Colby Allen

Professor Jim Ashe

Data Structures and Algorithms II – C950

21 October 2020

Final Project Write-Up

A. The algorithm that I chose to deliver my packages most efficiently was the *Greedy Algorithm*.

B. 1. Please see attached CoreAlgOverview.pdf.

2. This program is written in Python3 and I wrote and debugged this whole project using Visual Studio Code. While it wasn't a project requirement, I did use Docker for development and building just to maintain projects between different development environments.

3. I have included code comments throughout the program that has Space/Time Complexity. I would say that the Big O of Time complexity for the entire program is O(n^2) and Space of O(n).

4. The problem that is being solved by our program has no perfect solution, so scalability is an issue. The pros of this are that the priority packages are all delivered first so that there is no fear that they will be delivered late. The tradeoff for this solution is that non-priority packages could be delivered between the delivery of priority packages, which a more advanced algorithm that implemented checking for reducing redundancy fewer miles would be driven.

5. The efficiency as I wrote in B3, runs at O(n^2) which works well. The maintainability of this software is very good. I have modularized a vast majority of the code in this project, and documented it well, as well. I am very impressed with how this project turned out, and I think you will be too!!

6. The hash table that I created for this project is a self-adjusting data structure, as it can

adjust and update all of the packages that are put within it. Putting and Getting methods also run at O(1) time which is amazing.

C. This code is all original (with examples taken from the Zybook) and meets all of the acceptance criteria. The user can check the status of the package at any time between 8:00 to 17:00.

D. The hash table is built and can be found in the src/classes/package_table.py file. This hash table will double hash a Package Id and then determine the index in the array/list that the item will be placed at. The retrieve will calculate the same hash value to know where to grab the package out of the array.

E. Created. Found in src/classes/package_table.py

F. “”

G. A prompt will ask the user at the beginning of the run to see if they want to see the packages at a certain time. The user can decide yes or no and if yes, they can decide the time at which all the package statuses will be shown.

H. Screenshot provided.

I. 1. The strengths of the chosen algorithm are that it will always make sure that the priority packages are delivered on time and it always makes the choice that is best for the algorithm in the short term and not looking to the future.

2. The output at the end of the program shows the total miles that were driven by the trucks and allows for user input.

3. A dynamic programming solution would have applied to this problem or a Tree BFS algorithm. Dynamic programming would have added a lot of space complexity to this problem that the greedy algorithm did not, the BFS algorithm would have to be refactored the Graph I had made already to search for optimal routes this would have added EXTREME amounts of time complexity as it takes a long time to complete.

J. I would have done differently not having to clean the addresses, as that was an absolute nightmare to deal with until I knew what I was doing.

K. All verifications are met.

1.

a. The hash table that is used in this program is extremely O(1) efficient, as it doesn't have to search all through an array each time to look for a package. The type of data that was used in this project was Package objects that are indexed based on their unique "Package IDs". The program uses this Hash Table in order to have a universal entrypoint to retrieve the packages that are placed onto the trucks, change the delivery status of the packages, and to print out all of the package information.

b. The Overhead in the buit Hash Table is very minimal. No matter how many packages you add to the hash table, getting and setting the data within the Hash Table will always remain consistently at O(1). The space complexity of the Hash Table will remain at O(n), but for storing all of your information, you cannot achieve better results than that. As the scope of this operation grows and grows, Time Complexity will remain at O(1) and Space Complexity at O(n).

c. If more packages were added to the hashtable, it would still perform optimally, but it would need to be modified to hold more than 40 packages. Adding additional trucks would increase the number of times that the simulation must be run, which would be adding roughly O(t*n) [where t is the number of trucks] to the Time Complexity, which seems like it could be very bad for runtime performance actually still boils down to a Time Complexity of O(n). Adding additional cities to this program, while entirely out of scope, wouldn't affect the performance of the greedy algorithm as the greedy algorithm chooses the address to go to next purely on distance. But the adding of more addresses to this project will increase the amount of time to run the greedy algorithm, as the truck will have to compare more and more addresses to find its next destination.

2. Other data structures that could have been used in place of a hash table could have been a set or an array. The set would only be able to hold the whole object and would only be accessible by having the whole object available to pass into the get. The array would add a lot of time complexity to the program as it would have to search through the entire array every time to find the whole object.