Design Document

Group Members

David Yau - 19333385

Garbhan McCormack - 19334294

Tom Watson - 19335202

Oscar Whelan – 17331354

Part 1 – Shortest Path Between Two Stops

Data Structure: Adjacency List and Minimum Priority Queue (Hash-Map)

Algorithm: Dijkstra's Algorithm

Worst-Case Time Complexity: $O(|E| + |V|log|V|)$

- Where: E = number of edges in graph, V = number of vertices (stops) in graph

Worst-Case Space Complexity: $O(|E| + |V|Elog|V|)$

Notes:

Dijkstra's Algorithm can be implemented by use of an adjacency matrix or an adjacency list. I have chosen to use the adjacency list method as this has a better time and space complexity. The adjacency matrix implementation of Dijkstra's Algorithm has a Worst-Case Time Complexity is: $O(N^2)$ and the Worst-Case Space Complexity is: $O(N^2)$. By using the adjacency list method, it allows for the shortest path algorithm to run more efficiently, especially when there are more stops in the system. However, if there are less stops in the system, the adjacency matrix method would be a better implantation of the system. I used the Hash Map data structure in order to store the key-value pairs, where the key is the stop that the current stop connects to and the value represents the edge weight or cost associated with the edge. This is used within the context of a minimum priority queue. An ArrayList is used in order to store the stops from stops.txt. This is used to output the stop Name and Stop ID of each of the stops that are contained in the shortest path between the two user inputted stops. In order to locate a specified stop, binary search is used with another ArrayList of sorted Stop IDs used as a reference.

Part 2 – Bus Stop Search

Data Structure: Ternary Search Tree (TST)

Algorithm: Insert and Search Algorithm in TST

Worst-Case Time Complexity:

- Insert: O(N)
- Search: O(N)
    - Where N is the height of the tree

Worst-Case Space Complexity:

Proportional to the length of the string stored in the tree.

Notes:

I have used a recursive implementation of the TST in order to traverse and insert into the tree. Each node of the TST is initialised in the TSTNode class with char data. An ArrayList is used to store the names of the stops that are found to contain the user input term. I also used a sorted ArrayList (by brute force) to store all of the stops listed in stops.txt and their corresponding data. This was used in order to output the trip IDs and stop IDs of the stops that have been found by the search. Binary Search is used to search for the index of the stop that is needed, with a corresponding sorted ArrayList of Stop IDs being the reference.

Part 3 - Arrival Time Search

Data Structure: ArrayList

Algorithm: Merge Sort using the Collections.sort() method

Worst-Case Space complexity: O(N)

Worst-Case Time Complexity: O(NlogN)

Notes:

I used an ArrayList to store all the lines in stop_times.txt that have the arrival time input by the users. In the stop_times.txt file the arrival times that started with a single digit (for example 5:00:00), had a whitespace in front of the comma. I used the StringBuilder class in order to delete this whitespace from the string and form a new string with the correct format. In terms of the sorting algorithm, the Collections.sort() method is used (Merge Sort). This is used in order to take advantage of the better time complexity of Merge Sort in comparison to other brute force algorithms like selection and insertion sort each of which have a time complexity of O(N^2). I used another ArrayList in order to store the stops and their data. This was used in order to output the Stop Name and Stop ID. A corresponding ArrayList of all of the Stop IDs is used in order to sort the Stops ArrayList using a brute force method. Binary Search is used in order to locate the index of the stop that is required by the system, to locate the specified stop.