# Department of Computing

# School of Electrical Engineering and Computer Science

**CS-250: Data Structure and Algorithms**

**Class: BESE 13A**

# Lab 12: Graph Traversal

**Name:** Aimen Munawar

**Class:** BESE-13-A

**CMS ID:** 415867

**Date: 15th December 2023**

**Time: 10 am - 1 pm & 2 pm – 5 pm**

**Lab Engineer: Anum Asif**

# 

# Lab 12: Graph Traversal

**Introduction**

This lab is based on the graph traversals.

**Objectives**

Objective of this lab is to implement Breadth First Traversal algrithm for traversing nodes in a graph.

**Tools/Software Requirement**

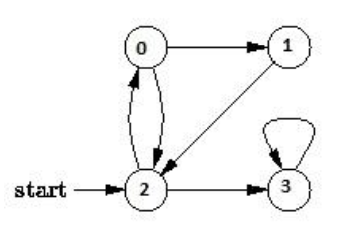
Visual Studio 2012 or gcc or g++

**Helping Material**

Lecture slides, text book

**Description**

Breadth First Traveral (or Search) for a graph is similar to Breadth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex. For example, in the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don’t mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Breadth First Traversal of the following graph is 2, 0, 3, 1.



## BFS algorithm

A standard BFS implementation puts each vertex of the graph into one of two categories:

1. Visited
2. Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

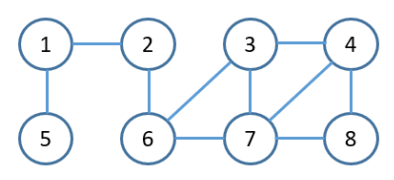
The algorithm works as follows:

1. Start by putting any one of the graph's vertices at the back of a queue.
2. Take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
4. Keep repeating steps 2 and 3 until the queue is empty.

The graph might have two different disconnected parts so to make sure that we cover every vertex, we can also run the BFS algorithm on every node

**Lab Tasks**

Implement BFS traversal algorithm studied in class for a graph with 8 vertices.  
As an initial input we have the graph (G) with 8 nodes shown below with and a starting vertex 2.



**Code:**

#include <iostream>

#include <queue>

using namespace std;

class GraphRepresentation {

int numVertices;

int\*\* adjacencyMatrix;

public:

// Constructor

GraphRepresentation(int numberOfVertices) : numVertices(numberOfVertices) {

// Dynamically allocate memory for the adjacency matrix

adjacencyMatrix = new int\*[numVertices];

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

adjacencyMatrix[i] = new int[numVertices];

// Initialize the matrix with zeros

fill\_n(adjacencyMatrix[i], numVertices, 0);

}

}

// Function to add an edge to the graph

void addEdge(int startVertex, int endVertex) {

startVertex--;

endVertex--;

// Mark the edge in the adjacency matrix

adjacencyMatrix[startVertex][endVertex] = 1;

adjacencyMatrix[endVertex][startVertex] = 1;

}

// Function to perform BFS traversal starting from a given source vertex

void performBFS(int startingVertex) {

// Array to keep track of visited vertices

bool\* visited = new bool[numVertices]{false};

// Queue for BFS

queue<int> bfsQueue;

startingVertex--;

// Mark the starting vertex as visited and enqueue it

visited[startingVertex] = true;

bfsQueue.push(startingVertex);

// BFS traversal loop

while (!bfsQueue.empty()) {

int currentVertex = bfsQueue.front();

cout << currentVertex + 1 << " "; // Adjusting for 1-based indexing in output

bfsQueue.pop();

// Explore neighbors of the current vertex

for (int neighbor = 0; neighbor < numVertices; ++neighbor) {

if (adjacencyMatrix[currentVertex][neighbor] == 1 && !visited[neighbor]) {

visited[neighbor] = true;

bfsQueue.push(neighbor);

}

}

}

// Clean up dynamically allocated memory

delete[] visited;

}

// Destructor

~GraphRepresentation() {

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

delete[] adjacencyMatrix[i];

}

delete[] adjacencyMatrix;

}

};

int main() {

// Creating an instance of the graph representation

GraphRepresentation graph(8);

// Adding edges to the graph

graph.addEdge(1, 5);

graph.addEdge(1, 2);

graph.addEdge(2, 6);

graph.addEdge(6, 3);

graph.addEdge(6, 7);

graph.addEdge(7, 3);

graph.addEdge(7, 4);

graph.addEdge(7, 8);

graph.addEdge(3, 4);

graph.addEdge(3, 6);

graph.addEdge(4, 8);

// Performing BFS traversal starting from vertex 2

cout << "BFS Traversal starting from vertex 2: ";

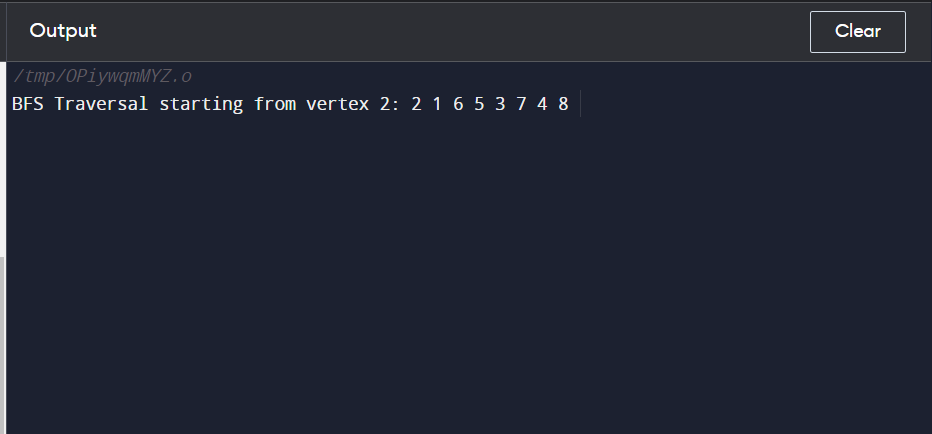
graph.performBFS(2);

return 0;

}

**Expected Output:** 2 1 6 5 3 7 4 8

**Output:**

****

### Deliverables

Compile a single word document by filling in the solution part and submit this Word file on LMS. Insert the solution/answer in this document. You must show the implementation of the tasks in the designing tool, along with your complete Word document to get your work graded. You must also submit this Word document on the LMS.