Design Document

Student: Jonathan Kelly, Student No: 20335359

Project Description:

- Implementation of a bus management system based on Vancouver bus system data

4 Main Functionalities of Project:

- Finding shortest paths between 2 bus stops (as input by the user), returning the list of stops en route as well as the associated “cost”.
- Searching for a bus stop by full name or by the first few characters in the name, using a ternary search tree (TST), returning the full stop information for each stop matching the search criteria (which can be zero, one or more stops)
- Searching for all trips with a given arrival time, returning full details of all trips matching the criteria (zero, one or more), sorted by trip id
- Provide front interface enabling selection between the above features or an option to exit the programme, and enabling required user input. It does not matter if this is command-line or graphical UI, as long as functionality/error checking is provided.

Purpose of Design Document:

A document explaining the design decisions (a choice of data structures and algorithms used to implement each of the 3 main features), justifying them based on specific space/time trade-offs between alternatives you have considered, considered in the context of the type and size of data you are dealing with.

Design Decisions for the 3 main functionalities of the Project:

Shortest Path:

- For this part of the project, I had to choose which algorithm to use to find the shortest path. There were many to choose from: A*, Floyd-Warshall, Dijkstra, Bellman-Ford etc.
- For a start, we must build a graph where a vertex represents a stop and an edge represents a connection between each stop. We must find the shortest path, so Depth First Search can’t be used as it only guarantees if a connection exists between two stops.

- We are given the cost of the edges and the values, which does not give us a heuristic function. Therefore algorithms like A* can't be used as they require one.
- Since there are most likely going to be cycles in this graph, and more importantly cycles in this graph will be non-negative, we won't use topological sort or Bellman-Ford
- This leaves us with Floyd-Warshall and Dijkstra. I chose Dijkstra as it has better space/time complexities compared to Floyd-Warshall
- Time Complexity of Dijkstra's Algorithm: O(E log V)
- Time Complexity of Floyd Warshall: O(V^$_3$)
- Space Complexity of Dijkstra's Algorithm: O(V)
- Space Complexity of Floyd Warshall: O(V^$_2$)

Searching for a Bus Stop:

I implemented the Ternary Search Tree as told in the project specification pdf,

- With regards to space/time trade-offs, the TST has the usual insertion, search and deletion methods which all have a worst case running time of O(n).
- After searching for the information, I stored the results in a Hash Table (hash map), which I felt was a better option than a Linked List or Array Data structure as it allows fast access to each element. I also wanted the results to be stored as key-value pairs.
- Considering there could be a large amount of data found in our search, we want to use a data structure for which the cost of each lookup would be independent of the number of elements stored (in the data structure), which is why I felt the hash table would be suitable.

Searching for Matching Arrival Times:

- I decided the easiest way to implement this system would be to use array lists to store the search results. I considered binary search as it would be a quick method of obtaining the search results (time complexity of Theta(logn)), however I was unable to implement it.
- I used one array list to read in the file stop_times.txt, and then using another array list to store the search results.To search by value in an array list has time complexity O(N), and to get a value by index has time complexity O(1).
- Another reason why I chose the ArrayList was because we don't know how big our search results are going to be. The fact that ArrayLists are resizable makes them an appropriate choice.