**Title Page (KG)**

## **1 Introduction**

## **Application Requirements (DC, SD, SL)**

The objective of this project is to develop an interface using techniques learnt in class which will act as a bus information system for Vancouver public transport. For the purpose of this project, we have used three text files including stops.txt, transfers.txt and stop\_times.txt. Using these input files, we have implemented the following functionalities: 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”. 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), 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 and Finally Providing front interface enabling selection between the above features or an option to exit the programme, and enabling required user input.

## **Technical Approach (DC, SD, SL)**

*As a group we came together and discussed the project and decided what algorithms we wanted to use/ were the best fit for the assignment/*

* We planned out what I wanted to do for my implementation and how I wanted to do it.
* We examined the requirements carefully.
* Individually we further researched our separate parts of the project
* Denisa researched more about ternary search trees
* Sarah researched datasets to store large arrays and algorithms to sort the trips by ID
* Sean researched ways to store the data into a graph, and how to traverse this graph.
* We simplified it in different parts and worked each one out one by one
* Created a really simple version of the algorithm and over time I added more and more to it until it met all the requirements.

## **2 Design**

## **Logic Flow Diagram (KG)**

## **3 Implementation**

## **Tools, Libraries, Platforms (ALL)**

* Eclipse on MacOS
* Eclipse on Windows
* GitHub (repo to store and share code)
* Instagram group chat to group call and discuss the project.

## **User Interfaces (KG)**

## **Algorithms (DC, SD, SL)**

The following are the design decisions in which we discussed and made, relative to each of the three main features:

Part 1:

-Consisted of three .java files, containing 8 classes, (findShortestPath, Dijkstra, Graph, Path, Node, Edge, and transfersAndTimes)

-Getting from one stop to another, consists of several interlinked transfers, where each single transfer is referred to as 1 Stop. For a transfer to exist, it needed a fromStopID, a toStopID, a minTransferTime, and a transferType. Each transfer has this very information stored about it in the transfersAndTimes class.

-Each stop was represented by a node, and each ‘journey’ was represented by an edge. From stop 1817, to 1818, there would be two nodes (1817,1818) and one edge connecting them. Travelling from one to the other is called a path, and when the weight is factored in, it is a transfer.

-To create the graph, I used a graph class, which takes in a string array containing our three files: “Stops.txt”, “transfers.txt” and “stop\_times.txt”. Using these files, we sort them into arrayLists and add them to our graph by creating edges and nodes from the data.

-We used Dijkstra algorithm to traverse the graph and help to find the shortest path between two stops. I used Dijkstra because this was its intended purpose of creation, finding the shortest distance between nodes.

-The findShortestPath class creates a very simple was of finding the shortest path between two stops, to be implemented in the main. The two methods, (numberOfStops and stopsAlongTheWay) each return the number of stops between two entered stops, and the tickers and cost associated with these stops, respectively.

-Dijkstra algorithm has time complexity O(ELogV) and space complexity of O(V) where E= number of edges, and V=number of vertices.

Part 2:

* Ternary search tree was used.
* A ternary search tree is a type of prefix tree where nodes are arranged as a binary search tree.
* There are two classes, a class for the Node and a class for the TernarySearchTree.
* The node class has a constructer for the nodes in the tree.
* I have an insert function in the TernarySearchTree class which is used to insert the list of bus stops into the tree
* The average Time Complexity for the insert function is of O(log n) and the worst case is O(n)
* There is a search function in the TernarySearchTree class which is used to search for the word (bus stop name).
* The average Time Complexity for the search function is of O(log n) and the worst case is O(n)
* There is a function in TernarySearchTree class to traverse the tree to be able to find the searched word
* The average Time Complexity for the transverse function is of O(log n) and the worst case is O(n)
* This algorithm has a Space Complexity of O(1).

Part 3:

* This part entailed searching for all trips with a given arrival time, returning full details of all trips matching the criteria, sorted by trip ID.
* The data structure used in this part of the project includes the use of Array List. We choose to use an array list here as we are using an input file of over 1.7 million entries and array lists have a space complexity of O(N) .
* As well as that Array lists work well with Collections.sort() which I initially thought implemented quick sort but after research I found it implements Tim sort which is a hybrid stable sorting algorithm derived from merge sort and insertion sort.
* Collections.sort() also sorts elements presented in a specified list of collection in ascending order which is ideal as This part of the project the trips are needed to be sorted by ID.
* We found it ideal to use Collections.sort() as it is fast and has a guaranteed run time of O( N log N ), it is stable meaning the order of stops is preserved.

## **Runtime Environment (KG)**

## **Conclusion**

We split the group work evenly amongst the team. We met frequently to discuss and review different parts of the project. The following is a description of each of the parts each member played in the group :

Sean Langan : Implemented part 1 & worked on the document .

Denisa Costinas: Implemented part 2 & worked on the document.

Sarah Dolan : Implemented part 3 & worked on the document.

Keira Gatt : Implemented part 4 & worked on the document.