CSU22012 FINAL PROJECT DESIGN DOCUMENT

STUDENT DETAILS

Student name: Eimear Ryan

Student ID: 20332642

OVERVIEW

The aim of this project was to use the algorithms and data structures covered over the course of the module CSU22012 to implement a system that enables a user to access data about the Vancouver public transport system. The system provides functionality to

i) Find the shortest path between 2 bus stops, returning the list of stops en route as well as the associated “cost”.
ii) Search for a bus stop by full name or by the first few characters in the name, using a ternary search tree, returning the full stop information for each stop matching the search criteria.
iii) Search for all trips with a given arrival time, returning full details of all trips matching the criteria, sorted by trip id.

My code was run on Eclipse version 2020-06 (4.16.0) and it was decided for simplicity and efficiency to run a terminal based user interface that could handle erroneous input from the user and provide clear instructions on how to navigate through the bus system.

To begin, the console outputs the main menu where the user is asked to select which of the three functions they would like to run. From here, once a function is selected, they can either enter what is asked of them by the command line or enter ‘exit’ to go back to the main menu.

The system handles incorrect input from the user by letting them know in the console that their input is invalid and asking them again to enter something valid.

PART 1 DESCRIPTION AND ANALYSIS

(Note E = the number of edges in the graph, V = the number of vertices in the graph)

Data structures used:

An adjacency list was used to make the graph with the number of vertices being the total number of stops in the “stops.txt” file. Each vertex then has its own array list of edges containing the source vertex, destination vertex and cost. The space complexity of an adjacency list is O(V + E) I considered using an adjacency matrix to store the graph, however, the space complexity for this would be worse, O(V^2).

I decided to use a hash map to store all the bus stops. Due to the fact the stop ids could not be indexed from 0, I made the key the stop number and the value the index number. This meant that the index values of the bus stops were used when adding edges and accessing vertices in the graph. This was also an efficient data structure to use as it has a worst-case time and space complexity of O(n). However, hash maps on average achieve O(1) run time for put and get operations.

Algorithms used:

Dijkstra's shortest path algorithm was implemented to find the shortest paths between two bus stops inputted by the user. This was a suitable algorithm to use as there were no negative weights and there was a possibility of cycles. The worst case run time for this algorithm is O(ElogV) and the space complexity is O(V). This is a better run time than the alternative algorithm that could have been used to find the shortest path, the Bellman Ford algorithm which runs in O(EV) time.

PART 2 DESCRIPTION AND ANALYSIS

Data structure used:

An array is used to take each line of the "stops.txt" file and split it into the stop name and stop information. The stop name had to be made "meaningful" by taking the key word and flagstop and putting them to the end of the stop name string.

A Ternary Search Tree (TST) was then used to search for stop information, putting the stop name in as the key and stop information in as the value.

The average time complexity for searching, inserting, and deleting from a TST is O(logN) and O(N) in the worst case. The space efficiency of a TST makes it the superior data structure to use implement the functionality required of the assignment.

PART 3 DESCRIPTION AND ANALYSIS

Data structures used:

An array list was used to store all the lines from the file "stop_times.txt" that contained valid times and then another array list was used to store all the arrival times from the first array list that matched the user input. An array list has O(N) time complexity for arbitrary indices of add/remove, but O(1) for the operation at the end of the list, which is all we need in this program, therefor it is a very efficient data structure to use.

A 2D Array is used to sort the trip ids. The array list of matching times can't access each individual stop id by an index value and therefore, the 2d array is necessary to access each trip id.

Algorithms used:

Bubble sort was used to sort the trip ids. This was chosen as it is simple to implement and appropriate because we are not dealing with a large amount of data to be swapped (only the array list containing matching times are to be swapped). Bubble sort has a worst case run time of O(N^2) and a space complexity of O(1).