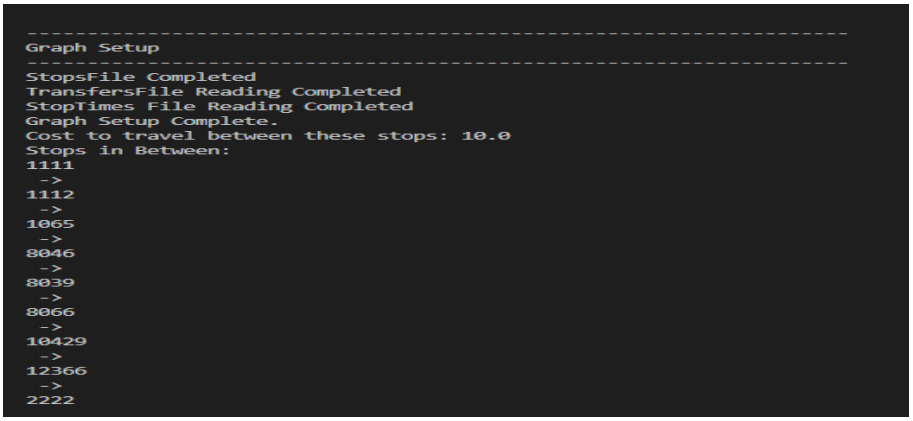
CSU22012: Data Structures and Algorithms Project:

DESIGN DOCUMENT:

Name: Karthik Venkatesh Nagaraj
TCD ID: 20330418

- **Part 1:**

  For the first part of this assignment, the way I went through with completing it, is that I created classes **stopsEdges**, **stopsInfo** and **tripInfo** which hold the information from the transfers.txt, stops.txt and stops_times.txt files respectively.

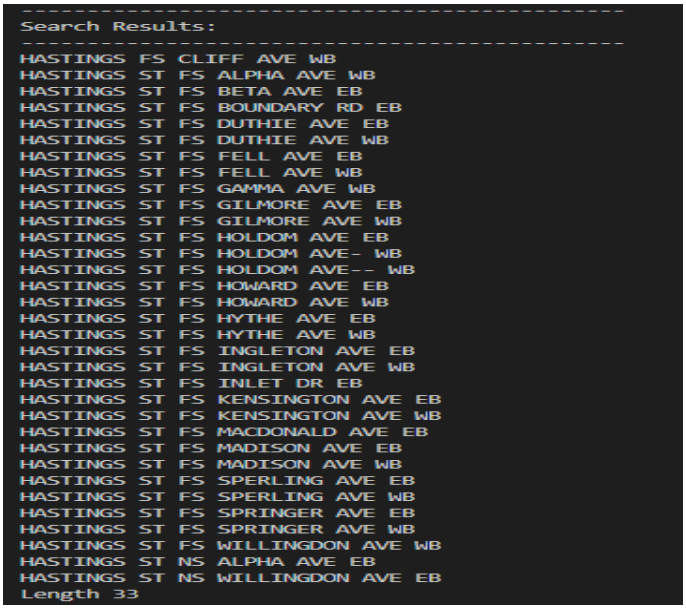


And once all the input files had been parsed through, the program takes in the initial stop and the destination stop from the main function, and uses Dijkstra's Shortest Path Algorithm to find the shortest path between the two stops, calculates the cost to travel between them, and also prints a list of stops between the two chosen spots. For eg: The results for the

stops “1111” and “2222”. We used **Dijkstra** because we thought it would be far more efficient to find the shortest path between any two stops rather than using Floyd Warshall where we would find the shortest path between every stop.

- ## Part 2:

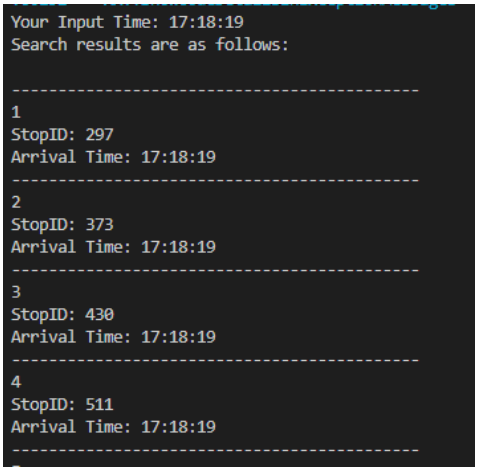
Part 2 involved implementing a search function where the user has the option to enter a string which contains the first few words from a bus stop’s name, then displays a list of meaningful stops. Inorder to do this, we used a Ternary Search Tree as instructed by the assignment requirements. I created a TST class which implements a Node class which contains the properties of a node in the tree.



Another requirement from the project specification required us to make the stop names more meaningful, which meant moving keywords like FLAGSTOP, WB, SB, EB and NB towards to the end of the string. For eg: “WB HASTINGS ST FS HOLDOM AVE” becomes “HASTINGS ST FS HOLDOM AVE WB”.

- ## Part 3:

This part required us to implement a feature where the user can input a string of the format “hh:mm:ss” and then the program displays a list of bus trips that are arriving at the same time, where all the buses are sorted by means of the Stop ID. Inorder to do this, I parse the stop_times.txt file and use only the entries that have valid arrival_time and once again use the **tripInfo** class from part1, and store



it in an ArrayList<tripInfo>. The program then searches for bus trips which have arrival_time equal to the input time and displays them sorted by tripID. I used **bubble sort** to fulfill my sorting purposes.

(Image Doesn't Contain Full Result)

- **Part 4:**

To create a simple UI interface, I used the terminal screen where I implemented this ASCIIArtGenerator Library to create some ASCII art to make it a little more visually pleasing. I also created an infinitely-continuous menu where the user has the option to either choose 1/2/3/4 for Part 1 func./ Part 2 func. / Part 3 func. / Quit the program.