CSU22012: Data Structures and Algorithms II Group Project Design Document

Abdelaziz Abushark 20332134

System Design

1. Shortest Path between bus stops

The primary goal was to find shortest paths between 2 bus stops and then return the stops and cost of the trips along the way. For this function, we used the Dijkstra shortest path algorithm.

Dijkstra's technique, by definition, can be used to find the shortest path from one node in a network to every other node in the graph data structure if the nodes can be reached from the starting point. Dijkstra is a fitting algorithm since every stop can be reached from any beginning point. I also don't have negative weights in this environment because my related costs are physical distances, hence Dijkstra is the best method to apply.

The shortest path algorithm developed by Dijkstra is a greedy algorithm that chooses the best choice locally at each stage. When it comes to bus routes, every time the bus stops, it will take the quickest path to the next targeted stop.

Other acceptable alternatives, such as the Floyd Warshall or Bellman Ford shortest routes algorithms, have higher runtime complexity than Dijkstra's Algorithm. It has an O(n.logv) complexity, where v is the number of vertices in the graph and O is the method's upper bound worst case runtime. Floyd Warshall's runtime is O(n3), which is far longer than Dijkstra's. The Bellman Ford algorithm, on the other hand, has an O (n.e) runtime, where 'e' is the number of graph edges.

Apart from the time constraints, there are a few more reasons we chose Dijkstra over competent opponents. The Floyd Warshall algorithm is used to find the shortest path from one point to every other reachable point. In this case, all we needed to do was find the quickest route to a given spot. As a result, the Dijkstra algorithm performed better under this scenario.

Dijkstra was also preferred over the Bellman-Ford algorithm. Even though B-F can accommodate negative-weight edges in its computations (which we don't need), the algorithm's runtime, space complexity, and overall performance are all below average in this scenario.

I did not use nor implement A* shortest path algorithm as I was more familiar with Dijkstra, and It was the easiest one to implement as I searched it up online and got some of the functions from the Algorithms 4th Edition Textbook by Robert Sedgewick and Kevin Wayne.

2. Bus Stop Search

The second part of the task was finding a bus stop using its entire name or just a few characters in its name and then returning information about it. Lecturer advised us to use TST, so I created a Ternary Search Tree that I got some functions from the Algorithms 4th Edition Textbook by Robert Sedgewick and Kevin Wayne to search for information about the system's stops in an efficient and simplified manner (TST). String input files are commonly ordered and searched using TSTs. While this is not considered the most efficient algorithm in terms of runtime (running at O(n) worst-case), it is the most efficient for space complexity, which suited this project well due to the large amount of data to be processed and stored. TST is a preferable sorting algorithm over others as it is an industry implementation standard when it comes to iterating, sorting string, and searching based on large data.

The main advantage of using ternary search trees over tries is that ternary search trees take up less space (involve only three-pointers per node as compared to 26 in standard tries). In addition, ternary search trees can be utilized in any situation where a hash table is used to hold strings. Tries are appropriate when there is a proper distribution of words throughout the alphabets, allowing for the most efficient use of space. Aside from that, ternary search trees are preferable. When the strings to be stored all have the same prefix, ternary search trees are the most economical (in terms of space).

Certain keywords had to be moved from the beginning to the end of the stop names as part of the requirements. A loop relocated the initial word to the end of the string until it wasn't a keyword when entering data into the TST. As a result, if the name of a stop was six words long and included one keyword, the keyword would become the sixth word in the sequence. The keyword would be the third word in the sequence if the same method was applied to a query using the first three words of the stop's name, and no match would be discovered. Instead of deleting keywords from the inputted string during the search, we added them to an Array list to address this. The keywords of these stop names were tested against the values in this Array list after our TST returned all possible bus stops. The stop would be included in the output if they matched.

3. Search for trips with Given Arrival Time

The final part of the task required you to look for all trips with a given arrival time. An Array list was used to hold the data. Even though binary search has a superior asymptotic time with a time complexity of O(N), it requires random access capabilities, which Linear search does not, hence it was better to be chosen for this section of the assignment. I used regex in which I researched online to match the stop-times.txt for me is the easiest and most efficient way as it defines a search pattern using symbols and allow you to find matches within strings.

To read the information more quickly and conveniently, a buffered reader was used. After that, it was put into an Array list data structure. The 'TripID,' 'Arrival Time,' 'Departure Time,' ,'StopID', ‘Stop sequence ‘, ‘stop head sign ‘, ‘pickup time ‘, ‘drop off time‘, and ‘distance traveled‘ were all read from the file then A for loop was created, and it continues to show the number of matches in ascending order by tripID number. This makes it very simple for the user to read through the available stops that are scheduled to arrive at the time of the user's input. When the function is run for a certain amount of time, the console displays a list of all possible outcomes as it will make it easier for the user to read via the available stops that have an arrival at the time of the user

4. Summary

I believe that I implemented the algorithms in the most efficient method possible in terms of projects time and space complexity. As well as I researched some of the methods online to help me complete this project and referenced them below

5. References:

https://algs4.cs.princeton.edu/44sp/
https://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/IndexMinPQ.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/code/edu/princeton/cs/algs4/Stack.java.html
https://algs4.cs.princeton.edu/52trie/TST.java.html
https://stackoverflow.com/questions/4736/learning-regular-expressions
https://regexr.com/