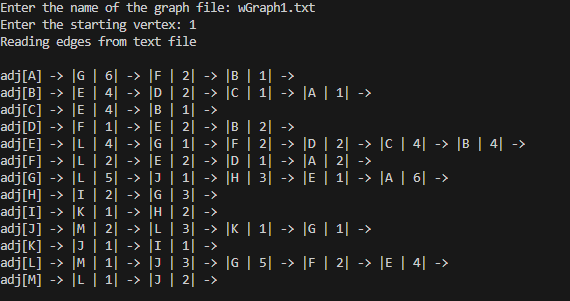
**Algorithms & Data Structures CA2 Project**

**By Harry Delaney C23361046**

**1. Introduction:**

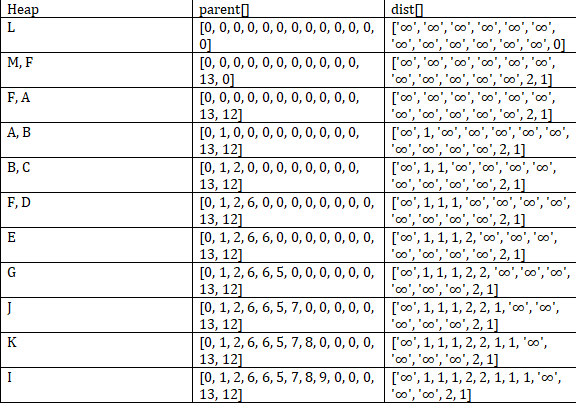
This report details the implementation of two Java files, GraphAlgorithms.java and dijkstra.java. The GraphAlgorithms.java file implements several graph algorithms, including Depth First Search (DFS), Breadth First Search (BFS), Prim's algorithm for Minimum Spanning Tree (MST), and Dijkstra's algorithm for shortest paths. These algorithms operate on an undirected, weighted graph represented using an adjacency structure, with the graph data loaded from a user-specified file (e.g., "wGraph1.txt"). The user also selects the starting vertex for the algorithms.

The dijkstra.java file only implements Dijkstra’s algorithm for shortest path, but in this case is used on a large real-world graph, rather than a smaller graph, which was used for the previous program. The user will also prompt the starting vertex for this.  
  
The program will give us graphs in the terminal to visualize how the different graph algorithms are working.

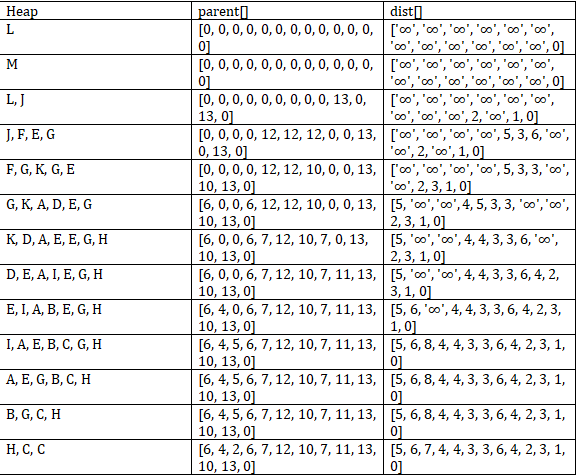
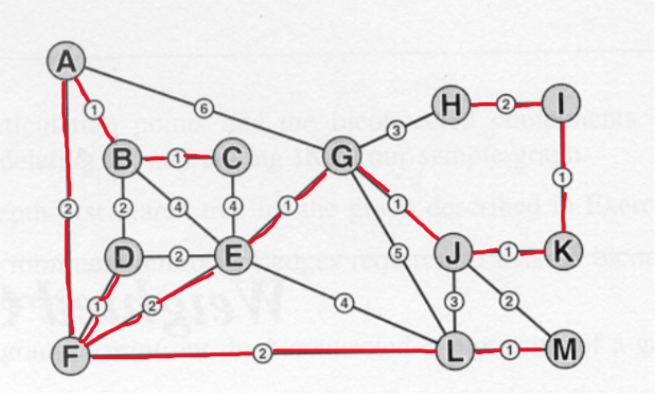
**2. Adjacency Lists Diagram**  
  


In the adjacency list, each line represents a vertex, i.e. a, b, c etc. And its corresponding vertex, i.e. for A we have G, F and B, with weights 6, 2 and 1 respectively.

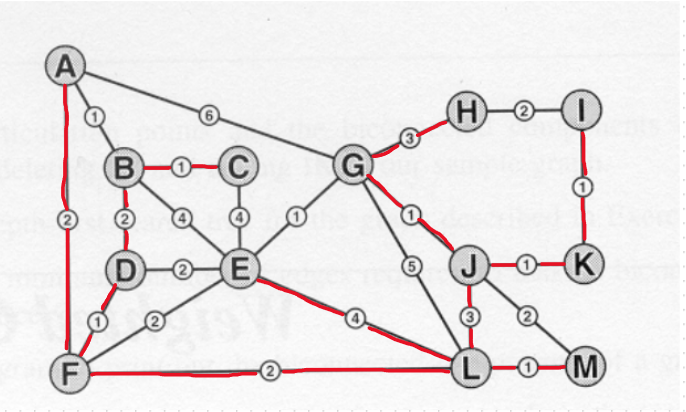
**3. Step by Step Construction of the MST**



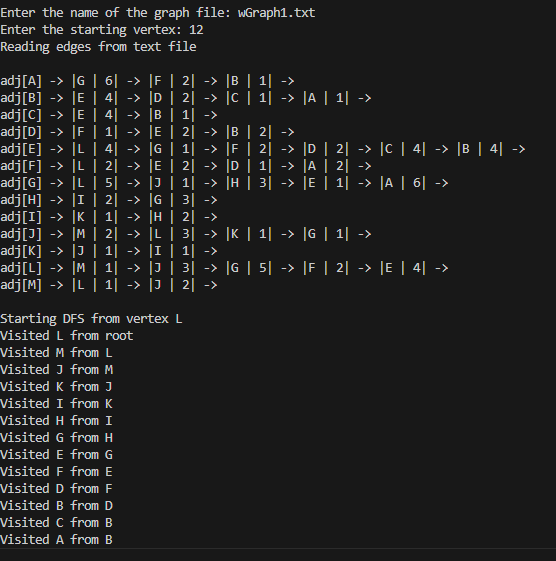
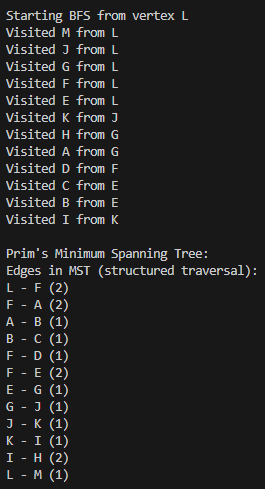
Above is a step-by-step construction of the MST using prims algorithm starting from vertex L.

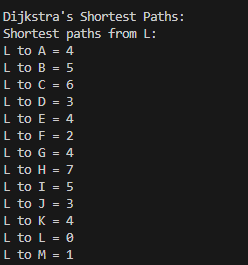
**4. Step by Step Construction of the SPT**  
  
  
  
**5. MST Diagram**  
  


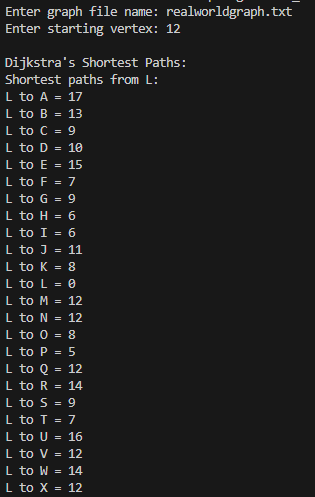
**6. SPT Diagram**



**7. Screen Captures of the programs executing**  
  
GraphAlgorithms.java:



dijkstra.java:   


**8. Dijkstra on the real-world graph**

The real-world graph used consisted of 24 vertices and 40 edges. The size of this graph proves large enough to challenge the Dijkstra program, but not too large to make the output challenging to understand, process and read. It also allows for Dijkstra to display its strengths well/

Strengths:  
1. Accuracy/Correctness: It accurately computes the shortest paths from a source node to the other nodes in the graph.

2. Efficiency with the priority queue: Thanks to PriorityQueue, the algorithm handles the edge relaxations efficiently.

3. It handles cycles well thanks to the visited array, preventing infinite loops from occurring

4. Accurate Parent tracking: It allows for future path reconstruction if needed.  
  
**9. Discussion/Analysis/Reflection**  
  
Through creation and implementation of this java project, I have bettered my understanding of the different traversal methods, such as Depth First traversal and Breadth First Traversal. My understanding of the minimum spanning tree using Prim's algorithm has also greatly improved, and I am now confident I can find the minimum spanning tree of a graph if given to me. My understanding of Dijkstra has also improved. The implementation of the systems in code has also allowed me to better understand how each of the searches works, from a theory standpoint. This knowledge will greatly aid me going forward in my Algorithms & Data Structures study, and for my degree.  
  
  
  
On reflection, while implementing my program, I believe it would have been better to make smaller programs at first, with each individual function, e.g. a DFS.java file, a BFS.java file etc., and then try to implement them as one big program. This would have allowed me to specifically work on problem areas of certain code in their own file, without it causing the whole file not to work, as they were altogether. In the end though I got all the code working, but slightly different implementation may have helped me when developing.