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 in the most efficient way was the *Greedy Algorithm*.

B. 1. This algorithm works on each of the trucks during the simulation, here is my pseudocode explained in plain English. For each truck, while the truck still has packages the truck will pull up its search through either the priority packages or EOD packages depending on if the priority packages still have any left. The list of packages are then scanned through to determine which package has the shortest distance from where the truck currently is. After finding the shortest distance, the truck delivers the package and then begins the loop again and pulls up the package list again. Once all of the packages are delivered, the truck will return to the hub.

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 have the 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 this being solved by our program really has no perfect solution, so scalability is definitely an issue. The good is that currently as it stands, the priority packages are all delivered first so that there is no problem they will be left behind. Now, a tradeoff because the packages are not delivered all in order, it will only target the

priority packages, which will be wasting time and miles that could be used to drive less and deliver non-priority packages along the way.

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 absolutely 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 is that it will always make sure that the priority packages are delivered on time as well as the fact that it always makes the choice that is best to 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 been applicable 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 in order 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. The hash table that is used in this program is extremely O(1) as it doesn't have to search all through an array each time to look for a package. The space complexity of this program is not very optimal, some corners were cut for fast lookups in the Verticie's by their address names. 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.

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.