CSU22012

Final Project Design Document

Peter Gutstein: 20300420

## Part 1: Search for Shortest Route Between 2 Stops

I was unable to implement this part of the project fully. However, my plan was to use Dijkstra’s algorithm to find the shortest path between two nodes (bus stops) as it would be more efficient that the Floyd Warshall algorithm as it is likely a user wouldn’t be looking for all stops in one use of the program, but rather only a select few, so having Dijkstra run a few times at a worse case runtime of θ(E + V\*logV) rather than have Fllod Warshall run once at the beginning for a worst case runtime of θ(V^3), where E and V are the number of edges and vertices, respectively.

## Part 2: Search for Bus Stop by Names

In order to implement part 2 of this project, I made use of a Ternary Search Tree as it would be the easiest and most efficient way to search for strings and substrings in a large set of data as average searching time is O(logN) for a hit and the worst case search time being O(N) for no result found. I first loaded all the stops from stops.txt into an array of stops, which contained all the information needed to fully describe a stop. After that, I loaded all of the names of the stops into a TST with their keys and values being the name of the stop. Once the TST was constructed, if someone wanted to search for a stop, the program would search through the TST and return all strings beginning with the search term before printing them all out. If no stops were found, then the program would print an error message saying no stops matched the search term.

## Part 3: Search for Trip by Arrival Times

For searching for trips by arrival times, I considered using another TST like I did for part 2, but ultimately decided against it as I was only looking for strings that exactly matched the search term. Instead, I used a simple for loop that scanned through an array of Trip objects (which contained basic information on each trip from stop\_times.txt). On each iteration of the loop, if the arrival time of the Trip was the same as the searched for arrival time, the Trip would be added to an ArrayList of Trips. After going through the entire array, I went through the ArrayList, finding the smallest numbered trip ID. Once I found the smallest trip ID, I would print the associated trip and its information and then remove it from the ArrayList. This process would repeat until the ArrayList was empty, meaning all Trips that matched the search were printed. If no trips matched the search term, then an error message would be printed. Even though this process sounds convoluted and complicated, it has an average running time of O(N).