**Algorithms Final Project – Design Document**

**Leah O’Connor 20333782**

**Part 1- Shortest Paths and cost search**

This section requires allowing the user to input two different stop numbers and returns the shortest path along with the associated cost. I considered three different shortest path algorithms for this execution, Bellman-Ford, Floyd-Warshall and Dijkstra, but ultimately settled on using the Dijkstra algorithm for the following reasons.

The Bellman Ford algorithm has a relatively high time complexity of O(V\*E), where V = number of vertices and E = number of edges, which ruled out this option. The time complexity of the Dijkstra algorithm is lower, O (V^2), or O(V+E log V) with min priority queue.

The A\* algorithm involves finding a suitable heuristic, which proved difficult, and therefore I settled on using the Dijkstra algorithm as it was superior in terms of space complexity, and works best as an ‘uninformed algorithm’, i.e. when you cannot estimate the distance from source node to target node in the graph.

I utilised a HashMap to store the key value pairs of stop IDs and their indexes, as it is a data structure which allows easy operations using the key index, and although they are less space efficient than arrays an array would not have been a suitable data structure to use for the nature of the data, and storing in the graph nodes led to an index out of bounds error.

**Part 2- Bus Stop Search by name**

This section requires allowing the user to input a bus stop’s full name or the first few characters and return all the stop information that matches the search criteria. This part uses a ternary search tree(TST) to search for the relevant information. My program reads in the stop name with a scanner, and utilises it as a key to get an ArrayList<String> with the bus stop’s data. The relevant information is put in nodes of the TST using the put() method. The worst-case time complexity of insertion into a TST is O(N), while the average-case time complexity is O(log N).

The stop name must first be modified, to provide meaningful search functionality. As per the requirements if the stop name contained any of the keywords flagstop, wb, nb, sb, eb at the beginning of them, they would be moved to the end. I achieved this by creating a temporary string with the keyword, and then creating a new string variable and adding characters starting after the length of the keyword, and in turn adding the temporary keyword variable at the end.

The program uses the relevant TST get() method to traverse the TST and obtain a list of strings with the stop information for all the stops. The program then uses the inputted string to compare against this data, and if the characters do not match, that stop’s data is removed from the list, leaving only those which match the search criteria. Traversal in a TST is similar to that of insertion, with a worst-case time complexity of O(N) and average-case of O(log N).

If the list of stop details that match that of the input is not null, those details are outputted to the console after their relevant prefixes(ID, stop name etc.)

**Part 3- Bus Stop Search by arrival time**

This section requires allowing the user to input a specific arrival time in the form hours:mins:seconds (hh:mm:ss), and return all the stop information that matches the search criteria.

The program begins by reading in the different stop information from the input file, splitting the data using indexes and putting into its relevant ArrayList, for ease of printing the data in the later stages.

The specifications of the project require a bit of error handling, to make sure the times in the file are valid, and remove any beginning with 27/28 as these are clearly invalid.

In order to sort the data to be printed out I transferred the time IDs to a new array of integers. I considered using both quicksort and merge sort and settled on quick sort for the following reasons: firstly, the quicksort algorithm doesn’t require any additional space for sorting elements, it is also has the fastest average-case time complexity of O(n log n), and its worst-case time complexity being O(n^2), the quicksort algorithm as also the best suited algorithm for large data sets, which is what this program is working with.