Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU22012
Algorithms and Data Structures II

Design Document

Vancouver Public Transport System

Emma Ruth Daly - 18319755
Jack Kennedy - 18328691
David Olowookere - 19335061
Cathal Leahy - 19334536

May 3, 2021

Introduction

This is a system designed and developed based on the Vancouver Public Transport System. From the data provided to use we had four main tasks:

1. Find the shortest paths between 2 bus stops (as input by the user),and return the list of stops en route as well as the associated “cost”.
2. Search for a bus stop by full name or by the first few characters in the name, using a ternary search tree (TST), return the full stop information for each stop matching the search criteria
3. Search for all trips with a given arrival time, return full details of all trips that match the search criteria, sorted by trip id
4. Provide front interface enabling selection between the above features or an option to exit the programme, and enabling required user input

1 Find the shortest path

The two stops to get the shortest path to are passed through a constructor and the files are read. Each line of the file is scanned and then each piece of data on each line is scanned using a different scanner.

A ”map” is created using the info from the files and an adjacency matrix. The shortest path between the departureStop and the arrivalStop is found using Dijkstra’s algorithm.

Dijkstra was used to find the shortest path between two bus stops because it can be used to find the shortest path from a bus stop to every other reachable bus stop from the starting bus stop.

Dijkstra’s algorithm also works well with weighted and directed graphs and does not require any prior knowledge of the graph to calculate the shortest paths.

The route that the bus would take from the departureStop to the arrivalStop is then printed which each stop along the way. The associated ”cost” is returned as a double.

The algorithm does run rather slowly so perhaps there is more efficient data structure that could be implemented.

2 Search for bus stop by name

Firstly, a Ternary Search Tree is initialised and created using a three step process.

First: the stops file is read into an array line by line.

Second: each line is split using the String.split() method and converted into a StopInfo object, which contains a variable for each element in the line.

Third and finally, the array of StopInfo objects is converted into nodes and added to the tree one by one, with the object itself as the value and the stop_name variable as the key.

After that, the searchStops() function works as follows. First, it searches for a path that matches the key, same as any ternary search tree. However, if it reaches the end of the key, instead of only returning the data on that node, it goes down every path branching off from that node, and every time it finds a node with a StopInfo object it adds its info to the list, since we know that the only StopInfo objects it will encounter are ones where the start of the stop names match the key. After the trawl, it returns the whole String, which is neatly formatted and ready for printing.

3 Search for trip by arrival time

The search by Arrival Time was implemented in a fairly simplistic way using an improved version of Hashmapping with the class LinkedMap Abstract Data Structure.
The LinkedMap ADS uses the idea of hashmapping and allows for objects in this case BusInfo objects to be added to the arrival time as the hashkey in a Linked List fashion.

Firstly, the userInputed ArrivalTime was tested and prints 'Invalid' to console if not accepted. Then once each supplied text file is tested we begin the functionability.

Each line of stops_times.txt was parsed into an array to contain the information of each line using split(","). tripID was also parsed into an integer from lines[0], its position within lines arrival times were parsed from stop_times.txt into an arrivalTimesFormattedArray using split(":") so 12:00:00 in the array was 120000 for arrivalTimes. Parsed in with the length " 5:21:10" a String Builder was used to remove the space character. For the performance StringBuilder was used to over String[].

Once an arrival time was in the array it was then checked to be valid, if the arrival time was indeed valid a BusInfo object was created using theinformation of the line, i.e line[size of information] and added to the LinkedMap known as busInfo.

Then the custom comparator for tripID was made to allow for comparison of BusInfo objects to be compared if the busInfo LinkedMap did include the userInputed arrival time as the key that implied there was buses arriving at the user searched time.

A copy of the linked list of BusInfo objects was then copied to another array called tripIDObjectArraySorted and sorted using the custom comparator and Arrays.sort which uses a selection of mergesort quicksort and insertionSort for simplicity. The custom Comparator was set up to compare objects and sort by TripID so the BusInfos were sorted by TripID.

Now that we had an array of sorted TripIDs BusInfo objects the last thing to do was add the info from stops.txt for a comprehensive look at all stop information. Using stopID the text from stops.txt was parsed into each respected BusInfo object and once all information was added to all BusInfo objects at the arrival time the loop terminates and uses the BusInfo.outputInfo() method to return all bus infos at the searched for arrival time.

4 User Interface

The JFrame class is used to display a graphical user interface. This class creates a pop-up window with options to exit, enlarge, or minimise.

A JPanel is used to contain the elements of the interface. Each feature has 2 JLabels, one that displays text explaining which feature the elements under it pertain to, and one explaining what input to put in the text fields and showing any errors arising from incorrect inputs.

Each feature has a number of inputs that are entered into a set of text fields implemented with JTextField. For each feature there is a JButtons that when pressed takes in the inputs in the associated text fields, checks for errors, calls the feature, and creates a new JPanel with the output from the feature and a JButton that brings the user back to the previous JPanel.