**Algorithms & Data Structures: Final Project**

**Graph (Finding shortest path between two stops):**

For this part of the program, I chose to use the Dijkstra algorithm as it only looks for the shortest path from one given source node to all others, which removes a lot of unnecessary calculations that an algorithm like Floyd-Warshall would have to run. This is especially useful with the input files given as there is an incredibly large number of connections in the stop\_times.txt file, which has around 1.7 million entries.

For each of the node objects in my graph, there was an integer value labelling their stopID and a double indicating the cost of traversing to that node from the source. They also had a Linked List storing the path of nodes from the source node to them, which allowed for quick and easy access to the actual path required to get from the source node to a destination node. Each node also had a Hash Map storing the stops that are adjacent to it, which gives quick access for the nodes to traverse when looking for the shortest paths.

**TST (Searching for a stop by its name or the first few characters):**

As required, I used a Ternary Search Tree for this part of the program, the functionality of which allowed the user to search for bus stops by giving the full name or the first few characters. This would return a list of stops that the given input matched.

For the TST I gave all the information of the stop in for each time I put the stops in, and while this ended up in requiring more space for the TST, it made it much easier to retrieve the full information of each stop that the user queried. Any stop names that started with WB, SB, EB, NB or FLAGSTOP had that first word moved to the end. This made the searches more meaningful and easier for the user to give, only requiring for them to give the stop name.

**TST (Searching for a stop by its arrival time):**

In the same manner as the other search query, I used a Ternary Search Tree as I saw it was the most effective way to access the data as given by the user. This did however pose issues initially, as depending on how the information was stored into the TST, there was the possibility of the program running out of memory. This could have been avoided, if I had placed the strings into the TST more effectively.

The values placed into the TST followed the specifications, ignoring any invalid times (times that are below 0 in any unit, hh:mm:ss, or above the maximum time allowed, 23:59:59). This method of checking if any input times were valid was also used to check if the user’s input was a valid time to be checked off of later.

When printing the results of the query, the arrival time of each value was removed to make the output more sensible, not repeating the arrival time for every stop as that would already be known.