Introduction

The task given to us was to create a system in which a user could navigate the Vancouver bus system. In this assignment, there were three main components to the final Assignment:

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

I was also asked to make a user interface which was easy to understand and could give you a list of options and how to use the system. The user interface was mainly made up of *if, else if* statements, and it was relatively easy to implement.

**Shortest Path**

Through much deliberation and research, I decided that Dijkstra was the best algorithm to use, mainly due to its simple implementation and its relatively quick time complexity of O(N^2). I was considering using A\* for a while, but due to my limited time to complete the project and due to the complexity of the implementation of A\*, I decided to use Dijkstra, which I had already used in the previous assignment.

I also decided on ausing an adjacency matrix rather than a hash map, as mentioned above, due to time constraints I had to pick a fast and effective solution. Unlike a hash map, the adjacency matrix only takes up one class, making it far easier to understand and work with. Initially I started the adjacency matrix using only 2000 inputs, and then finally increasing it to encapsulate all possible inputs and outcomes

**Bus Stop search**

We were instructed to use a Ternary search tree for this question, and I would agree that this was by far the simplest solution to the problem. I used a mixture of hit or miss searc, whilst also focussing on character key matches, giving out the proper address’. I also got rid of the Compass directions of the bus routes, as in the instructions it said to focus on the physical address of the place and not the direction.

**Arrival Time**

This was an easy implementation and amalgamation of arrayLists and collection.sort which sorts the given times in ascending or descending order depending on what you would prefer.

CollectionSort is basically just a mergesort compiler. Although Quicksort would seem like the better option in this case, and is also probably quicker, Mergesort is more stable and more reliable, especially when dealing with huge chunks of information, such as seen in stop\_times.txt