A. Introduction

This laboratory exercise is to implement graphs using an adjacency list implementation and an adjacency matrix implementation and study the time completion of Kruskal's minimum spanning tree algorithm and testing the connectivity of a graph if it all nodes within the graph are connected with at least one member in the graph and can be accessed from any node in the class.

B. Design Approach

Two classes NamedEdge and NumberedEdge were created with field names that hold values for node references where Strings were value references for NamedEdge while integers were that for NumberedEdge. These classes also stored weight values as floats.

An abstract class UndirectedWeightedGraph was created to create abstract functions to be implemented in children classes ListUndirectedWeightedGraph and MatrixUndirectedWeightedGraph which implements graph using adjacency list and adjacency matrix respectively.

A simulation class tests the performance of Kruskal's minimum spanning tree algorithm and the search by depth algorithm which tests to see if all elements are connected to the graph by recording 100 trials and averaging the time.

C. Method

A data file was used to send information into the program. This data file held probabilities to be tested.

Values used were probability values of 0.1 to 0.9.

The program was then run, the output data containing the average time of performance for each algorithm for each implementation was received and data was then plotted onto the graph.

Method of obtaining runtime was using Java's `System.NanoTime()` which is not stable due to its poor accuracy. Its time reference is arbitrarily set, even to a time in the future. Reasons why it was chosen was because `System.currentTimeMillis()` was not giving any values apart from zeros.

Thus, if the initial time and final time are recorded from `System.NanoTime()` with different references the true time taken to execute the algorithm might be wrong.

D. Data and Analysis

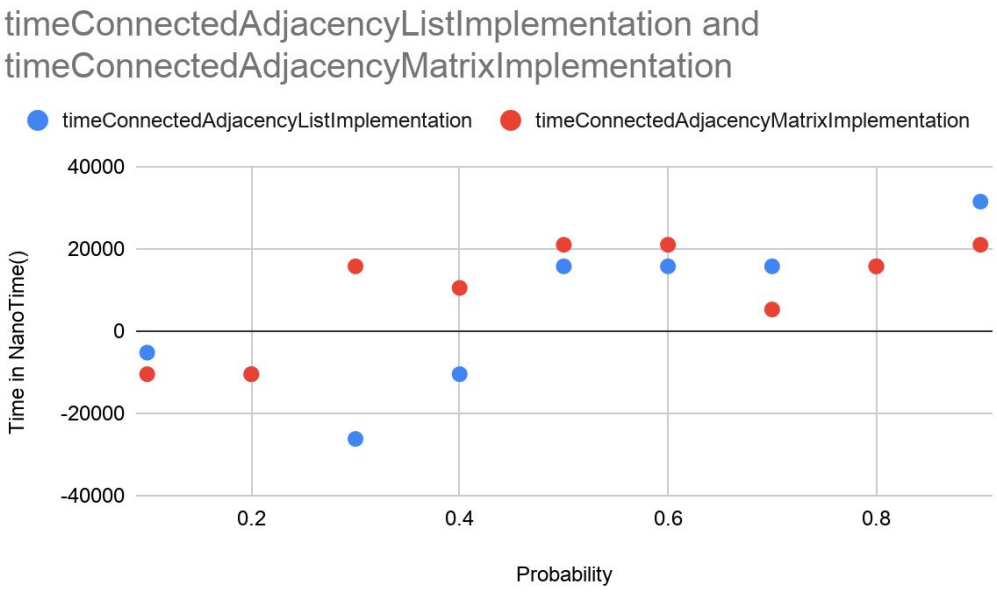

Figure 1: Performance of isConnected algorithm for adjacency list and adjacency matrix implementation

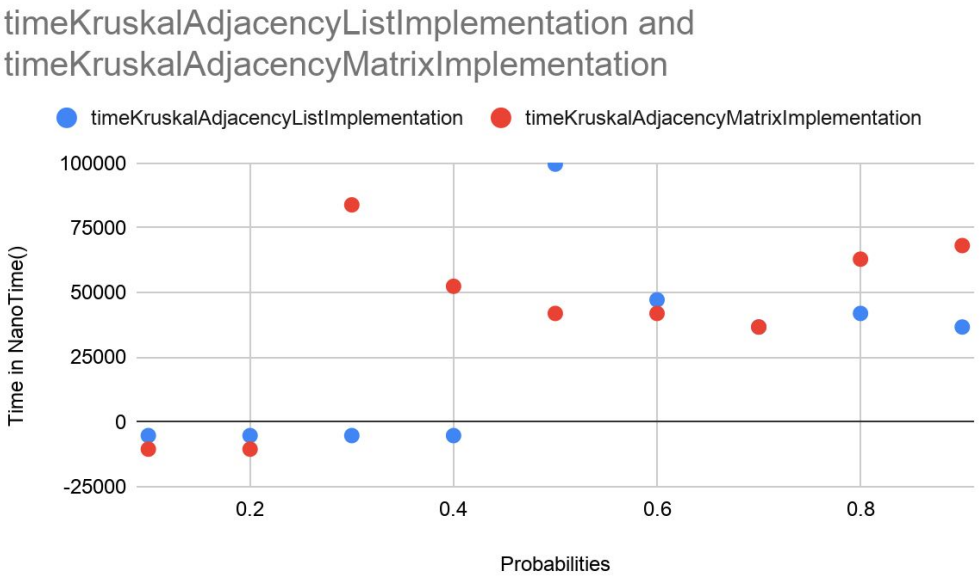

Figure 2: Performance of Kruskal's algorithm for adjacency list and adjacency matrix implementation

From Figure 1 and 2 for low probabilities, the time recorded is negative because the graph is not connected.

When graphs are finally connected, for Figure 1, the time taken tends to flush around a horizontal linear curve for both adjacency list and adjacency matrix implementation. Neither values for both graphs are consistently higher or lower than the other.

For Figure 2, when graphs are finally connected, the time taken to perform Kruskal's algorithm has a bit of a flat U-shape for the adjacency matrix implementation. The shape for the graph of the adjacency list implementation is half of the shape of a flat U.

The time taken for Kruskal's algorithm for adjacency list implementation is higher than adjacency matrix implementation until at probability value 0.8 then it becomes lower than that of the adjacency matrix implementation.

Reasons why this shape manifests is not clear at the moment looking at the implementation. Expected behavior was an increasing time for increasing edges which increases due to increasing probability but that was not the case here.

E. References

Lysecky, R. V., & Lysecky, F. V. (2013). *Data Structures and Algorithms*. Los Gatos, CA. Zyante Inc.

Oracle Corporation. (2018, October 6). Java API String. Retrieved from https://docs.oracle.com/javase/7/docs/api/java/lang/String.html.

Oracle Corporation. (2018, October 6). HashSet. Retrieved from https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html.

Oracle Corporation. (2019, September 11). HashMap (Java Platform SE 8). Retrieved from https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html.

Oracle Corporation. (2019, September 11). ArrayList (Java Platform SE 8). Retrieved from https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html.