# 

# Department of Software Engineering

**CS 250: Data Structures and Algorithms**

**Class: BESE-7AB**

**Lab 12: Graph Traversals**

**CLO1: Understand the fundamentals of data structures and algorithms.**

**Date: December 29th, 2017**

**Time: 9:00 am -12:00pm, 2:00pm – 5:00pm**

# Instructor: Dr. Muhammad Shahzad

**Name Nimra**

**Class: BESE-7b**

**Registration No.199281**

# 

**Lab 12: Graph Traversals: Breadth First Traversal (BFS)**

**Task**

#include<iostream>

#include <list>

using namespace std;

class Graph

{

list<int> \*adjacency;

int Vertices;

public:

Graph(int Vertices);

void BFS(int s);

void addEdge(int v, int w);

};

void Graph::addEdge(int v, int w)

{

adjacency[v].push\_back(w);

}

Graph::Graph(int V)

{

this->Vertices = V;

adjacency = new list<int>[V];

}

void Graph::BFS(int s)

{

int d[16];

bool \*visited = new bool[Vertices];

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

visited[i] = false;

d[i] = 0;

}

// Create a queue for BFS

list<int> queue;

// Mark the current node as visited and enqueue it

visited[s] = true;

queue.push\_back(s);

int Ver = s;

list<int>::iterator i;

while (!queue.empty())

{

s = queue.front();

queue.pop\_front();

cout << "The distance of the " << Ver << " from " << s << " is " << d[s] << endl;

// Get all adjacent vertices of the dequeued

// vertex s. If a adjacent has not been visited,

// then mark it visited and enqueue it

for (i = adjacency[s].begin(); i != adjacency[s].end(); ++i)

{

if (!visited[\*i])

{

visited[\*i] = true;

d[\*i] = d[s] + 1;

queue.push\_back(\*i);

}

}

}

}

int main()

{

cout << "Following is Breadth First Traversal "

<< "which is starting from 2 vertice \n";

Graph g(16);

g.addEdge(1, 5);

g.addEdge(5, 1);

g.addEdge(1, 2);

g.addEdge(2, 1);

g.addEdge(2, 6);

g.addEdge(6, 2);

g.addEdge(6, 3);

g.addEdge(3, 6);

g.addEdge(6, 7);

g.addEdge(7, 6);

g.addEdge(3, 7);

g.addEdge(7, 3);

g.addEdge(3, 4);

g.addEdge(4, 3);

g.addEdge(7, 4);

g.addEdge(4, 7);

g.addEdge(8, 7);

g.addEdge(7, 8);

g.addEdge(4, 8);

g.addEdge(8, 4);

g.BFS(2);

system("Pause");

return 0;

}

**OUTPUT**

