C950 WGUPS Algorithm Overview

Joshua James

ID #011101737

WGU Email: jjam532@wgu.edu

Date: 7/5/23 (Revised: 7/8/23)

C950 Data Structures and Algorithms II

# Introduction:

For this assessment, I was tasked with creating and implementing an algorithm to route delivery trucks that deliver 40 packages, while ensuring to meet all delivery constraints, and keeping the overall traveling under 140 miles total.

# A. Algorithm Identification: Identify a named self-adjusting algorithm (e.g., “Nearest Neighbor algorithm,” “Greedy algorithm”) that you used to create your program to deliver the packages.

The self-adjusting algorithm I used for this project is commonly known as the “Nearest Neighbor” algorithm.

# B1. Logic Comments: Explain the algorithm’s logic using pseudocode.

I used a Nearest Neighbor algorithm for this project. A Nearest Neighbor algorithm is one way to solve this project’s variation of the traveling salesmen problem. It is easy to implement and executes quickly, but a drawback is that it can sometimes miss a more optimal route, as it is only checking between packages for the shorter distance. This is opposed to, for example, a brute force algorithm which would find the most optimal route by going through all possible combinations.

For this project, the algorithm first loops through every Package on the Truck. The second loops through each package id in the Truck, and package information (delivery address, delivery deadline, notes, etc.) is retrieved from the Package hash table via a search using the package id. Next, the distance between two of the Truck’s addresses and the Package address are stored, which is then used to check to see whether the next Package has a shorter delivery distance than the previous Package checked. The next Package to be delivered is determined by being the Package with the shorter delivery distance. The Truck’s distance traveled and time spent are recorded and updated at each delivery stop. The function continues to loop with the next Package to be delivered being the Package with the shorter distance, along with recording Truck distance traveled, and Truck time, until all Packages have been delivered (Package equals none). Once Package equals none, the function ends. Please note, this function uses each individual Truck as a parameter. Please see below for the pseudocode:

Function Nearest Neighbor:

Pass in: Truck

Set optimal Package equal to None

Set optimal Distance equal to 100000

FOR all package id in the truck’s current packages:

Package = Search Package hash table to obtain package information by each package’s package id

Distance = The distance between the Truck’s location and the Package’s address.

IF Distance is less than optimal Distance (previous package distance checked) and the package’s time delivered is none:

optimal Package equals Package

optimal Distance equals Distance

End IF

The next Package to be delivered equals optimal Package

IF Package does not equal none:

Distance = The distance between the Truck’s location and the Package’s address.

Calculate Truck Mileage (the total mileage traveled by the truck) by adding the current Distance to Truck Mileage after each delivery

Calculate Truck Time by adding Distance calculated as Time (minutes) to Truck Time after each delivery

End IF

The function loops until Package equals none (all packages have been delivered)

End FOR

End Function

# B2. Development Environment: Describe the programming environment you used to create the Python application.

I used Python 3.9 and PyCharm 2023.1.3 (Community Edition) for this project. I worked on this project using a Lenovo ThinkPad 13 business laptop. This laptop uses an intel Celeron 3865U processor chip, has 8 GB of RAM, and uses the Windows 10 Pro operating system.

# B3. Space-Time and Big-O: Evaluate the space-time complexity of each major segment of the program, and the entire program, using big-O notation.

The overall Big-O for this project is O(n^2). Please refer to program comments in “main.py” for space-time complexity for each major segment of the program.

# B4. Scalability and Adaptability: Explain the capability of your solution to scale and adapt to a growing number of packages.

There are a number of ways that my solution is able to scale and adapt to a growing number of packages:

* New packages can either replace existing package entries (since they have been delivered), or easily can be added to the existing hash table.
* Since a hash table was selected as the data structure to store the package data, the search time does not increase.
* The hash table is dynamic, and will increase in size as needed.
* Additional delivery trucks can be added to accommodate the increased number of packages.

# B5. Software Efficiency and Maintainability: Discuss why the software is efficient and easy to maintain.

My software is efficient and easy to maintain due to the following:

* The software is thoroughly commented on throughout, to ensure both readability and understanding.
* The hash table, dictionary, and list used in the program, will all adjust automatically as needed, whether they increase or decrease.

# B6. Self-Adjusting Data Structures: Discuss the strengths and weaknesses of the self-adjusting data structures (e.g., the hash table).

A hash table has many varying strengths and weaknesses:

Strengths:

1. Hash tables have fast look-up times (O(1)), due to a hash function being used to map keys to array indices.
2. Hash tables are space efficient due to only storing the key-value pairs, and the array to store them.

Weaknesses:

1. Hash tables have a finite storage capacity, and at some point, will become full.
2. Hash tables don’t preserve the order of elements.

# C. Original Code: Write an original program to deliver *all* the packages, meeting *all* requirements, using the attached supporting documents “Salt Lake City Downtown Map,” “WGUPS Distance Table,” and the “WGUPS Package File.”

Please refer to the “main.py” file submitted along with this overview.

# C1. Identification Information: Create an identifying comment within the first line of a file named “main.py” that includes your first name, last name, and student ID.

Please refer to the “main.py” file submitted along with this overview.

# C2. Process and Flow Comments: Include comments in your code to explain the process and the flow of the program.

Please refer to the program comments throughout the “main.py” file submitted along with this overview.

# D. Data Structure: Identify a self-adjusting data structure, such as a hash table, that can be used with the algorithm identified in part A to store the package data.

A hash table could be used with the algorithm identified in part A to store the package data.

# D1. Explanation of Data Structure: Explain how your data structure accounts for the relationship between the data points you are storing.

All 40 package objects are created, then a new hash table is created, in which the 40 package objects are then inserted into the hash table. The hash table is accessed and searched by using the hash table key, in this case, the Package ID. The address data is stored as a dictionary, and the distance data is stored as a list. These three data structures are then used by the algorithm to deliver the packages and compute the distance traveled.

# E. Hash Table: Develop a hash table, without using *any* additional libraries or classes, that has an insertion function that takes the following components as input and inserts the components into the hash table: Package ID number, delivery address, delivery deadline, delivery city, delivery zip code, package weight, delivery status.

Please refer to the “main.py” file submitted along with this overview.

# F. Look-Up Function: Develop a look-up function that takes the following components as input and returns the corresponding data elements: Package ID number, delivery address, delivery deadline, delivery city, delivery zip code, package weight, delivery status.

Please refer to the “main.py” file submitted along with this overview.

# G. Interface: Provide an interface for the user to view the status and info (as listed in part F) of *any* package at *any* time, and the total mileage traveled by *all* trucks. (The delivery status should report the package as *at the hub*, *en route*, or *delivered*. Delivery status *must* include the time.)

Please refer to the “main.py” file submitted along with this overview.

# G1. First Status Check: Provide screenshots to show the status of *all* packages at a time between 8:35 a.m. and 9:25 a.m.

# 

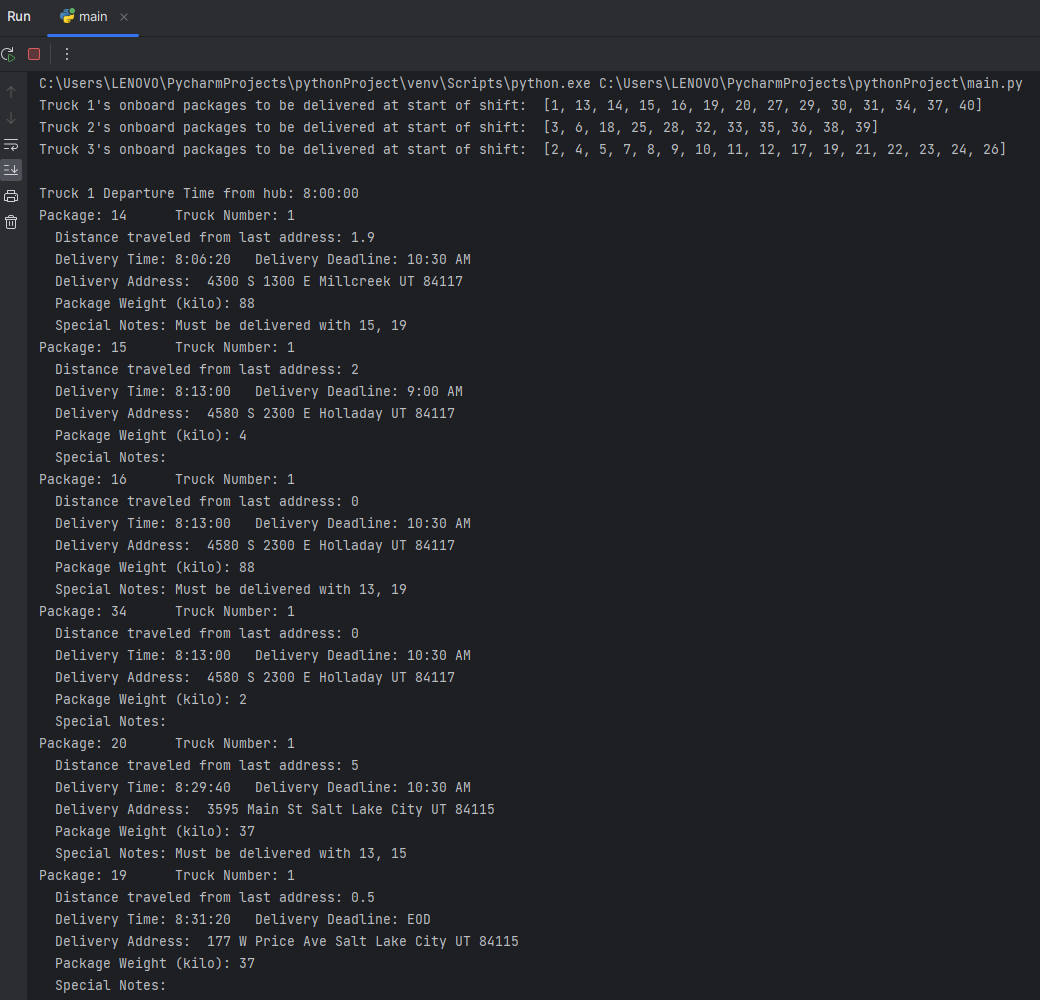
# G2. Second Status Check: Provide screenshots to show the status of *all* packages at a time between 9:35 a.m. and 10:25 a.m.

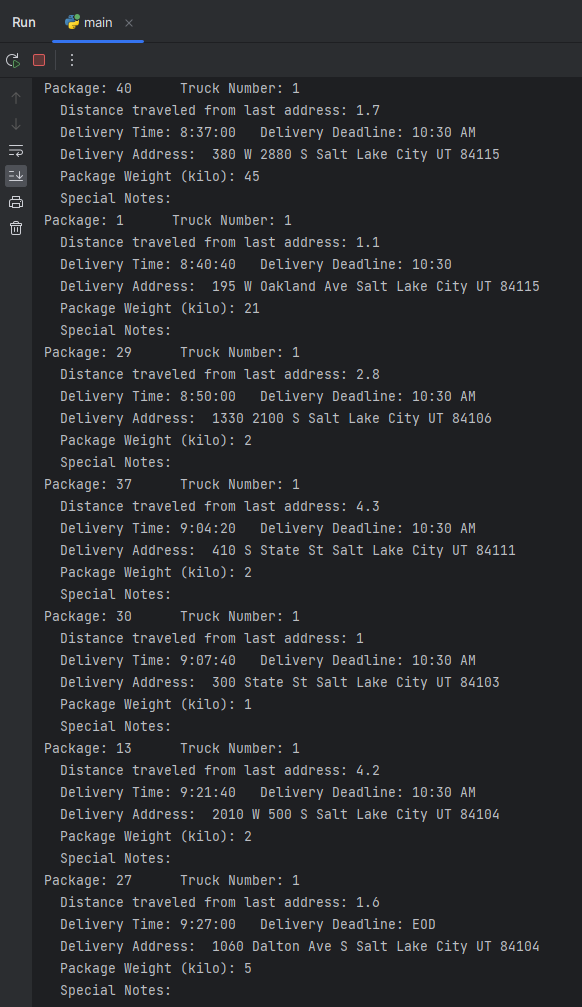
# 

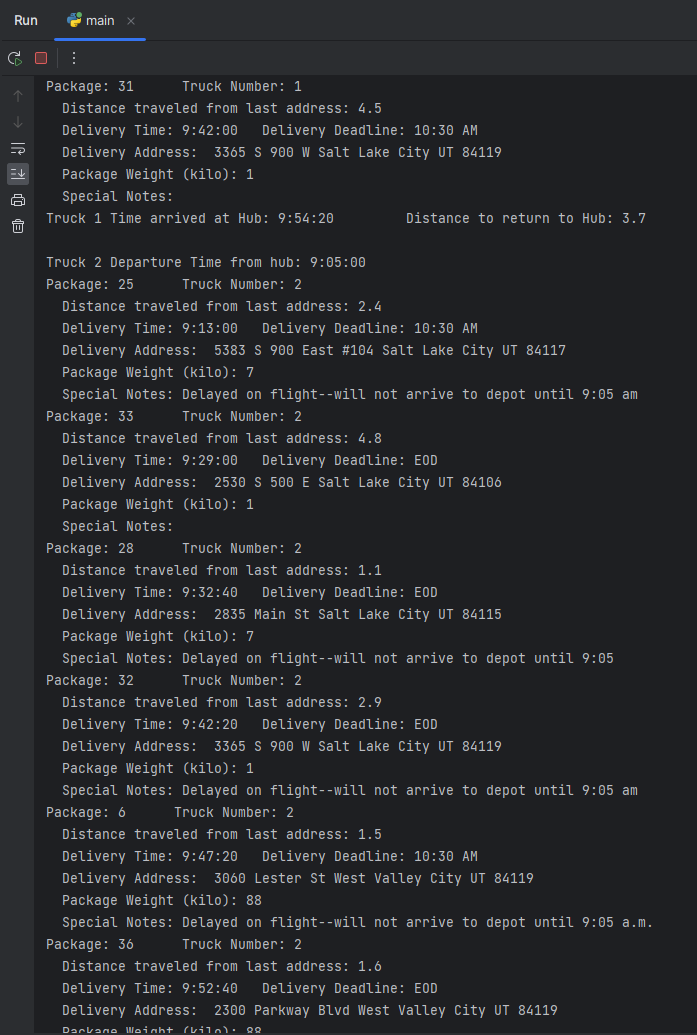
# G3. Third Status Check: Provide screenshots to show the status of *all* packages at a time between 12:03 p.m. and 1:12 p.m.

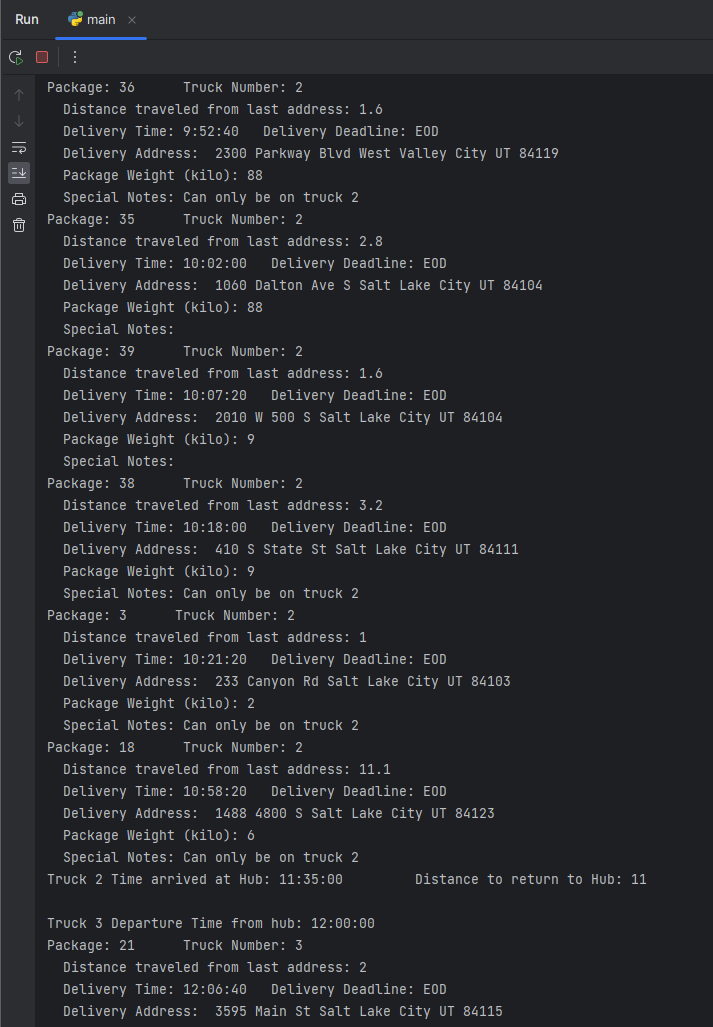
# 

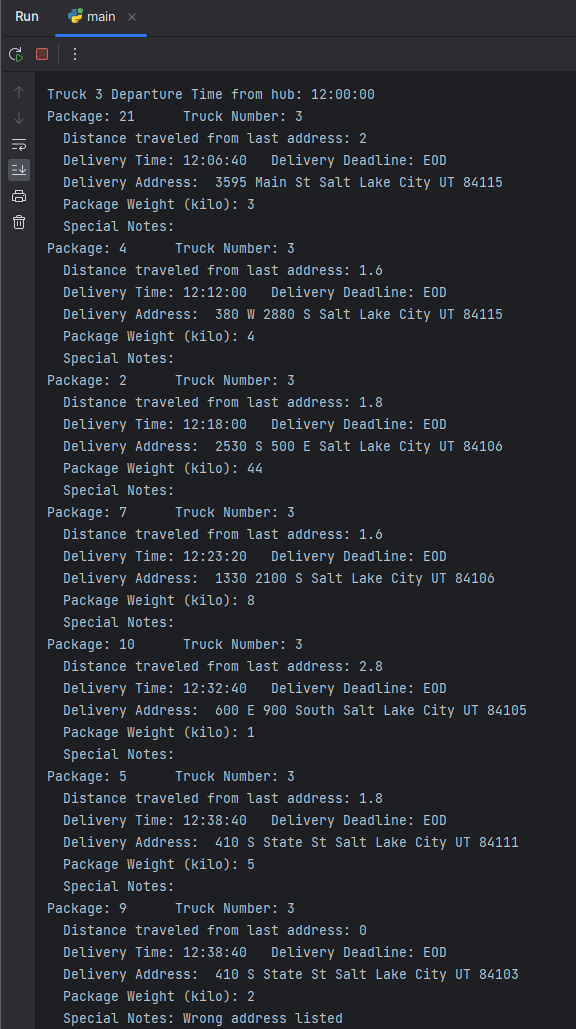
# H. Screenshots of Code Execution: Provide a screenshot or screenshots showing successful completion of the code, free from runtime errors or warnings, that includes the total mileage traveled by *all* trucks.

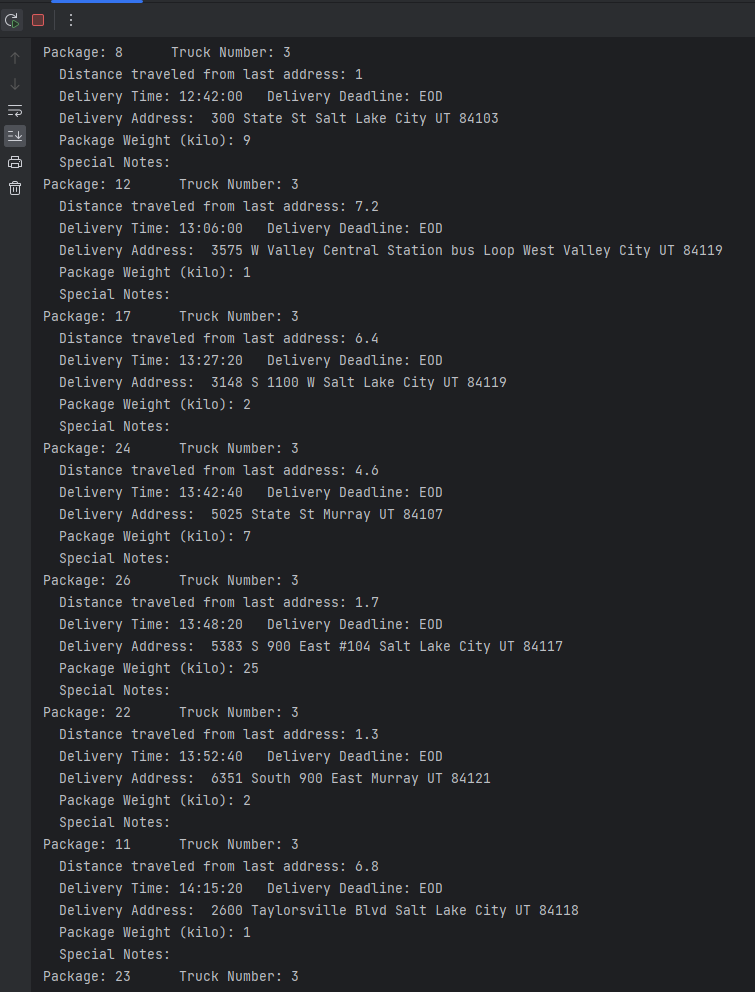


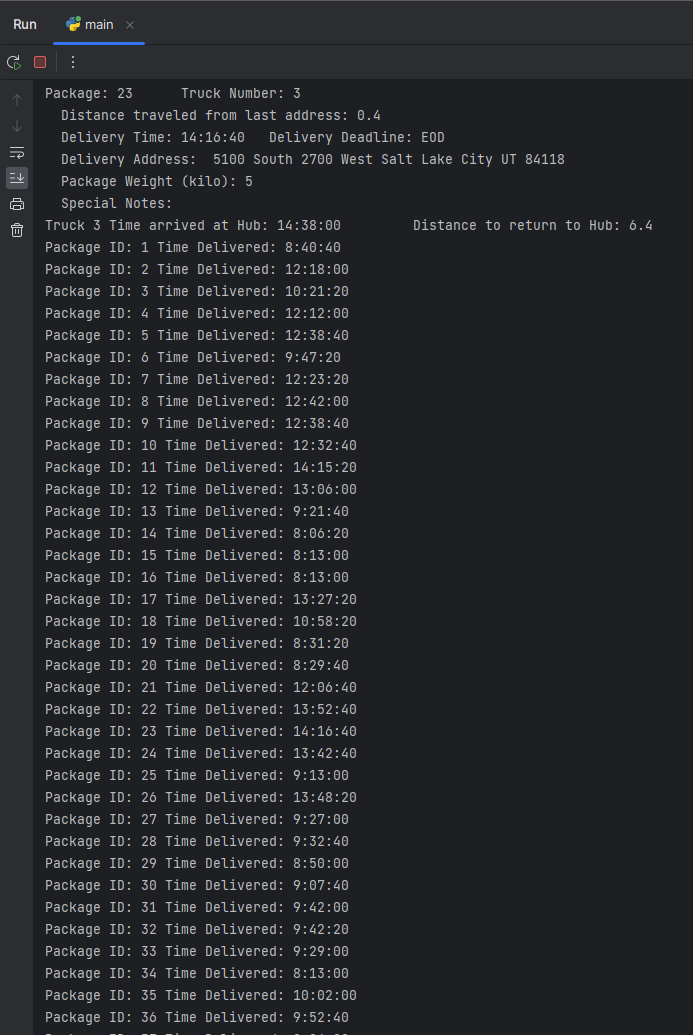


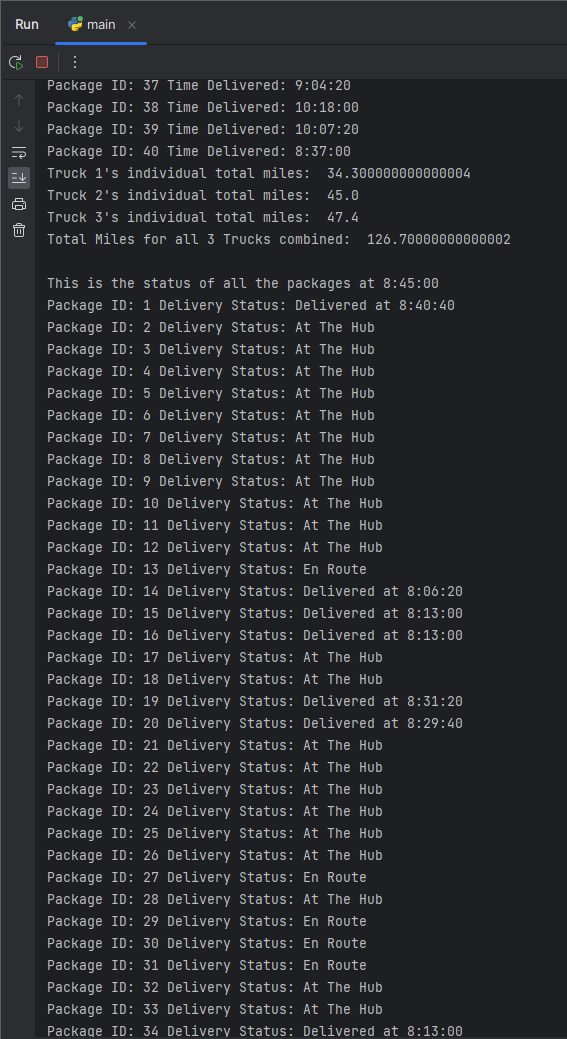


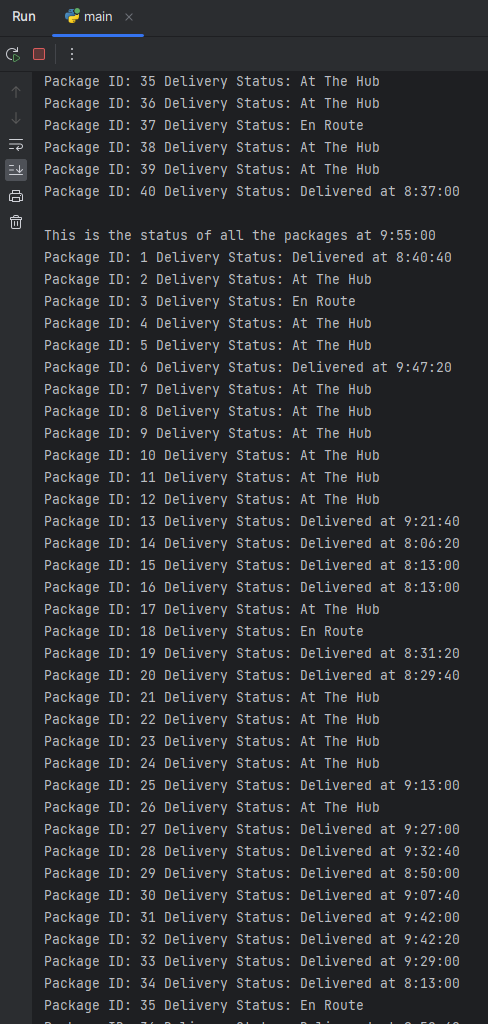












# 

# 

# I1. Strengths of Chosen Algorithm: Justify the core algorithm you identified in part A and used in the solution by doing the following: Describe *at least***two** strengths of the algorithm used in the solution.

There are multiple strengths of the “Nearest Neighbor” algorithm, but two of them include:

1. The algorithm is relatively simple to use, as it requires only a few lines of code.
2. The algorithm’s space time complexity is O(n), which is relatively fast and has a linear time complexity.

# I2. Verification of Algorithm: Justify the core algorithm you identified in part A and used in the solution by doing the following: Verify that the algorithm used in the solution meets *all* requirements in the scenario.

The algorithm meets all requirements.

# I3. Other possible Algorithms: Justify the core algorithm you identified in part A and used in the solution by doing the following: Identify **two** other named algorithms, different from the algorithm implemented in the solution, that would meet the requirements in the scenario.

There are multiple algorithms other than the “Nearest Neighbor” algorithm that could have been used, including:

1. Randomized algorithm
2. Brute Force algorithm

# I3A. Algorithm Differences: Describe how *each* algorithm identified in part I3 is different from the algorithm used in the solution.

1. A “randomized” algorithm would make the deliveries in a random order. This differs from the “Nearest Neighbor” algorithm, which checks between packages for the next closest package, instead of being random.
2. A “brute force” algorithm would solve the problem through exhaustion, as in, it would go through all possible combinations to come up with the best solution. This differs from the “Nearest Neighbor” algorithm, which checks between packages for the next closest package, instead of checking all possible combinations.

# J. Different Approach: Describe what you would do differently, other than the two algorithms identified in I3, if you did this project again.

I would create additional tables to store the hash table, packages, and truck classes. I would also use a csv reader to import the distance, address, and package csv files. Both of these steps would help with the readability, simplicity, and overall length of the “main” table, as well as help simplify scalability if needed, as it would be easier to add new data, without the risk of unintendingly interfering with code that would disrupt the software as a whole.

# K1. Verification of Data Structure: Justify the data structure you identified in part D by doing the following: Verify that the data structure used in the solution meets *all* requirements in the scenario.

The data structure meets all requirements.

# K1A. Efficiency: Explain how the time needed to complete the look-up function is affected by changes in the number of packages to be delivered.

The time needed to complete the look-up function is not affected by changes in the number of packages to be delivered. This is due to a hash table being used which has a look-up function space time complexity of 0(1).

# K1B. Overhead: Explain how the data structure space usage is affected by changes in the number of packages to be delivered.

A hash table’s space usage has a space complexity of O(n), so that means it is linearly affected by the number of packages.

# K1C. Implications: Describe how changes to the number of trucks or the number of cities would affect the look-up time and the space usage of the data structure.

A hash table’s time complexity is O(1), while it’s space complexity is O(n), therefore, a change in the number of trucks or the number of cities would not affect it’s look-up time. However, if the number of trucks or the number of cities were to be increased, then the look-up time would be linearly affected.

# K2. Other Data Structures: Identify **two** other data structures that could meet the same requirements in the scenario.

Two other data structures that could meet the same requirements in the scenario would be:

1. Binary Search Tree (BST)
2. Weighted Graph

# K2a. Data Structure Differences: Describe how *each* data structure identified in part K2 is different from the data structure used in the solution.

1. Binary Search Tree (BST): A BST is a binary tree that has an ordering property whereas any of the node's left subtree keys must be less than or equal to the node's key, and the right subtree's keys must be greater than or equal to the node's key. This differs from a hash table, as a hash table is a data structure that uses keys/value pairs and linked lists to store, insert, and search data, and does not maintain a sorted order like a BST does. In addition, other than in a worst-case scenario, a hash table is in most cases, faster than a BST, as a hash table has a time complexity of 0(1), while a BST’s time complexity is O(log n).
2. Weighted Graph: A weighted graph is a directed or undirected graph in which each edge is associated with a weight. If this were to be implemented instead of a hash table, one advantage would be that there would no longer be a need to compute and store the distances between each delivery address destination. However, one of the major trade-offs is that in a worse-case scenario, a weighted graph is slower compared to a hash table, as a weighted graph’s worse-case time complexity is O(n^2), while a hash table’s worse-case time complexity is O(n).

# M. Professional Communication: Demonstrate professional communication in the content and presentation of your submission.

# L. Sources - Works Cited: Acknowledge sources, using in-text citations and references, for content that is quoted, paraphrased, or summarized.

1. Lysecky, R., & Vahid, F. (2018, June). *C950: Data Structures and Algorithms II*. zyBooks.

Retrieved July 5, 2023, from <https://learn.zybooks.com/zybook/WGUC950AY20182019>

2. C950: Data Structures and Algorithms II. Webinar-1. *Let's Go Hashing*.

Retrieved July 5, 2023, from <https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=f08d7871-d57a-496e-a6a1-ac7601308c71>, https://srm--c.vf.force.com/apex/coursearticle?Id=kA03x000000e1fuCAA