Bus Management System

Kajus Krisciunas - 20332808

1 Introduction

Application Requirement

- stops.txt – list of all bus stops in the system, cca 8,000 entries
- transfers.txt – list of possible transfers and transfer times between stops, cca 5,000 entries
- stop_times.txt – daily schedule containing the trip times of all routes on all stops, cca 1,7 million entries

It must do all of the following:

1. Finding shortest paths between 2 bus stops (as input by the user), returning the list of stops en route as well as the associated “cost”.
2. Searching for a bus stop by full name or by the first few characters in the name, using a ternary search tree (TST), returning the full stop information for each stop matching the search criteria (which can be zero, one or more stops)
3. Searching for all trips with a given arrival time, returning full details of all trips matching the criteria (zero, one or more), 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. It does not matter if this is command-line or graphical UI, as long as functionality/error checking is provided.

Design Decisions

Part 1:

Used the following .java files: Digraph,Dijkstra,Edge,Path,Node

The Node class was used to store every bus stop that occurred and the Edge class stored the path taken by the bus from one node to the other. The Edge class also stored the weight of the journey.

The Digraph class (directed graph) was used to store all of the information from the three given files. It would sort them into arrayLists,nodes and edges. I chose Arraylists

because they are very easy to use and don't require a fixed size like arrays do.

Time complexity for ArrayLists:

-Insert O(1)
-Search O(n)

I used the Dijkstra algorithm to find the shortest path between any two nodes. This to me was an obvious choice because it is such a prominent algorithm when it comes to finding the shortest path which we learned in the lectures.

Dijkstra has a time complexity of O(ElogV) and a space complexity of O(V)

Lastly the Path class was used to return the information gathered by Dijkstra. It returned both the total number of stops between the two stops that the user entered and also returned the information on each of those stops. This was then later outputted in the terminal

Part 2:

Used the following .java file: TST

The ternary search tree was used as was specified. It allowed the process of looking up the names of bus stops and returning information to be very fast and efficient. A ternary search tree is a type of trie (sometimes called a *prefix tree*) where nodes are arranged in a manner similar to a binary search tree, but with up to three children rather than the binary tree's limit of two. This was also chosen because it is more space efficient compared to standard prefix trees.

Time complexity for the ternary search tree:
Search - O(logn)
Insert - O(logn)

Space complexity for the ternary search tree:
Search - O(1)
Insert - O(1)

Part 3:

Used the following .java file: Time Search

This part required the user to enter a time in the format hh:mm:ss and all bus stops with an arrival time equal to that of the user input would be returned.

The algorithm that I wrote used Arraylists once again as they are super handy and efficient when dealing with unknown array sizes.

For the algorithm I put all of the stop times into an ArrayList and then compared the in the user inputted time with every entry in the ArrayList

Time complexity for ArrayLists:

-Insert O(1)
-Search O(n)

UI/Command-Line

Part 4:

For this part of the project I decided to go with a simple command-line interface for the user.
When the program is launched the user is met with a choice:

Shortest Path(A) Bus Stop Search(B) Arrival Time(C) Exit(Q)

To choose between either of the options you just type the letter in the brackets.
There is an error control system for all inputs. To exit the program the user simply has to type Q.