Lab 11 Report

Chad Lape and Colton Murray

Objectives

This lab explored the concept of the abstract data type of graphs. This data structure is useful in many applications by being able to put a concept of connected nodes into the form of data. This is helpful through allowing computers to then simulate things like a path from one point to another through highways or roads. This will be helpful in the course as it provides another structure in our toolbelt to be able to use effectively and to know when it may be necessary to use them. In a career of CS this is helpful as it can allow for data which is stored and can be traversed only by going through the proper paths or directions.

Task 2: Screenshot

```
Enter first vertice:
3
Enter second vertice:
2
Edge between 3 and 2 created...

Press 1 to add an edge to graph.
Press 2 remove an edge from graph.
Press 3 Find an edge in the graph.
Press 4 Find the out edges of a vertices
Press 5 Find the in edges of a vertices
Press 6 to DFS
Press 7 to BFS
Press 8 to Quit

Choice:  2

Enter first vertice:
0
Enter second vertice:
1
Edge between 0 and 1 removed...

Press 1 to add an edge to graph.
Press 2 remove an edge from graph.
Press 3 Find an edge in the graph.
Press 4 Find the out edges of a vertices
Press 5 Find the in edges of a vertices
Press 6 to DFS
Press 7 to BFS
Press 8 to Quit

Choice:  3

Enter first vertice:
1
Enter second vertice:
2
Edge between 1 and 2 found...

Press 1 to add an edge to graph.
Press 2 remove an edge from graph.
Press 3 Find an edge in the graph.
Press 4 Find the out edges of a vertices
Press 5 Find the in edges of a vertices
Press 6 to DFS
Press 7 to BFS
Press 8 to Quit

Choice:  4

Enter vertice:
1
The out edges for 1 are:
->3
->0
```

Task 3: Screenshot

```
Choice:  6

Enter vertice:
0
Verticies visited in DFS 0 are:
->0
->1
->2
->3

Press 1 to add an edge to graph.
Press 2 remove an edge from graph.
Press 3 Find an edge in the graph.
Press 4 Find the out edges of a vertices
Press 5 Find the in edges of a vertices
Press 6 to DFS
Press 7 to BFS
Press 8 to Quit
```

Task 4: Screenshot

```
Choice:  7

Enter vertice:
3
Verticies visited in BFS3 are:
->3
->1
->0
->2

Press 1 to add an edge to graph.
Press 2 remove an edge from graph.
Press 3 Find an edge in the graph.
Press 4 Find the out edges of a vertices
Press 5 Find the in edges of a vertices
Press 6 to DFS
Press 7 to BFS
Press 8 to Quit
```

Task 5: Depth and Breadth First Search

In order to implement the Depth First Search I had Used two functions. One was forward facing in the public part and the other was a private member. The first would set up an array of Booleans which represented the already visited vertices as well as an output linked list to be passed by pointer reference and which will be returned. The first function only receives the first vertices to check and then recursively calls its helper, passing the vertices being checked and the other pieces of data. Then these will recursively first call the vertices children adding in order the vertices checked to the output linked list and updating the visited list. If there are no valid targets, then it moves up the stack to its parent to then check its sibling. Thus, being a depth first search.

The Breadth Search then posed a different challenge as it didn't seem easy to do with recursion but then at the same time there would need to be a system which was able to store the values to check and remove them in a first in first out order... so I grabbed a queue class from an earlier class and fixed it up to work with this data. I would pass in order the first of the vertices to check, the program would then enqueue its children; the program would store the visited nodes

in another Boolean array and add the index of the vertices to a linked list as it checked them. Then just dequeue the next vertices to check and repeat. The program will check first the siblings then the first child's children and so on.

One should choose DFS when wishing to find all possible paths from a certain point, an example being mazes where one could then check every branch possible and find the exit. While BFS useful for finding the shortest path between two vertices.

Contributions:

All Group Members did equal work a deserve equal credit.

Chad Lape:

- Created and implemented graph and DFS/BFS
- Fixed some import errors of the older Linked_List and Graph classes
- Wrote Objectives and Task 5 of Lab Report

Colton Murray:

- Created Testing Class
- Debugged minor issues with graph class
- Completed Lab Report

Compilation:

Recommended to compile using mingw and g++ if on windows. If on linux just utilize the make file. There is also included executables for both Linux and Windows of working versions of the program. Please reach out if a program doesn't work as it may be a small mistake on our end ie. forgetting a file.