CSU22012: Data Structures and Algorithms Group Project Design Document

Stephen Davis	18324401
Devin O’Keefe	19334915
Aoife Khan	19335981
Alice Doherty	19333356

Finding Shortest Path Between Bus Stops (Part 1)

Dijkstra's shortest path algorithm was used to find the shortest path between 2 bus stops as inputted by the user. The A* shortest path algorithm was considered since this algorithm is effective for single pair shortest path, however, finding an accurate heuristic proved to be problematic, which is needed for the A* approach. Since this assignment involves no negative weights, and possibly involves cycles, Dijkstra's shortest path algorithm seemed most suitable to implement.

The worst case asymptotic running time of Dijkstra is O(ElogV), with extra space of V being used, where E is the number of edges in the graph and V is the number of vertices in the graph. Bellman-Ford was another option, however this algorithm has a worst-case asymptotic running time of O(EV), which is slower than Dijkstra's O(ElogV). Bellman-Ford's worst-case extra space requirement is the same as Dijkstra's, with V, ultimately proving Dijkstra to be superior in terms of efficiency.

In order to return the list of stops en route as well as the associated "cost", a Stack was used.

Originally, the stop_id of each bus stop was added as the vertex in the graph, however, this resulted in an array index out of bounds exception for any stop_id that was greater than 8756 (the total number of stops). To resolve this issue, a HashMap was implemented, which uses the stop_id as the key, and the associated index for each value.

The extra space requirement for the HashMap was O(E), where E is the number of edges in the graph.

Bus Stop Search (Part 2)

A Ternary Search Tree (TST) was used to search for bus stop information, as detailed in the requirements. Our program used each bus stop's name as a key and split the bus stop's data into a string array, to be used as the value. Insertion into a TST has a worst-case time complexity of O(N) and an average-case time complexity of O(log N).

It was necessary to modify the search operation, as we needed to return the value of the node associated with the key and also the values associated with all of its child nodes. To accomplish this, we recursively traversed each child node. During traversal, if the current node was not null, its value would be added to a list containing all possible values and its children would be traversed. Traversal had a worst-case time complexity of O(N). Similar to insertion, search also has a worst-case time complexity of O(N) and an average-case time complexity of O(log N).

The primary advantage of a TST over R-way tries and other alternatives is its space efficiency. Whereas an R-way trie has R null links for each leaf, TSTs only have 3 null links for each leaf. TSTs are also as fast as hashing when dealing with string keys, while also being space efficient. As such, TSTs are an excellent option for these dictionary implementations, due to their superior space efficiency.

As part of the requirements, we had to move certain keywords from the start to the end of the stop names. When inserting data into the TST, a loop moved the first word to the end of the string, until the first word was not a keyword. As such, if a stop's name was six words long and contained one keyword, that keyword would become the sixth word in the sequence. However, if the same method was applied to a query containing the first three words of the stop's name, the keyword would be the third word in the sequence and no match would be found. To fix this, we instead deleted keywords from the inputted string during search and added them to an arraylist. After our TST returned all possible bus stops, the keywords of these stop names were compared against the values in this arraylist. If they matched, the stop would be included in the output.

Search for Trips with Given Arrival Time (Part 3)

A linear search was used to search for given arrival time from the input files and return matching trips. Although binary search has a better asymptotic time complexity than linear search, which has a worst case time complexity of O(N), it requires the data to already be sorted and requires random-access capability. Linear search does not have these requirements and therefore was the best suited algorithm to use given the data provided.

To sort the matching searches by trip ID quicksort was used, which has a worst case performance of O(N^2). However, quicksort is the most efficient choice when considering its average case compared to any other sort algorithm, such as mergesort. Quicksort also requires little space as compared to mergesort, which requires a temporary array to merge the sorted arrays. Therefore, quicksort was the best choice to use when sorting the given data by trip ID.