Algorithms and Data Structures

Final Project Design Document

Béibhinn Nic Ruairí
20333963

Overview:
Using given input files, my system provides a user with the ability to search for the shortest path between two bus stops, search for a bus stop using its name or prefix, and search for trips using an arrival time. I used the following data structures and algorithms to do this.

Graph Class
I imported the library algs4.jar from https://algs4.cs.princeton.edu/code/. I used the EdgeWeightedDigraph class from this to store the bus network in. I chose this class because it supports the specification that edges are directed and weighted.

Initially, I wanted to make the stop_ids the vertices in the graph. However, the stop_ids seem to be randomised numbers, with the biggest number being 12375 whereas there are only around 8000 stops. This would cause an array index out of bounds issue. To combat this, I added all of the stop_ids to an arraylist of integers and used Collections.sort() to put them in ascending order. This sorting algorithm has a guaranteed performance of O(n*log(n)).

I made the directed graph the size of this arraylist (as that contains all stops). To add connections and transfers between stops as directed, weighted edges in the digraph, I read the stop_id of the bus stops and used Collections.BinarySearch to find their index within the arraylist and then added the edge between the indexes in the digraph. I chose to use Collections.Binary Search as it has a time complexity of O(logn) and a space complexity of O(1) and searching algorithms don't perform better than this.

Shortest Path Algorithm
I chose to use Dijkstra's shortest path algorithm because it has a time complexity of O(**E**log(**V**)) (where **E** is the number of edges and **V** is the number of vertices in the graph). I considered using Bellman Ford but none of the edge weights are negative and this algorithm is slower than Dijkstras. I rejected Floyd Warshall as it has a large time complexity and the A* shortest path algorithm as I found it easier to implement Dijkstra than trying to calculate a reliable heuristic.

TST

I implemented a ternary search trie to complete the second part of the project - searching for a bus stop using its name or the first few letters of it. Insertion into a TST has a worst case time complexity of O(n). I used the bus stop's name as the key and stored that stop's corresponding index in the arraylist of bus stop ids. However, I reformatted the bus stop's name before I put it in the TST. I used a string array to store the keywords "FLAGSTOP", "WB", "NB", "SB" and "EB", and checked the first word in the bus stop name against this array, moving it to the end of the street name if found. I also moved the stop_id and stop_code to the end of the line. By doing this, I could use the TST class method keysWithPrefix to search for bus stops as the name of the stop would be first.

Searching By Time Algorithm

For the third main feature, I added all the trips with valid times (between 00:00:00 and 23:59:59) to an arraylist of strings. I sorted this by trip ID using Collections.Sort(). When a user enters a time, the program checks that the input is also a valid time and then checks through the arraylist of strings, checking the inputted time against the stored arrival times and printing out all info on a stop if it matches.

User Interface

I used a while loop that runs if a boolean flag isOver is false. It is set to true if a user enters "exit". I printed out a basic summary to the user, explaining the system and asking them to enter 1, 2 or 3 to choose one of the features. I used a switch statement to run each feature depending on their choice, with a default case if the user inputted anything other than that.