**Assignment 3: SEARCHING WITH GRAPHS**

Department of Computer Engineering

Dr Mélanie Bouroche

**Submitted By:**

Aniket Agarwal

Student ID: 17317437

**TASK 1:**

**DFS**

Depth First Search is a systematic way to find all the vertices reachable from a source vertex. Like BFS, DFS traverse through a given graph that I created using the Adjacency List. The basic idea behind DFS is that it must explore each and every edge, methodically. I start over from different source vertex ‘A’ as asked in the task1. As soon as I discover a vertex, DFS starts exploring from it (unlike BFS, which puts a vertex on a queue so that it explores from it later). To implement this I have used recursive function for knowing the BFS traversal.

Two stages involved:

1. Unvisited vertices marked as 0
2. As soon as it visits the new vertex print it and mark it as 1 by incrementing count and running a recursive function to visit all the other vertices connected to that vertex.

**BFS**

Breadth First Search is one of the many ways to find the vertices which are reachable from a given source vertex. Like the Depth first search it access all the vertices from the given source s. I have used the Queue the FIFO data structure to solve the problem.

I started the BFS from the vertex ‘A’ which was at level 0. Firstly I visit all the vertices that are at the distance of only one edge. When I visit I mark it as visited, the vertices that are adjacent to that vertex I just visited that is at Level 1. In the next step I visit all the new vertices I can reach at the distance of two edges from the starting vertex. Following this the BFS traversal finally terminates when every vertex has been visited.

So there were basically two stages:

1. Unvisited vertex marked as 0
2. Visited marked as 1 or incremented when visited.

**TASK 2 and TASK 3:**

Dijkstra algorithm can solve the single source shortest path problem for both directed and undirected graphs when all edges have non-negative weights. It is a part of greedy algorithm. Algorithm starts at the source vertex ‘A’ for task2 and 300 for task3 it grows a graph, which ultimately spans all vertices reachable from the source vertex. This leads us to the graph G which is represented by an adjacency lists.

In the following tasks EXTRACT\_MIN operations takes O (log V) time and there are V such operations. The binary heap can be built in O (V) time. Hence, the running time of the Dijkstra algorithm with binary heap provided given graph is sparse is O ((V + E) log V). This time becomes O (E log V) if all vertices in the graph is reachable from the source vertices.

The main reason for implementing Dijkstra using heap in the second task was so that I could later apply it on third task. The third task is just an extension of the second task in which I have added parsers to take data from edges.csv to create the graph and then know the path using Dijkstra Algorithm and then compare the starting vertex with the Stop ID in the vertices.csv and then print all the details from the vertices.csv.

The values for the no. of edges and vertices were calculated and are being displayed in the output only when the program is run. Algorithm used is mostly based on the one mentioned in lecture notes and referred from geekforgeeks.