CSU22012: Data Structures and Algorithms Project

Name: Noel Peter

Student ID: 20330157

Bus Management System

Interface

The program welcomes you to the system and asks the user to fish out which of the options they would like to choose from the following parts

The User is obliged to choose between option 1, 2, 3 or can exit the system.

The User entered choice is then navigated to either of the parts.

Part 1

Algorithm:

For this part I used Dijkstra’s algorithm, to find the shortest paths between any two given bus stops as input by the user via the command line while returning the list of stops en route as well as the associated “cost” and the total cost of the path.

Data Structure

I picked the graph data structure with the adjacency list representation instead of the adjacency matrix as the matrix would end up taking a large amount of space to store the weights in which more than 50% of which wouldn’t be used through the program anyway hence the adjacency lists are better for sparse graphs when you need to traverse all outgoing edges, they can do that in O (degree of the node).

Adjacency lists have a space complexity of n whereas adjacency matrices have a space complexity of n^2 and as N (the no. of bus stop in this case) is a large amount, N^2 would result in an even large amount in terms of space complexity. Although adjacency matrices have a time complexity of O (1) to find if two nodes are connected but adjacency lists take up to O(n), in this system the time would have to be sacrificed for better performance of the system.

In this I have used the arraylist with a list of nodes for the representation, but we can also use linked list for the nodes as well.

Interface

The User is asked to enter the name of the stop where they’d like to start the journey and later the name of the stop where they’d like to end it, then the program displays to wait for a moment while computing the shortest path between the two stops and displays the paths and the associated cost as well as the total cost at the end.

Part 2

Algorithm:

For this part I used TST to store the name of the stops and I set the bus number as value field. As the user enters the name of the bus stop TST is used to find the bus number and that is then used to find the remaining information about the bus stop from the stops.txt file and displays it to the user’s satisfaction.

Data Structure

Although as an alternative we could have used tries in place of the TST, but TST provides a data structure which helps majorly in saving space as it involves only three-pointers per node as compared to radix in standard tries. Further, ternary search trees can be used any time a hashtable would be used to store strings.

Interface

The User is asked to enter the full name of the stop of which they’d like to find the information for. The Program then uses TST to check for the stop and displays out the information regarding the stop.

Part 3

Algorithm:

For this part I used simple file search which reads a line and checks if the time is valid and if it does it puts it in a pair class with the trip ID and stops the bus makes while making the trip. And then at the end displays all the trips within the time the user had put in as an input.

The reason I went with file reader was for the only reason to save space rather than storing it in a data structure and then extracting all the information backs again just to get few of the many trips and its stops

Interface

When the user punches in their choice as 3 the system asks for a particular time, they wish to see all the busses for, and it only checks in 24-hour format if the user enter otherwise, they are sent back with, and error message stating their mistake and takes back an input again. When the user put in a time the system displays a temporarily wait message and then spits out the information following the user’s choice.