Design Document – Algorithms and Data Structures II Final Project

Aritro De (20331942)

**Part 1 – Shortest Path Algorithm:**

First I started by loading all the files (stops.txt, stopTimes.txt & transfers.txt) into ArrayLists so that I could easily access them throughout the rest of the project. I created a class for each different text file that was available so that I could create them into objects with the variables attributed to them being all the different columns included in the files. I did this because I find it easier to search for objects and then use the “object.variable” way of calling the attributes of all the files.

I decided to use the Dijkstra code from the Princeton website, the same as the Algorithms book written by Robert Sedgewick and Kevin Wayne. The reason I decided to use Dijkstra is because of the fact that it performs very well with calculating the shortest path within a weighted graph.

I added error handling to the check if the entered stop is within the list of stops in the files (all stops were within 0 and 12478) and also another one to check if there is a valid path between the two stops entered by the user.

**Part 2 – Ternary Search Tree:**

For the ternary search tree, I created an object class for the nodes and then implemented them into a ternary search tree class. In the ternary search tree class there is an insert, search and null methods. I implemented the search method by making it search all the way down to the last character in the input from the user and then it will return all of the possible names that could be the finished from the input.

For example, if you type in “hastings” it will return all of the stops names that start with that word EXCLUDING THE WB, NB, EB & SB from the name. The input is also not case sensitive which I did by making everything upper case.

**Part 3 – Search By Time:**

To search by time, I made a simple loop that checks the inputted string from the user against the time of arrival for all the stopTimes objects time of arrival. The if the input matches the arrival time of a trip, I made it load out all the other attributes of the object including the departure time and stop ID.

For error checking, I checked that the time entered was a valid time in the form of HH:MM:SS and if it was anything but this or the word “quit” the console would ask for another input.

**Part 4 – UI:**

For the user interface, I made a different main function in all of the classes. Each of these consoles was easy to use and understand while also being clear and informative. When displaying data, I made sure to have dividers between different sets while also making sure that every attribute being shown was of use and easily readable.

On top of that, I made an overall main class where I call all of the other ones, just for easy use.