**Final project**

**Design document**

**Name : Zehao Yu**

**Student ID : 19336864**

1. **Introduction**

**1.1. Overview**

The main purpose of this project is to implementation of a bus management system based on Vancouver bus system data. This bus management system needs to provide three functionalities :

1. Finding shortest paths between 2 bus stops.
2. Searching for a bus stop by full name or by the first few characters in the name.
3. Searching for all trips with a given arrival time.

Also, it contains a front interface to enable selection between three functionalities and enable required user input. The whole project is designed in java language.

**1.2. Reference**

Algorithms, 4th Edition

<https://algs4.cs.princeton.edu/44sp/EdgeWeightedDigraph.java.html>

<https://algs4.cs.princeton.edu/44sp/DirectedEdge.java.html>

<https://algs4.cs.princeton.edu/44sp/DijkstraSP.java.html>

<https://algs4.cs.princeton.edu/52trie/TST.java.html>

1. **System Design**
   1. **Finding shortest paths**

This functionality aims to find shortest paths between 2 bus stop and return the list of stops en route as well as the associated “cost”. Stops are listed in “stops.txt” and connections (edges) between them come from “stop\_times.txt” and “transfers.txt files”.

Since the costs of the edges in the “transfers.txt” are all positive, I chose Dijkstra algorithm to find the shortest path. Dijkstra algorithm can find the shortest path from one given bus stop to another given bus stop. This algorithm runs until all reachable nodes are visited, and saves the result. So, just running Dijkstra's algorithm once can be used again and again without re-running the algorithm unless the graph changes. Bellmanfort's algorithm is similar to Dijkstra's algorithm, but takes longer to run. And the Floyd-Warshall algorithm is not a single-source algorithm, it is not as efficient as Dijkstra's algorithm in this project. In this case, compared to other algorithms, Dijkstra's algorithm is concise and optimal. To implement Dijkstra's algorithm, I first created direct edges to connect bus stops and recorded it’s cost. Then, I created an empty edge weighted direct graph and put all the edges into the graph. After that, I used Dijkstra's algorithm to draw the shortest path graph. In the Dijkstra class, I wrote a function to list all the bus stops in the path between two given bus stops and return the list to the main class.

* 1. **Searching for a bus stop**

This functionality aims to search for a bus stop by full name or by the first few characters in the name and return the full information of these matching bus stops.

Since the Project Specification asked me to use ternary search tree (TST) in this functionality, I used the TST class I found from “Algorithms, 4th Edition” website as my skeleton code. Ternary search tree has the advantages of saving space and fast query. Ternary search tree has three child nodes. When searching, compare the current character, if the searched character is smaller, then jump to the left node. If the searched character is larger, then jump to the right node.

Then I add two functions in TST class to store the name of bus stops in an array list and the information about the bus stops in another array list. When I entered a bus stop name, it will compare the given bus stop name to the bus stop name list and return the information of the matching bus stop.

* 1. **Searching for all trips with a given arrival time**

This functionality aims to search for all trips with a given arrival time and return full details of all trips matching the criteria. I wrote two functions in the SearchingTrips class to implement this functionality.

The first function read the data from “stops\_times.txt” and stored bus stop name and arrival time in an array list. The other function simply read the data from “stops.txt” and stored all information about the bus stop in another array list. Then I wrote code to compare the given input to the arrival time list and print full information of all bus stops matching the criteria. This method of comparing data using an array list is very simple and easy to understand. But it also takes a lot of time to run and is not efficient. When I enter an arrival time, it takes a while for the program to print all the results.

* 1. **Front interface**

My Front interface contains a while loop and four if statement in the loop. The while loop makes sure that my programme won’t end until some specific condition happen. And the four if statements will enable selection between the above features or an option to exit the programme. Also, I wrote code for detecting input error within the four if statements.