CSU22012 Algorithms & Data Structure 2

Design Document

Group Members	**Student ID**
Bernadine Lao	19333627
Merlin Prasad	19333557
Filip Kowalski	19334250
Darren Aragones	19335456

Introduction

We created a system to help passengers of the Vancouver public transport system. We ran our code on eclipse version 2019-06(4.12.0) and no additional dependencies were required.

We were asked to provide the following three functions :

1. 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”.
2. 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.
3. 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.

We decided to use a terminal based user interface that handles erroneous input from the user and gives them feedback on what went wrong. It was a simple but effective way to provide the user access to the functions asked for . The user inputs a clear and simple .

Finding the shortest path was implemented with an adjacency matrix and dijkstra. We asked the user to input the stop_id of the two stops they wanted to find the shortest path between .

Searching for a bus stop was implemented by TST. The user was asked to input characters of the bus stop name and a list of results was returned in alphabetical order.

Searching trips by arrival time was implemented using an ArrayList. An object StopTimesInfo was created. A function getStopsInfo() takes in arrival time preferred in the format hh:mm:ss as parameter and returns a list of stops with the given arrival time as well as the information related to it sorted in an ascending order by the trip_id.

Proof read above as i might have gotten it wrong

Shortest path between two stops

Data structure : Adjacency Matrix

- Space complexity: O(N^2)
- Access time : O(1)

Adjacency matrix was used to represent the graph of all the paths . We decided to use this over an adjacent list as it did not require any additional classes to be built . An adjacency matrix does have a space complexity of O(N^2)..

The method to create the matrix has time complexity of ____ . However we only run this method once and after words the search time to access an element in the array is O(1) making our dijkstra algorithm that much faster . Searching in an array is much quicker than searching in a linked list which is what an adjacency list could have been represented by.

We traded space efficiency for faster search access time .This is why we decided to use an adjacency matrix to represent the graph for this method.

(is this a dense graph ? as an array is optimal for dense graph also going to make this shorter)

Algorithm : Dijkstra

- Time complexity : O(N^2)

We used the Dijkstra algorithm to find the shortest path between two nodes . It has a time complexity of O(N^2). This is much quicker than some of the other algorithms we considered such as Floyd which has a time complexity of O(N^3). Given the large data set we are dealing with time efficiency is crucial. Dijkstra is one of the most optimum algorithms to find the shortest path between two nodes in a graph . The graph also contains no negative weights as we are measuring distances so dijkstra works well.

Searching for a bus stop

Data structure: Ternary Search Tree (TST)

- Search hit : O(L+ln(N)) average case
- Search miss : O(ln(N)) average case
- Insert : O(L+ln(N)) average case

Implementing TST is much more efficient than its alternative of hashing as it's faster. It's also more flexible than red-black BST and supports character based operations . We are only searching by strings so it seems like the best thing to implement . **(note we aren't really given a choice so why am i arguing for it lol?)**

Algorithm

Idk what goes in here :|

Search all trips with a given arrival time

Data structure: ArrayList

- Space complexity: O(N)
- Access time: I mean.... Its easy to get stuff? I love you its fine I was just trying different formats. BTw i get O(1) and O(N) so maybe ill just remove this

Algorithm : Collections.sort (Merge sort)

actually nvm idk what collections.sort is for sure research said Tim sort? Which is an improvement of merge sort but someone look it up and check more? This analysis is based on merge

- Time complexity : O(NLogN)
- Space complexity: O(N)
- Stable : Yes
- In-place : No

We used this collections sort which implements merge sort to sort the data in ascending order because it was stable . This ensured the stops order was preserved after sorting compared to quicksort which is not stable even if it has a better space complexity.

Collections sort also works on array lists and other data structures which was good as we stored the data in an arraylist and arrays.sort would not be able to work on it.

Merge sort also has a guaranteed worst case runtime of O(nlogn) compared to quick sort which has a O(N^2) . Merge sort is optimum for sorting an array of objects and this is why we used it to implement the sorting.

Btw is this too long she said we can go over 2 pages and its more just nicely formatted? Its currently like 4 pages.??

What do you think of the format i can remove the bullet points to reduce space and probably condense the writing more