**Algorithms & Data Structures II**

**Final Project**

**Design Document**

# Design Flow:

I wanted to start designing the project incrementally, by doing the tasks one at a time and then fine tuning them.

So,

1. Begin Task 1
   1. Implement methods to read in all three files to begin filling data structures
   2. Implement a method add vertices and edges to an Edge Weighted Digraph
   3. Implement a method to run a Dijkstra shortest path algorithm and print the path.
2. Begin Task 2
   1. Build on reading in the stops to fill a Trinary search tree
   2. Edit the stop names, moving NB/EB/SB/WB from the start to the end of a string
   3. Implement a method to run a keys with prefix method and print all lines that match the criteria
3. Begin Task 3
   1. Build on reading in stop\_times.txt to fill a new data structure dedicate to arrival times
   2. Implement a method to add values to a red-black binary search tree
   3. Implement a method to return all lines from stop\_times.txt that contain the specified arrival time
4. Begin Task 4
   1. Implement a new file, that contains a main method with several loops to repeatedly ask the user for what method they wish to run and their inputs
   2. Begin Error Handling and Management
   3. Clean up the code

# Analyses

## Outside of tasks: Start up:

Before the program can run any of the tasks, it must complete two methods, *fillStops()* and *fillGraph(), fillStops()* uses a buffered reader to read in the stops.txt file line by line, in it lies a while loop that will run until it has read every line in stops.txt, hence we can consider so far a worst time of O(N), almost all operation occurring in that while loop are constant time operations including adding to an arraylist, hashmap and searching the hashmap for a key, but it calls another method to fill a TST, which must check if a key is present first, which has a worst case time complexity of O(LogN), hence we can assume the worst case runtime for *fillStops()* is O(NLogN) and a space complexity of O(N).

For *fillGraph()*, the construction of the digraph has a time and space complexity of O(V + E), the number of vertices + the number of edges. The method is comprised of two sections, one to read in in transfers.txt, which it’s process only contributes to the total space time complexity, and the other reads in stop\_times.txt, that however references another method that fills a red-black binary search tree, which uses contains and put which take O(LogN) time. Hence knowing E1 + E2 = E, the worst case runtime for *fillGraphs()* is O(V + E1 + E2LogE2) with a similar space complexity.

## Task 1: Shortest Path Algorithm:

The shortest path algorithm starts with several error checks that have a constant run time, then it reads in values from a hashmap and hashtable for the algorithm, each only have a runtime of O(LogN), before reaching a construct for the Dijkstra shortes path algorithm which has a time complexity of O(ELogV) and O(V) extra space. hasPathTo() is a constant time operation but pathTo() has a worst case time complexity of O(V). Hence we can assume the method has a worst case time complexity of O(V + ELogV) and addition space complexity of O(V).

## Task 2: Searching stops Algorithm:

## Task 3: Searching Times Algorithm:

## Task 4: Interface Program:

# References: