514344**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**QUESTION:**

Make an undirected, but disconnected graph of at least 15 vertices and 25 edges.

You may add function calls in your main to add edges between vertices - one function call per edge.  You may choose any graph representation of your choice from the 3 ways we discussed in the class.

Write a program to do DFS traversal of the graph.

Also write a program to do BFS traversal of the same graph.

Draw your graph on a paper and upload it with your code.

Run each of the two programs and capture screen shots and submit them.

Check that the program does indeed do the correct traversal (based on your hand drawn graph)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**SOURCE CODE:**

**PART 1: DFS Traversal**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Course: CS 5343.502 – Spring 2020

\* Assignment <4>

\* Name: Divya Birla

\* E-mail: dxb190021@utdallas.edu

\* Submitted: <03/31/20>

This program performs DFS traversal on an undirected, disconnected graph having 15 vertices and 25 edges.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

// Creating the graph using adjacency list

vector< vector<int> > CreateGraphAdjList()

{

vector< vector<int> > adjList;

// To create adjacency list size

const int n = 15;

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

{

// intialising list with vertices

vector<int> list;

adjList.push\_back(list);

}

//Adding edges using 2D vector

// adding edges incident on vertex 0

adjList[0].push\_back(1);

adjList[0].push\_back(3);

// adding edges incident on vertex 1

adjList[1].push\_back(0);

adjList[1].push\_back(2);

adjList[1].push\_back(4);

// adding edges incident on vertex 2

adjList[2].push\_back(1);

adjList[2].push\_back(4);

adjList[2].push\_back(5);

// adding edges incident on vertex 3

adjList[3].push\_back(0);

adjList[3].push\_back(4);

adjList[3].push\_back(5);

// adding edges incident on vertex 4

adjList[4].push\_back(1);

adjList[4].push\_back(2);

adjList[4].push\_back(3);

adjList[4].push\_back(5);

// adding edges incident on vertex 5

adjList[5].push\_back(2);

adjList[5].push\_back(3);

adjList[5].push\_back(4);

// adding edges incident on vertex 6

adjList[6].push\_back(7);

adjList[6].push\_back(9);

adjList[6].push\_back(11);

// adding edges incident on vertex 7

adjList[7].push\_back(6);

adjList[7].push\_back(8);

adjList[7].push\_back(9);

// adding edges incident on vertex 8

adjList[8].push\_back(7);

adjList[8].push\_back(11);

adjList[8].push\_back(12);

// adding edges incident on vertex 9

adjList[9].push\_back(6);

adjList[9].push\_back(7);

adjList[9].push\_back(10);

adjList[9].push\_back(11);

// adding edges incident on vertex 10

adjList[10].push\_back(9);

adjList[10].push\_back(11);

adjList[10].push\_back(14);

// adding edges incident on vertex 11

adjList[11].push\_back(6);

adjList[11].push\_back(8);

adjList[11].push\_back(9);

adjList[11].push\_back(10);

adjList[11].push\_back(12);

// adding edges incident on vertex 12

adjList[12].push\_back(8);

adjList[12].push\_back(11);

adjList[12].push\_back(13);

adjList[12].push\_back(14);

// adding edges incident on vertex 13

adjList[13].push\_back(12);

adjList[13].push\_back(14);

// adding edges incident on vertex 14

adjList[14].push\_back(10);

adjList[14].push\_back(11);

adjList[14].push\_back(12);

adjList[14].push\_back(13);

return adjList;

}

//Function for DFS traversal

void AdjListDFS(vector< vector<int> > &adjList, int &vertex, bool \*visited)

{

visited[vertex] = true;

cout << vertex << " ";

for (int i = 0; i < adjList[vertex].size(); i++)

{

int adjNode = adjList[vertex][i];

if (visited[adjNode] == false)

{

AdjListDFS(adjList, adjNode, visited);

}

}

}

//Function for choosing starting node for traversal

void DFSStartVertexSearch(vector< vector<int> > &adjList)

{

int n = adjList.size();

bool visited[15] = { false };

for (int j = 0; j < n; j++) {

if (visited[j] == false) {

AdjListDFS(adjList, j, visited);

}

}

cout << endl << endl;

}

int main()

{

cout << "\n--PROGRAM TO PERFORM DFS TRAVERSAL FOR UNIDRECTED DISCONNECTED GRAPH--\n";

vector< vector<int> > adjList = CreateGraphAdjList();

cout << "\n DFS for the graph is as follows:" << endl;

cout << "\n ";

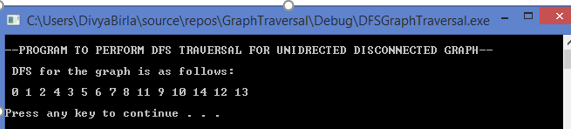
DFSStartVertexSearch(adjList);

system("pause");

return 0;

}

**OUTPUT:**



**PART 2: BFS Traversal**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Course: CS 5343.502 – Spring 2020

\* Assignment <4>

\* Name: Divya Birla

\* E-mail: dxb190021@utdallas.edu

\* Submitted: <03/31/20>

This program performs BFS traversal on an undirected, disconnected graph having 15 vertices and 25 edges.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

// Creating the graph using adjacency list

vector< vector<int> > CreateGraphAdjList()

{

vector< vector<int> > adjList;

// To create adjacency list size

const int n = 15;

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

{

// intialising list with vertices

vector<int> list;

adjList.push\_back(list);

}

//Adding edges using 2D vector

// adding edges incident on vertex 0

adjList[0].push\_back(1);

adjList[0].push\_back(3);

// adding edges incident on vertex 1

adjList[1].push\_back(0);

adjList[1].push\_back(2);

adjList[1].push\_back(4);

// adding edges incident on vertex 2

adjList[2].push\_back(1);

adjList[2].push\_back(4);

adjList[2].push\_back(5);

// adding edges incident on vertex 3

adjList[3].push\_back(0);

adjList[3].push\_back(4);

adjList[3].push\_back(5);

// adding edges incident on vertex 4

adjList[4].push\_back(1);

adjList[4].push\_back(2);

adjList[4].push\_back(3);

adjList[4].push\_back(5);

// adding edges incident on vertex 5

adjList[5].push\_back(2);

adjList[5].push\_back(3);

adjList[5].push\_back(4);

// adding edges incident on vertex 6

adjList[6].push\_back(7);

adjList[6].push\_back(9);

adjList[6].push\_back(11);

// adding edges incident on vertex 7

adjList[7].push\_back(6);

adjList[7].push\_back(8);

adjList[7].push\_back(9);

// adding edges incident on vertex 8

adjList[8].push\_back(7);

adjList[8].push\_back(11);

adjList[8].push\_back(12);

// adding edges incident on vertex 9

adjList[9].push\_back(6);

adjList[9].push\_back(7);

adjList[9].push\_back(10);

adjList[9].push\_back(11);

// adding edges incident on vertex 10

adjList[10].push\_back(9);

adjList[10].push\_back(11);

adjList[10].push\_back(14);

// adding edges incident on vertex 11

adjList[11].push\_back(6);

adjList[11].push\_back(8);

adjList[11].push\_back(9);

adjList[11].push\_back(10);

adjList[11].push\_back(12);

// adding edges incident on vertex 12

adjList[12].push\_back(8);

adjList[12].push\_back(11);

adjList[12].push\_back(13);

adjList[12].push\_back(14);

// adding edges incident on vertex 13

adjList[13].push\_back(12);

adjList[13].push\_back(14);

// adding edges incident on vertex 14

adjList[14].push\_back(10);

adjList[14].push\_back(11);

adjList[14].push\_back(12);

adjList[14].push\_back(13);

return adjList;

}

//Function for BFS traversal

void AdjListBFS(vector< vector<int> > adjList)

{

int n = adjList.size();

//Array to keep track of visited nodes

bool visited[15] = { false };

//Queue to push and pop elements visited and processed in BFS traversal

queue<int> q;

for (int j = 0; j < n; j++) {

//To check if vertex has been visited

if (visited[j] == false) {

q.push(j);

visited[j] = true;

while (q.empty() == false)

{

int vertex = q.front();

q.pop();

cout << vertex << " ";

for (int i = 0; i < adjList[vertex].size(); i++)

{

int adjNode = adjList[vertex][i];

if (visited[adjNode] == false)

{

q.push(adjNode);

visited[adjNode] = true;

}

}

}

}

}

cout << endl << endl;

}

int main()

{

cout << "\n--PROGRAM TO PERFORM BFS TRAVERSAL FOR UNIDRECTED DISCONNECTED GRAPH--\n";

vector< vector<int> > adjList = CreateGraphAdjList();

cout << "\n BFS for the graph is as follows:" << endl;

cout << "\n ";

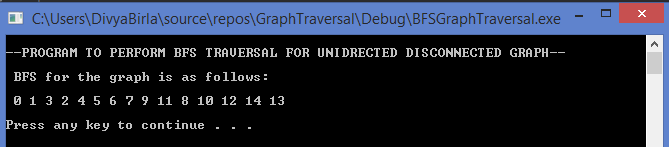
AdjListBFS(adjList);

system("pause");

return 0;

}

**OUTPUT:**



**HAND DRAWN TREE :**

