Data Structures and Algorithms Final Project Design Document

This project consists of three main parts: the first part was to find the shortest path between two bus stops, the second part was to retrieve all information about a bus that stop by searching the name of the bus stop or the first few letters of it’s name and, the third part was to search for all trips with a certain arrival time ,which returns the full details on any trips matching the criteria.

I decided that the first thing I needed to do was to read in the input files. I began by researching different ways of doing this as I knew one way to do it but wanted the method which best suits my project. I decided to use the BufferedReader. This is because it is much more efficient than other methods when reading in files containing large strings. Buffered reader is, in fact, the fastest way to read strings infrom a file. Buffers are very efficient at reading arrays, characters and lines, whereas the FileReader class, which is another class which aids reading strings from an input file, makes many assumptions about the data which can be problematic. Using the Bufferedreader also meant I could use the readLine method which allowed me to, very easily, skip the first line of the file when reading it in. This was extremely useful as the first line in each input file ccontained the format of the data in the file, not any of the input data. The readline method also allowed me to loop through each line of the file very efficiently.

There are many algorithms for finding the shortest path from a source vertex to a destination vertex. An obvious choice in the case of this project would’ve been to use a single source, single destination algorithm such as A \*. But using Dijkstra’s algorithm, with a binary heap, gives the best runtime of all shortest path algorithms. Dijkstra gives you the shortest path from a source vertex to all other vertices, so all you need to do is run Dijkstra’s algorithm and then get the distance from the source to a particular vertex (destination vertex).

To create the weighted directed graph containing the vertices and edges, I needed a way of storing the data associated with each stop id in a data structure to then use to calculate the edges between specific vertices. I struggled for quite a while with this, I started off by trying to use a 2d array and then an array list, but with no luck. I then came across a data structure called a HashMap. Using HashMaps allowed me to store a stop and its index in a way that they are associated with each other. I could then use this to get the vertices and edges for the graph. The HashMap's class contains many useful methods. HashMaps allow you to access elements very quickly due to hashing, although, when using HashMaps, you run the risk of two distinct keys generating the same hashcode. Hashmaps have excellent time complexity with a runtime of theta(1). This is constant runtime and also tops both arrays and linked lists.

Parts two and three of the project were to return all information relating to a bus stop and an arrival time. I was unsure about how to store all the information in each line of the file and associate it with a stop id or an arrival time which would be the key in the TST. Again, I tried using 2d arrays where the first element in each column would be the stop id/ arrival time and then each successive entry in the row would be the information to be returned. I considered using linked lists, having a linked list of each stop id or arrival time and then a list attached to this containing all the information to return. I had no luck with any of these attempts. I then realized that I could add each line of the input file as the “value” associated with its stop id/arrival time.

TST was used to store the data associated with the stop ids and the arrival times. TSTs are the most efficient symbol table in terms of runtime, with a runtime of theta(logN).