Danielle Buggle

DSA Final Project Design Document

Overview

To create this project I decided to implement the following data structures and algorithms to complete the functionality of the three main parts:

Graph Class

I used a graph class to store the entire bus network of Vancouver. I imported the algs4.jar from https://algs4.cs.princeton.edu/code/. This way I could use the EdgeWeightedDigraph class to store the bus network since it needed to be directed and edge-weighted as per the specification. Each node represents a given bus stop using its Bus Stop ID. I used the DirectedEdge class to represent each of the routes and transfers within the bus network. I used this class as it is necessary create the EdgeWeightedDigraph of the entire network.

However, there was a problem when creating the edges to store in the graph. The bus stops do not begin at 0 and increase by 1 until 8757, they appear to be random instead. My approach was to store each Bus Stop ID in an ArrayList of Integers, use Collections.sort to sort the array in ascending order so that I could use Binary Search later on. Binary Search has a time complexity of O(logN) and the Iterative Binary Search has a space complexity O(1). I then use the index value of the Bus Stop ID to create the edges and the graph. I chose not to use ArrayList.indexOf() to obtain the index value to create the graph as it runs in O(N) time complexity which is worse than Binary Search.

Shortest Path Algorithm

The first part of this project we were asked to find the shortest path from one bus stop to another as inputted by the user, inputted in terms of the Bus Stop ID. To do this I implemented Dijkstra's Shortest Path algorithm. I didn't use Floyd-Warshall as it has a longer run time than Dijkstra and it doesn't record the path taken. Dijkstra records the path taken, I used the ArrayList of Integers, identified which index was returned from the algorithm to get the corresponding Bus Stop ID from the list and print out each stop visited.

Ternary Search Tree

I used to the Ternary Search Tree class from the algs4.jar to implement part two of the required functionality. This TST stored the names of the bus stops and all of the information associated with the stop. First of all, I created a function to change the name of the bus stop and remove any unnecessary keywords at the beginning of the name such as WB, EB etc and move them to the end of

the string. To do this I created a shift left by N function which allowed a string array to be shifted to the left by the number passed in as a parameter. This function was also used to shift the name of the bus stop to the beginning of the information provided by the text file so that I could use the TST to search by name of the stop. This TST class has a keyWithPrefix function which gets each value in the tree which starts with the prefix inputted by the user which I used to print out the stops to the user.

Reading Stops and Transfers

I created a method that would allow me to read in the stops.txt file and store the necessary data. I had a loop that would read each line in the file, store each bus stop ID into an ArrayList of Integers. I editted the bus stop names and order of the information so that the bus stop name would be first and then stored to then be put into the ternary search tree I had created. I then sorted the Array List so that I could use Binary Search as it's more efficient than other forms of searching when the array is not sorted.

I created a method that would allow me to read in the transfers.txt file and store the data that was necessary. I split each line by a comma into a String array and check the index of that array where the transfer type would be. I search for each bus stop ID in my ArrayList of Integers and create an edge between these two stops based on the index value they were stored at in the array. If the transfer type was 0 I would give the edge a cost of 2 otherwise I would get the cost by dividing the min_transfer_time by 100.

I created another method that would allow me to read in the stop_times.txt file and store the necessary data. I read in each line and add each valid trip was added to a ArrayList of Strings for use in printing out the trip when user searches by arrival time. I created an edge between each corresponding trip that had the same trip ID with a cost of 1.

Searching for Arrival Times

I used an ArrayList of Strings to implement part 3 of the required functionality. When reading the data in I stored all trips that were valid i.e. with times between 00:00:00 and 23:59:59. I then created a function that allowed the user to input the arrival time they are searching for in the correct format with appropriate error handling. As the array was sorted after the data was read in, I was able to create a function that would loop through the array and print out each trip with the users searched for arrival time sorted in order of trip ID.

User Interface

For the user interface I used the console built into Intellij. I asked the user to input a number related to which function of the entire program they would like to use. Then each function would loop and perform the appropriate functionality alongside appropriate error handling i.e. error messages to the user when they input something in the wrong format such as the arrival time in words instead of h:mm:ss. Each function would loop until the user entered the correct word to exit the current section. The entire program would loop as well until the user entered “exit”, then the program would terminate. The entire program and each section contains appropriate error handling and messages to the user to indicate what they may have entered wrong, if anything.