

C950 Data Structures and Algorithms II

Eddy Bryan Leon Silva

Student ID: 001021060

eleon18@wgu.edu

WGUPS ROUTING PROGRAM

A. Algorithm Identification

The self-adjusting algorithm I used to create my program to deliver the packages was the Greedy algorithm. This algorithm uses the approach of solving a problem by selecting the most efficient next step available at that moment. This process cannot reverse a decision taken. While this algorithm finds a solution for the problem at hand, it is not the most optimal solution. The Greedy algorithm is optimal when applying a local solution leads to a global solution.

The algorithm works in three steps. First, the packages, distance and address data must be uploaded. Secondly, packages are loaded onto trucks according to the package's specifications (These could be deadlines, or special requests) and finally the algorithm delivers the packages.

B1. Logic Comments

The routing program solves the traveling salesman problem in 3 steps.

1. Data Upload: Data in the csv files provided, has to be read and parsed and stored. This data has pertinent information about each package, a list of addresses, and distance data between each address.

```
load_package_data(package_hashtable):
```

load_package_data is a function that takes in 1 argument, the hash table object where the packages will be stored. It utilizes the

csv reader to read data from a csv file and create package objects with that data. These packages are then inserted into the hashmap using the insert function defined in PackageHash.py

Args:

package_hashtable: This is a hash table object where package objects will be inserted

Returns:

This function doesn't return anything

Time complexity: Because the load_package_data function reads through the rows n times, corresponding to the number of packages the time complexity is $O(n)$

Space complexity: Because the hash table will be as big as n which references the number of packages the space complexity is $O(n)$

load_distance_data(distance_data_array):

load_distance_data is a function that takes in 1 argument, the list where the distance data will be represented. It utilizes the csv reader to read data from a csv file, change the data from string to a float, and add distance information into a 2D structured list.

Args:

distance_data_array: This is a list argument. This list should be empty and is the tool the main program will use to calculate distances

Returns:

This function doesn't return anything

Time complexity: Because the load_distance_data function reads through the rows m times for m times in the nested for loops, where m is the number of addresses which will always be a number equal or less than n (the number of packages) the time complexity is $O(n^2)$

Space complexity: The employment of a 2D list structure to hold the distance data utilizes m^2 memory spaces, where m (the number of

addresses) is a number always less than or equal to n (the number of packages) therefore the space complexity is $O(n^2)$

`load_address_data(address_data_array):`

`load_address_data` is a function that takes in 1 argument, the list where the address data will be stored. It utilizes the csv reader to read data from a csv file, parses it to avoid inconsistencies and adds addresses to the list provided.

Args:

`address_data_array`: This is a list argument. This list should be empty and is how the program will map addresses and their indexes to the `distance_data`

Returns:

This function doesn't return anything

Time complexity: Because the `load_address_data` function reads through the rows m times, where m is the number of addresses which will always be a number equal or less than n (the number of packages) the time complexity is $O(n)$

Space complexity: The list structure will need a list with size m , where m is the number of addresses and m will always be equal or less than n (The number of packages) the space complexity is $O(n)$

2. Load Truck Packages: After uploading package data and creating package objects to represent the packages to be delivered, they have to be placed on the WGUPS delivery trucks. The packages are loaded based on delivery criteria. Truck 1 and Truck 2 leave at the same time at 8:00, therefore these trucks are the only candidates for the packages with 9:00, 9:30, 10:00, and 10:30 deadlines. Then there are truck-only packages. These packages can be loaded only on a specific truck. There are packages that have to be delivered together. And the final constraint is that the number of packages a truck can hold at any given time is 16 and only 2 trucks can be in circulation.

`truck_load_packages()`:

`truck_load_packages` is a function that takes no arguments. It loads the packages into the corresponding trucks based on criteria identified through analysis of the `WGUPSPackageFile.csv` file

Args:

N/A

Returns:

This function does not return data

Time complexity: The `truck_load_packages` does a number of operations that add up to n (the number of packages), and iterations of all three loops is equal to n , making the time complexity $O(2n)$ which simplifies to $O(n)$

Space complexity: The list structure will need a list with size n (the number of packages), therefore the space complexity is $O(n)$

3. Deliver Packages: With all the pieces set up for the simulation of the variant of the traveling salesman problem, the program can finally run it. A function `truck_deliver_packages` takes in a parameter which is a truck object. This truck object has a list of packages that will be updated as the simulation is run. The simulation will add delivered times with the help of the function `min_distance_from`. To evaluate package status when a user enters a time, the program will use `get_status` to calculate package status

`truck_deliver_packages(delivery_truck)`:

`truck_deliver_packages` is a function that takes 1 argument, which is a truck object. As it iterates through the truck's packages it selects the next closest delivery address, changes its delivery status, and calculates total mileage until all packages have been delivered

Args:

`delivery_truck`: the truck which contains a `trailer` property which contains the packages to be delivered

Returns:

This function does not return data

Time complexity: truck_deliver_packages function has a loop that runs a maximum of 16 times per function call, but will run for a total of n times (the number of packages) in which a set number of value assignments, addition, multiplication and modulus operations are made, because of this the time complexity is $O(n)$

Space complexity: The total size of the list structure will be size 16 at any one time (the number of packages in the truck), therefore the space complexity is $O(1)$

Pseudo code:

Initialize list that contains the packages in the order that they were delivered starting from closer to the Hub

Initialize list that contains the distances traveled after every package delivery, this will aid in calculating total truck mileage

Enter loop that will run while the truck's trailer still contains packages that have not been delivered

[Loop start]

Get data from `min_distance_from` function

Add distance found to a list that will contain all distances traveled

Update package status after it has been delivered

Calculate delivery time

Create time object for intuitive time manipulation

Add package to list of ordered packages

Remove package from unordered list of packages

[Loop End]

Add total driven distance by the truck after all packages have been delivered

Add ordered list of packages back into the truck object

Store the distances traveled and their order in a list

`min_distance_from(from_address, truck_packages):`

`min_distance_from` is a function that takes in 2 arguments. The first is an address index, and the second is a list that holds all the packages being delivered by a truck. It analyzes distance data between the given address and the addresses of the packages that have to be delivered and picks the one closest.

Args:

`from_address`: This is an integer argument. This integer represents the starting location

`truck_packages`: This is a list object. This list contains all the packages loaded in a truck object

Returns:

This function returns 3 results which are the distance to the next closest package, the package to be delivered, and position which keeps track of what location the truck is at while delivering

Time complexity: Because the `min_distance_from` function has a loop that will iterate k times, where k is the number of packages loaded on the truck, k will always be a number less than a maximum of 16 and/or n (The number of packages read from the csv file). The number of operations will be at maximum 16 therefore the time complexity is $O(1)$

Space complexity: The function operates within a maximum space of 16 objects at a time, therefore the space complexity is $O(1)$

Pseudo code:

Unrealistic initial minimal distance in order to begin minimal distance assignment

Initialize a variable that will hold the address index for the next package address

Initialize a variable that will hold the package to be delivered next

Enter a loop that iterates through the truck's package list

[Loop start]

Look up the distance to the next package

If the distance selected previously is less than the one being stored in min_distance then a new package is selected

Assign new minimum distance

Update position with current package address

Variable to hold candidate package to deliver to next

[Loop End]

If clause is reached when the next package in the trailer is meant to be delivered to the same address with the conditional check of the variable that holds the current min_distance.

Update position with current package address

Minimum distance is 0 because next package is to be delivered at the same location

Enter a for loop to find the package with the 0.0 distance

[Loop start]

If the position of the truck is the same as the package then select the current package

[Loop End]

Return results(minimal distance, position, and package)

get_status(time):

get_status is a function that takes in a datetime argument and compares it to the packages' time of delivery and departure from the Hub via a reference to a datetime object in the truck it is in. After the comparison is complete a string about the status of the package will be formed.

Args:

time: This is a datetime argument that is taken from the user and formatted in the function user_console()

Returns:

This function returns a string containing information about the status of the package

Time complexity: Because `get_status` does 3 comparisons, value assignment and time addition plus formatting the time complexity is $O(1)$

Space complexity: Because there are three data spaces that will be accessed independently of the number of packages the space complexity is $O(1)$

Pseudo code:

Check if time given by the user is before the package has left the hub

 Return status as in "Hub" if true

Check if time given by the user is after the package has been delivered

 Return the stored time delivered if true

Check if time given by the user is in between leaving the Hub or being delivered

 Return status as "On Route" if true

B2. Development Environment

For the development of the program, python was the required programming language. The programming environment contains Python 3.1. In order to code and compile python code, PyCharm was used. This was installed on a Windows machine running Windows 10. The data the program reads is stored in CSV format and is accessed locally on the machine. There are no network dependencies, and the program can be developed with only offline capabilities.

B3. Space-Time and Big-O

Through analysis of the functions demonstrated in section B2 and other functions commented in the code, I have compiled a table with all functions and all files that contribute to the space time complexity of the program.

Package.py

Method	Time - Complexity	Space - Complexity
__init__	O(1)	O(1)
get_status	O(1)	O(1)

Truck.py

Method	Time - Complexity	Space - Complexity
__init__	O(1)	O(1)

PackageHash.py

Method	Time - Complexity	Space - Complexity
__init__	O(1)	O(1)
insert	O(1)	O(n)
search	O(1)	O(n)
remove	O(1)	O(n)

main.py

Method	Time - Complexity	Space - Complexity
load_package_data	O(n)	O(n)
load_distance_data	O(n ²)	O(n ²)
load_address_data	O(n)	O(n)
min_distance_from	O(1)	O(1)
truck_load_packages	O(n)	O(n)
truck_deliver_packages	O(n)	O(1)
get_truck_mileage	O(1)	O(n)
user_console	O(n)	O(n ²)

B4. Scalability and Adaptability

The program developed can scale with a growing number of packages. The `load_package_data` function reads through all the packages from the csv file specified, there is a single variable that initiates the size of the hash table that holds all the packages. The size of this hash table will take up n memory spaces, corresponding to the number of packages read. This function operates in linear time or better making it a proper candidate for scalable solution. The `load_distance_data` function reads through all the distance data in the csv file specified, in this case the space complexity for holding this data can be as big as exponential space. This is a disadvantage when trying to select it as a scalable solution for the WGUPS Routing Program.

The program can easily change its simulation parameters to meet different solution requirements. The program can add or subtract truck objects, it can control when the trucks can begin delivering, and how many trucks can be on the road at any time.

B5. Software Efficiency and Maintainability

The algorithm I am using in my program solution is the greedy algorithm. In terms of efficiency it might sound ironic to say this approach is efficient. However, when the number of packages is relatively small for a space complexity of $O(n^2)$, the program offers $O(n)$ time complexity for delivering the packages in the smallest and shortest set of distances identified.

The code is organized with the principle of writing clean and structured code that a developer other than me can also contribute to. The program contains 3 object constructors in separate files and one file where the objects are initialized and the logic is run. All functions are commented with what they do, what arguments they take, what elements are returned and an analysis of its space-time complexity.

B6. Self-Adjusting Data Structures

In order to address an unknown number of packages, a hash table data structure was developed. While the hash table wouldn't have to increase its size given that it has to be set initially, in case this was disregarded, the hash table can accommodate a number of packages higher than its initial capacity thanks to chaining. This operation can be expensive computationally, but useful when the number of packages is not known upfront.

C. Original Code

The program's code is attached with this document.

C1. Identification Information

Identification information has been written in the main file of the program “main.py” on line 1.

C2. Process and Flow Comments

Process and flow comments have been written in the code document attached, they explain the logic of the program.

D. Data Structure

A hash table is a self-adjusting data structure that can be used with the algorithm identified in part A to store the package data.

D1. Explanation of Data Structure

A hash table is a data structure that can store data with a unique key. It supports operations like insertion, search and removal of elements. In this implementation of a hash table with chaining, the structure is self-adjusting to accommodate an unknown number of packages. The packages read from the csv file contain a data field called package ID, a unique dataset. This ID can help in the creation of a key. The parallel of a hash table key and package ID and that of an element with a package itself, makes a self-adjusting hash table suitable to store package data.

E. Hash Table

The implementation of the hash table is in the “PackageHash.py” file attached. It contains the following functions

```
PackageHash(total_storage):
```

Truck is a constructor that takes in 2 arguments and assigns starting values for the truck object.

Args:

total_storage: This is an integer argument that denotes the total number of packages that will enter the system to be delivered

Returns:

This function returns a reference to the created object

Time complexity: Because the PackageHash constructor assigns as many empty lists as there are packages it has a time complexity of $O(n)$

Space complexity: Because there are n spaces being accessed, 1 for every package the space complexity is $O(n)$

`insert(package):`

`insert` is a function that takes in 1 argument, a package object and adds/updates it to the hashtable(`self.storage`).

Args:

`package`: This is a package object that will be inserted into the hashtable

Returns:

This function doesn't return data

Time complexity: Because the `insert` function calculates the hash key, uses a for loop to access a list that will always be size 2, and does value assignment, the time complexity is $O(1)$

Space complexity: Because the number of data that has to be referenced can be up to the size of the number of packages the space complexity is $O(n)$

`search(package_id):`

`search` is a function that takes in 1 argument, an id of a package and searches for it in the hashtable

Args:

`package_id`: This is an integer argument that represents a package id

Returns:

This function returns the package if a package is found otherwise returns `none`

Time complexity: Because the search function is successful by finding the package with the package_id as a key makes the time complexity is $O(1)$

Space complexity: Because the data that has to be searched through can be up to the size of the number of packages the space complexity is $O(n)$

`remove(package_id):`

remove is a function that takes in 1 argument, an id of a package and removes it from the hashtable

Args:

package_id: This is an integer argument that represents a package id

Returns:

This function returns the package if a package is removed otherwise returns none

Time complexity: Because the remove function is successful by removing the package by accessing where the package is the time complexity is $O(1)$

Space complexity: Because the data that has to be searched through can be up to the size of the number of packages the space complexity is $O(n)$

F. Look-Up Function

The lookup function can be utilized from the command line of the program. The user console contains a menu with 3 options. Option 1 contains the loop up function which will return all related data to a package ID. This is located in the "main.py" file, in the function `user_console`.

Below is a demonstration of the look-up function

```
Welcome to the WGUPS!
You can use this console to perform the following actions:
1 - Look up a package
2 - Status Report
3 - Exit
1
Enter a package ID: 21
-----
Package  21  Report
-----
Package ID:  21
Delivery Address:  3595 Main St
City:  Salt Lake City
Zip Code:  84115
Weight:  3
Delivery Deadline:  EOD
Status:  Delivered  at  10:26:00
-----
-----END-----
-----
```

G. Interface

The program contains a command line interface where the user can look up a package through its ID. In option 2 of this program, the user can ask for a status report. They must enter military time and the program will calculate status for the packages and mileage for the trucks.

```
Welcome to the WGUPS!
You can use this console to perform the following actions:
1 - Look up a package
2 - Status Report
3 - Exit
```

G1. First Status Check

```
Welcome to the WGUPS!
You can use this console to perform the following actions:
1 - Look up a package
2 - Status Report
3 - Exit

Specify a valid business hour(8:00 - 17:00) 9
Specify a valid minute number (0-59) 00
-----
Status Report for time  9 :00
-----
Package ID: 40 | Delivery Address: 380 W 2880 S | City: Salt Lake City | Zip Code: 84115 | Weight: 45 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:37
Package ID: 1 | Delivery Address: 195 W Oakland Ave | City: Salt Lake City | Zip Code: 84115 | Weight: 21 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:40
Package ID: 2 | Delivery Address: 2530 S 500 E | City: Salt Lake City | Zip Code: 84106 | Weight: 44 | Delivery Deadline: EOD | Status: On Route
Package ID: 3 | Delivery Address: 233 Canyon Rd | City: Salt Lake City | Zip Code: 84103 | Weight: 2 | Delivery Deadline: EOD | Status: On Route
Package ID: 4 | Delivery Address: 380 W 2880 S | City: Salt Lake City | Zip Code: 84115 | Weight: 4 | Delivery Deadline: EOD | Status: Delivered at 08:44
Package ID: 5 | Delivery Address: 410 S State St | City: Salt Lake City | Zip Code: 84111 | Weight: 5 | Delivery Deadline: EOD | Status: On Route
Package ID: 6 | Delivery Address: 3060 Lester St | City: West Valley City | Zip Code: 84119 | Weight: 88 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:49
Package ID: 7 | Delivery Address: 1330 2100 S | City: Salt Lake City | Zip Code: 84106 | Weight: 8 | Delivery Deadline: EOD | Status: Hub
Package ID: 8 | Delivery Address: 300 State St | City: Salt Lake City | Zip Code: 84103 | Weight: 9 | Delivery Deadline: EOD | Status: Hub
Package ID: 9 | Delivery Address: 410 S State St | City: Salt Lake City | Zip Code: 84111 | Weight: 2 | Delivery Deadline: EOD | Status: Hub
Package ID: 10 | Delivery Address: 600 E 900 S | City: Salt Lake City | Zip Code: 84105 | Weight: 1 | Delivery Deadline: EOD | Status: Hub
Package ID: 11 | Delivery Address: 2600 Taylorsville Blvd | City: Salt Lake City | Zip Code: 84118 | Weight: 1 | Delivery Deadline: EOD | Status: Hub
Package ID: 12 | Delivery Address: 3575 W Valley Central Station bus Loop | City: West Valley City | Zip Code: 84119 | Weight: 1 | Delivery Deadline: EOD | Status: On Route
Package ID: 13 | Delivery Address: 2010 W 500 S | City: Salt Lake City | Zip Code: 84104 | Weight: 2 | Delivery Deadline: 10:30 AM | Status: On Route
Package ID: 14 | Delivery Address: 4300 S 1300 E | City: Millcreek | Zip Code: 84117 | Weight: 88 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:06
Package ID: 15 | Delivery Address: 4580 S 2300 E | City: Holladay | Zip Code: 84117 | Weight: 4 | Delivery Deadline: 9:00 AM | Status: Delivered at 08:13
Package ID: 16 | Delivery Address: 4580 S 2300 E | City: Holladay | Zip Code: 84117 | Weight: 88 | Delivery Deadline: 10:30 AM | Status: On Route
Package ID: 17 | Delivery Address: 3148 S 1100 W | City: Salt Lake City | Zip Code: 84119 | Weight: 2 | Delivery Deadline: EOD | Status: Delivered at 08:42
Package ID: 18 | Delivery Address: 1488 4800 S | City: Salt Lake City | Zip Code: 84123 | Weight: 6 | Delivery Deadline: EOD | Status: On Route
Package ID: 19 | Delivery Address: 177 W Price Ave | City: Salt Lake City | Zip Code: 84115 | Weight: 37 | Delivery Deadline: EOD | Status: Delivered at 08:31
Package ID: 20 | Delivery Address: 3595 Main St | City: Salt Lake City | Zip Code: 84115 | Weight: 37 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:29
Package ID: 21 | Delivery Address: 3595 Main St | City: Salt Lake City | Zip Code: 84115 | Weight: 3 | Delivery Deadline: EOD | Status: Hub
Package ID: 22 | Delivery Address: 6351 S 900 East | City: Murray | Zip Code: 84121 | Weight: 2 | Delivery Deadline: EOD | Status: Delivered at 08:12
Package ID: 23 | Delivery Address: 5100 S 2700 West | City: Salt Lake City | Zip Code: 84118 | Weight: 5 | Delivery Deadline: EOD | Status: Hub
Package ID: 24 | Delivery Address: 5025 State St | City: Murray | Zip Code: 84107 | Weight: 7 | Delivery Deadline: EOD | Status: Hub
Package ID: 25 | Delivery Address: 5383 S 900 East #104 | City: Salt Lake City | Zip Code: 84117 | Weight: 7 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:08
Package ID: 26 | Delivery Address: 5383 S 900 East #104 | City: Salt Lake City | Zip Code: 84117 | Weight: 25 | Delivery Deadline: EOD | Status: Hub
Package ID: 27 | Delivery Address: 1060 Dalton Ave S | City: Salt Lake City | Zip Code: 84104 | Weight: 5 | Delivery Deadline: EOD | Status: Hub
Package ID: 28 | Delivery Address: 2835 Main St | City: Salt Lake City | Zip Code: 84115 | Weight: 7 | Delivery Deadline: EOD | Status: Delivered at 08:33
Package ID: 29 | Delivery Address: 1330 2100 S | City: Salt Lake City | Zip Code: 84106 | Weight: 2 | Delivery Deadline: 10:30 AM | Status: On Route
Package ID: 30 | Delivery Address: 300 State St | City: Salt Lake City | Zip Code: 84103 | Weight: 1 | Delivery Deadline: 10:30 AM | Status: On Route
Package ID: 31 | Delivery Address: 3365 S 900 W | City: Salt Lake City | Zip Code: 84119 | Weight: 1 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:49
Package ID: 32 | Delivery Address: 3365 S 900 W | City: Salt Lake City | Zip Code: 84119 | Weight: 1 | Delivery Deadline: EOD | Status: Delivered at 08:44
Package ID: 33 | Delivery Address: 2530 S 500 E | City: Salt Lake City | Zip Code: 84106 | Weight: 1 | Delivery Deadline: EOD | Status: Hub
Package ID: 34 | Delivery Address: 4580 S 2300 E | City: Holladay | Zip Code: 84117 | Weight: 2 | Delivery Deadline: 10:30 AM | Status: On Route
Package ID: 35 | Delivery Address: 1060 Dalton Ave S | City: Salt Lake City | Zip Code: 84104 | Weight: 88 | Delivery Deadline: EOD | Status: Hub
Package ID: 36 | Delivery Address: 2300 Parkway Blvd | City: West Valley City | Zip Code: 84119 | Weight: 88 | Delivery Deadline: EOD | Status: Delivered at 08:54
Package ID: 37 | Delivery Address: 410 S State St | City: Salt Lake City | Zip Code: 84111 | Weight: 2 | Delivery Deadline: 10:30 AM | Status: On Route
Package ID: 38 | Delivery Address: 410 S State St | City: Salt Lake City | Zip Code: 84111 | Weight: 9 | Delivery Deadline: EOD | Status: On Route
Package ID: 39 | Delivery Address: 2010 W 500 S | City: Salt Lake City | Zip Code: 84104 | Weight: 2 | Delivery Deadline: EOD | Status: Hub
Total Mileage by all trucks: 36.00 miles
-----
-----END-----
-----
```

9:00 AM Report

G2. Second Status Check

```
Welcome to the WGUPS!
You can use this console to perform the following actions:
1 - Look up a package
2 - Status Report
3 - Exit
>
Specify a valid business hour(8:00 - 17:00) 10
Specify a valid minute number (0-59) 00
-----
Status Report for time 10 :00
-----
Package ID: 40 | Delivery Address: 380 W 2880 S | City: Salt Lake City | Zip Code: 84115 | Weight: 45 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:37
Package ID: 1 | Delivery Address: 195 W Oakland Ave | City: Salt Lake City | Zip Code: 84115 | Weight: 21 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:40
Package ID: 2 | Delivery Address: 2530 S 500 E | City: Salt Lake City | Zip Code: 84106 | Weight: 44 | Delivery Deadline: EOD | Status: Delivered at 09:03
Package ID: 3 | Delivery Address: 233 Canyon Rd | City: Salt Lake City | Zip Code: 84103 | Weight: 2 | Delivery Deadline: EOD | Status: Delivered at 09:29
Package ID: 4 | Delivery Address: 380 W 2880 S | City: Salt Lake City | Zip Code: 84115 | Weight: 4 | Delivery Deadline: EOD | Status: Delivered at 08:44
Package ID: 5 | Delivery Address: 410 S State St | City: Salt Lake City | Zip Code: 84111 | Weight: 5 | Delivery Deadline: EOD | Status: Delivered at 09:29
Package ID: 6 | Delivery Address: 3060 Lester St | City: West Valley City | Zip Code: 84119 | Weight: 88 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:49
Package ID: 7 | Delivery Address: 1330 2100 S | City: Salt Lake City | Zip Code: 84106 | Weight: 8 | Delivery Deadline: EOD | Status: Hub
Package ID: 8 | Delivery Address: 300 State St | City: Salt Lake City | Zip Code: 84103 | Weight: 9 | Delivery Deadline: EOD | Status: Hub
Package ID: 9 | Delivery Address: 410 S State St | City: Salt Lake City | Zip Code: 84111 | Weight: 2 | Delivery Deadline: EOD | Status: Hub
Package ID: 10 | Delivery Address: 600 E 900 S | City: Salt Lake City | Zip Code: 84105 | Weight: 1 | Delivery Deadline: EOD | Status: Hub
Package ID: 11 | Delivery Address: 2600 Taylorsville Blvd | City: Salt Lake City | Zip Code: 84118 | Weight: 1 | Delivery Deadline: EOD | Status: Hub
Package ID: 12 | Delivery Address: 3575 W Valley Central Station bus Loop | City: West Valley City | Zip Code: 84119 | Weight: 1 | Delivery Deadline: EOD | Status: Delivered at 09:05
Package ID: 13 | Delivery Address: 2010 W 500 S | City: Salt Lake City | Zip Code: 84104 | Weight: 2 | Delivery Deadline: 10:30 AM | Status: Delivered at 09:40
Package ID: 14 | Delivery Address: 4300 S 1300 E | City: Millcreek | Zip Code: 84117 | Weight: 88 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:06
Package ID: 15 | Delivery Address: 4580 S 2300 E | City: Holladay | Zip Code: 84117 | Weight: 4 | Delivery Deadline: 9:00 AM | Status: Delivered at 08:13
Package ID: 16 | Delivery Address: 4580 S 2300 E | City: Holladay | Zip Code: 84117 | Weight: 88 | Delivery Deadline: 10:30 AM | Status: On Route
Package ID: 17 | Delivery Address: 3148 S 1100 W | City: Salt Lake City | Zip Code: 84119 | Weight: 2 | Delivery Deadline: EOD | Status: Delivered at 08:42
Package ID: 18 | Delivery Address: 1488 4800 S | City: Salt Lake City | Zip Code: 84123 | Weight: 6 | Delivery Deadline: EOD | Status: On Route
Package ID: 19 | Delivery Address: 177 W Price Ave | City: Salt Lake City | Zip Code: 84115 | Weight: 37 | Delivery Deadline: EOD | Status: Delivered at 08:31
Package ID: 20 | Delivery Address: 3595 Main St | City: Salt Lake City | Zip Code: 84115 | Weight: 37 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:29
Package ID: 21 | Delivery Address: 5025 State St | City: Murray | Zip Code: 84115 | Weight: 3 | Delivery Deadline: EOD | Status: Hub
Package ID: 22 | Delivery Address: 6351 S 900 East | City: Murray | Zip Code: 84121 | Weight: 2 | Delivery Deadline: EOD | Status: Delivered at 08:12
Package ID: 23 | Delivery Address: 5100 S 2700 West | City: Salt Lake City | Zip Code: 84118 | Weight: 5 | Delivery Deadline: EOD | Status: Hub
Package ID: 24 | Delivery Address: 5025 State St | City: Murray | Zip Code: 84107 | Weight: 7 | Delivery Deadline: EOD | Status: Hub
Package ID: 25 | Delivery Address: 5383 S 900 East #104 | City: Salt Lake City | Zip Code: 84117 | Weight: 7 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:08
Package ID: 26 | Delivery Address: 5383 S 900 East #104 | City: Salt Lake City | Zip Code: 84117 | Weight: 25 | Delivery Deadline: EOD | Status: Hub
Package ID: 27 | Delivery Address: 1060 Dalton Ave S | City: Salt Lake City | Zip Code: 84104 | Weight: 5 | Delivery Deadline: EOD | Status: Hub
Package ID: 28 | Delivery Address: 2835 Main St | City: Salt Lake City | Zip Code: 84115 | Weight: 7 | Delivery Deadline: EOD | Status: Delivered at 08:33
Package ID: 29 | Delivery Address: 1330 2100 S | City: Salt Lake City | Zip Code: 84106 | Weight: 2 | Delivery Deadline: 10:30 AM | Status: Delivered at 09:08
Package ID: 30 | Delivery Address: 300 State St | City: Salt Lake City | Zip Code: 84103 | Weight: 1 | Delivery Deadline: 10:30 AM | Status: Delivered at 09:26
Package ID: 31 | Delivery Address: 3365 S 900 W | City: Salt Lake City | Zip Code: 84119 | Weight: 1 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:49
Package ID: 32 | Delivery Address: 3365 S 900 W | City: Salt Lake City | Zip Code: 84119 | Weight: 1 | Delivery Deadline: EOD | Status: Delivered at 08:44
Package ID: 33 | Delivery Address: 2530 S 500 E | City: Salt Lake City | Zip Code: 84106 | Weight: 1 | Delivery Deadline: EOD | Status: Hub
Package ID: 34 | Delivery Address: 4580 S 2300 E | City: Holladay | Zip Code: 84117 | Weight: 2 | Delivery Deadline: 10:30 AM | Status: On Route
Package ID: 35 | Delivery Address: 1060 Dalton Ave S | City: Salt Lake City | Zip Code: 84104 | Weight: 88 | Delivery Deadline: EOD | Status: Hub
Package ID: 36 | Delivery Address: 2300 Parkway Blvd | City: West Valley City | Zip Code: 84119 | Weight: 88 | Delivery Deadline: EOD | Status: Delivered at 08:54
Package ID: 37 | Delivery Address: 410 S State St | City: Salt Lake City | Zip Code: 84111 | Weight: 2 | Delivery Deadline: 10:30 AM | Status: Delivered at 09:23
Package ID: 38 | Delivery Address: 410 S State St | City: Salt Lake City | Zip Code: 84111 | Weight: 9 | Delivery Deadline: EOD | Status: Delivered at 09:32
Package ID: 39 | Delivery Address: 2010 W 500 S | City: Salt Lake City | Zip Code: 84104 | Weight: 9 | Delivery Deadline: EOD | Status: Hub
Total Mileage by all trucks: 72.00 miles
-----
-----END-----
-----
```

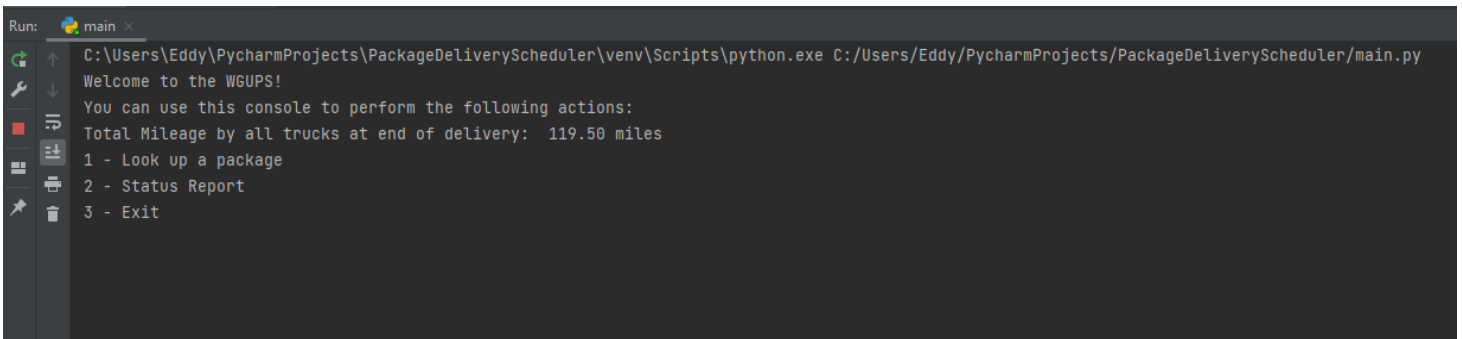
10:00 AM Report

G3. Third Status Check

```
-----
Welcome to the WGUPS!
You can use this console to perform the following actions:
1 - Look up a package
2 - Status Report
3 - Exit
>
Specify a valid business hour(8:00 - 17:00) 13
Specify a valid minute number (0-59) 00
-----
Status Report for time 13 :00
-----
Package ID: 40 | Delivery Address: 380 W 2880 S | City: Salt Lake City | Zip Code: 84115 | Weight: 45 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:37
Package ID: 1 | Delivery Address: 195 W Oakland Ave | City: Salt Lake City | Zip Code: 84115 | Weight: 21 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:40
Package ID: 2 | Delivery Address: 2530 S 500 E | City: Salt Lake City | Zip Code: 84106 | Weight: 44 | Delivery Deadline: EOD | Status: Delivered at 09:03
Package ID: 3 | Delivery Address: 233 Canyon Rd | City: Salt Lake City | Zip Code: 84103 | Weight: 2 | Delivery Deadline: EOD | Status: Delivered at 09:29
Package ID: 4 | Delivery Address: 380 W 2880 S | City: Salt Lake City | Zip Code: 84115 | Weight: 4 | Delivery Deadline: EOD | Status: Delivered at 08:44
Package ID: 5 | Delivery Address: 410 S State St | City: Salt Lake City | Zip Code: 84111 | Weight: 5 | Delivery Deadline: EOD | Status: Delivered at 09:29
Package ID: 6 | Delivery Address: 3060 Lester St | City: West Valley City | Zip Code: 84119 | Weight: 88 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:49
Package ID: 7 | Delivery Address: 1330 2100 S | City: Salt Lake City | Zip Code: 84106 | Weight: 8 | Delivery Deadline: EOD | Status: Delivered at 11:01
Package ID: 8 | Delivery Address: 300 State St | City: Salt Lake City | Zip Code: 84103 | Weight: 9 | Delivery Deadline: EOD | Status: Delivered at 11:20
Package ID: 9 | Delivery Address: 410 S State St | City: Salt Lake City | Zip Code: 84111 | Weight: 2 | Delivery Deadline: EOD | Status: Delivered at 11:16
Package ID: 10 | Delivery Address: 600 E 900 S | City: Salt Lake City | Zip Code: 84105 | Weight: 1 | Delivery Deadline: EOD | Status: Delivered at 11:10
Package ID: 11 | Delivery Address: 2600 Taylorsville Blvd | City: Salt Lake City | Zip Code: 84103 | Weight: 9 | Delivery Deadline: EOD | Status: Delivered at 12:03
Package ID: 12 | Delivery Address: 3575 W Valley Central Station bus Loop | City: West Valley City | Zip Code: 84119 | Weight: 1 | Delivery Deadline: EOD | Status: Delivered at 09:05
Package ID: 13 | Delivery Address: 2010 W 500 S | City: Salt Lake City | Zip Code: 84104 | Weight: 2 | Delivery Deadline: 10:30 AM | Status: Delivered at 09:40
Package ID: 14 | Delivery Address: 4300 S 1300 E | City: Millcreek | Zip Code: 84117 | Weight: 88 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:06
Package ID: 15 | Delivery Address: 4580 S 2300 E | City: Holladay | Zip Code: 84117 | Weight: 4 | Delivery Deadline: 9:00 AM | Status: Delivered at 08:13
Package ID: 16 | Delivery Address: 4580 S 2300 E | City: Holladay | Zip Code: 84117 | Weight: 88 | Delivery Deadline: 10:30 AM | Status: Delivered at 10:22
Package ID: 17 | Delivery Address: 3148 S 1100 W | City: Salt Lake City | Zip Code: 84119 | Weight: 2 | Delivery Deadline: EOD | Status: Delivered at 08:42
Package ID: 18 | Delivery Address: 1488 4800 S | City: Salt Lake City | Zip Code: 84123 | Weight: 6 | Delivery Deadline: EOD | Status: Delivered at 10:07
Package ID: 19 | Delivery Address: 177 W Price Ave | City: Salt Lake City | Zip Code: 84115 | Weight: 37 | Delivery Deadline: EOD | Status: Delivered at 08:31
Package ID: 20 | Delivery Address: 3595 Main St | City: Salt Lake City | Zip Code: 84115 | Weight: 37 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:29
Package ID: 21 | Delivery Address: 3595 Main St | City: Salt Lake City | Zip Code: 84115 | Weight: 3 | Delivery Deadline: EOD | Status: Delivered at 10:26
Package ID: 22 | Delivery Address: 6351 S 900 East | City: Murray | Zip Code: 84121 | Weight: 2 | Delivery Deadline: EOD | Status: Delivered at 08:12
Package ID: 23 | Delivery Address: 5100 S 2700 West | City: Salt Lake City | Zip Code: 84118 | Weight: 5 | Delivery Deadline: EOD | Status: Delivered at 12:02
Package ID: 24 | Delivery Address: 5025 State St | City: Murray | Zip Code: 84107 | Weight: 7 | Delivery Deadline: EOD | Status: Delivered at 10:34
Package ID: 25 | Delivery Address: 5383 S 900 East #104 | City: Salt Lake City | Zip Code: 84117 | Weight: 7 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:08
Package ID: 26 | Delivery Address: 5383 S 900 East #104 | City: Salt Lake City | Zip Code: 84117 | Weight: 25 | Delivery Deadline: EOD | Status: Delivered at 10:40
Package ID: 27 | Delivery Address: 1060 Dalton Ave S | City: Salt Lake City | Zip Code: 84104 | Weight: 5 | Delivery Deadline: EOD | Status: Delivered at 11:39
Package ID: 28 | Delivery Address: 2835 Main St | City: Salt Lake City | Zip Code: 84115 | Weight: 7 | Delivery Deadline: EOD | Status: Delivered at 08:33
Package ID: 29 | Delivery Address: 1330 2100 S | City: Salt Lake City | Zip Code: 84106 | Weight: 2 | Delivery Deadline: 10:30 AM | Status: Delivered at 09:08
Package ID: 30 | Delivery Address: 300 State St | City: Salt Lake City | Zip Code: 84103 | Weight: 1 | Delivery Deadline: 10:30 AM | Status: Delivered at 09:26
Package ID: 31 | Delivery Address: 3365 S 900 W | City: Salt Lake City | Zip Code: 84119 | Weight: 1 | Delivery Deadline: 10:30 AM | Status: Delivered at 08:49
Package ID: 32 | Delivery Address: 3365 S 900 W | City: Salt Lake City | Zip Code: 84119 | Weight: 1 | Delivery Deadline: EOD | Status: Delivered at 08:44
Package ID: 33 | Delivery Address: 2530 S 500 E | City: Salt Lake City | Zip Code: 84106 | Weight: 1 | Delivery Deadline: EOD | Status: Delivered at 10:56
Package ID: 34 | Delivery Address: 4580 S 2300 E | City: Holladay | Zip Code: 84117 | Weight: 2 | Delivery Deadline: 10:30 AM | Status: Delivered at 10:22
Package ID: 35 | Delivery Address: 1060 Dalton Ave S | City: Salt Lake City | Zip Code: 84104 | Weight: 88 | Delivery Deadline: EOD | Status: Delivered at 12:27
Package ID: 36 | Delivery Address: 2300 Parkway Blvd | City: West Valley City | Zip Code: 84119 | Weight: 88 | Delivery Deadline: EOD | Status: Delivered at 08:54
Package ID: 37 | Delivery Address: 410 S State St | City: Salt Lake City | Zip Code: 84111 | Weight: 2 | Delivery Deadline: 10:30 AM | Status: Delivered at 09:23
Package ID: 38 | Delivery Address: 410 S State St | City: Salt Lake City | Zip Code: 84111 | Weight: 9 | Delivery Deadline: EOD | Status: Delivered at 09:32
Package ID: 39 | Delivery Address: 2010 W 500 S | City: Salt Lake City | Zip Code: 84104 | Weight: 9 | Delivery Deadline: EOD | Status: Delivered at 11:34
Total Mileage by all trucks: 119.50 miles
-----
-----END-----
-----
```

1:00 PM Report

H. Screenshots of Code Execution



```
Run: main x
C:\Users\Eddy\PycharmProjects\PackageDeliveryScheduler\venv\Scripts\python.exe C:/Users/Eddy/PycharmProjects/PackageDeliveryScheduler/main.py
Welcome to the WGUPS!
You can use this console to perform the following actions:
Total Mileage by all trucks at end of delivery: 119.50 miles
1 - Look up a package
2 - Status Report
3 - Exit
```

I1. Strengths of Chosen Algorithm

The greedy algorithm best use case is when it can be applied to smaller instances of a problem. While WGUPS could be delivering hundreds or thousands of packages a day, the constraints of our task limit our number of packages to 40. The flow and construction of the program is easy to understand, which contributes to simplifying collaboration. Its time complexity is directly related to the number of packages, so even when a large number of packages has to be delivered, a csv file of packages with earliest deadlines can be used to prioritize delivering these packages.

There are also various simulation parameters that can be changed with ease. These can be like adding an additional driver, truck and/or package. The distances between addresses and addresses themselves can also be updated through the csv file.

I2. Verification of Algorithm

Requirement List of Algorithm

- Each truck can carry a maximum of 16 packages, and the ID number of each package is unique.

The `load_truck_packages` function loads less than or equal to 16 packages in every truck

- The trucks travel at an average speed of 18 miles per hour and have an infinite amount of gas with no need to stop.

The distance traveled by all trucks is calculated by using 18MPH as the truck speed

- There are no collisions.

The code does not account for collisions

- Three trucks and two drivers are available for deliveries. Each driver stays with the same truck as long as that truck is in service.

At any one time only two trucks are on the road during the simulation.

- Drivers leave the hub no earlier than 8:00 a.m., with the truck loaded, and can return to the hub for packages if needed.

The trucks start delivering packages at 8:00 AM

- The delivery and loading times are instantaneous, i.e., no time passes while at a delivery or when moving packages to a truck at the hub (that time is factored into the calculation of the average speed of the trucks).

The code does not add time not specified in the requirements

- There is up to one special note associated with a package.

The special notes are processed through the `load_package_data` function

- The delivery address for package #9, Third District Juvenile Court, is wrong and will be corrected at 10:20 a.m. WGUPS is aware that the address is incorrect and will be updated at 10:20 a.m. However, WGUPS does not know the correct address (410 S State St., Salt Lake City, UT 84111) until 10:20 a.m.

Package #9 is loaded on truck 3. This truck starts delivering at 10:20AM. Because by this time the addresses has been updated in the system, the csv file contains the correct address for package #9.

```
Welcome to the WGUPS!
You can use this console to perform the following actions:
Total Mileage by all trucks at end of delivery: 119.50 miles
1 - Look up a package
2 - Status Report
3 - Exit
1
Enter a package ID: 9
-----
Package 9 Report
-----
Package ID: 9
Delivery Address: 410 S State St
City: Salt Lake City
Zip Code: 84111
Weight: 2
Delivery Deadline: EOD
Status: Delivered at 11:16:00
-----
-----END-----
-----
```

- The distances provided in the WGUPS Distance Table are equal regardless of the direction traveled.

The distance looked up by reversing the indexes provided to the `distance_data` 2D list provides the same distance as if they were not reversed.

- The day ends when all 40 packages have been delivered.

In my simulation, all packages are delivered by 11:37AM

13. Algorithm Differences

The two algorithms that would meet the requirements in this scenario are Dijkstra and Brute-Force.

Dijkstra's shortest path would have required the use of a graph data structure. In this structure, nodes would represent each address, and the link connecting each node would represent the distance between each other. By finding the shortest path between all nodes, we could get a solution for the delivery path the truck should take to deliver all packages. The Dijkstra algorithm would reproduce a travel route that is more efficient than the greedy approach as above, but it would come at the cost of high space-time complexity. The Dijkstra algorithm has a higher implementation cost as well, as the greedy algorithm can be incorporated easily to solve the traveling salesman problem, it ties it an economical advantage.

The brute force algorithm to solve the traveling salesman problem would include finding every path sequence between every package uploaded into the program. There would be a total of " $n!$ " paths. Select the path that is the smallest. While this would give us the most optimal solution, it is also the most expensive as space-time complexity of the program would be $O(n!)$. This is very high in comparison to the $O(n^2)$ space-time complexity of the greedy algorithm. The greedy algorithm does not give a perfect solution, like the brute-force method, but it can be optimized to give a solution close to one we would reach with a brute-force one without all the expensive operations of it.

J. Different Approach

The first modification to the program would be the creation of a function that optimally loads packages into the trucks. The current `load_truck_packages` function loads packages on the trucks based on criteria identified outside the program. I would like to realize this step programmatically in the future. The program would also handle the packages' "special notes" and/or requirements as weighted options. These weights can then be compared to justify priority and package delivery order.

The second modification would be the integration of a database or an integration to Google Sheets. In this manner, the data can be updated externally, the simulation can be run locally, and then create a report back on the cloud. This makes it easy and fast to add or remove packages, add addresses, and add address data.

K1. Verification of Data Structure

The hash table developed for section E was built from the bottom up without using any external libraries or python's dictionary structure. It can accommodate a large number of packages to be read into the program

K1a. Efficiency

The lookup function of the program uses the hash table's search function to display a package.

This is taken from Section E Hash Table

```
search(package_id):
```

search is a function that takes in 1 argument, an id of a package and searches for it in the hashtable

Args:

package_id: This is an integer argument that represents a package id

Returns:

This function returns the package if a package is found otherwise returns none

Time complexity: Because the search function is successful by finding the package with the package_id as a key makes the time complexity is $O(1)$

Space complexity: Because the data that has to be searched through can be up to the size of the number of packages the space complexity is $O(n)$

The lookup function is independent from the number of packages uploaded to the document. Because the structure limits changing its size by assigning it a constant value at the start of the program, the time complexity of accessing a hash table and returning the key-value pair found is always constant time $O(n)$

K1b. Overhead

The data structure space usage is directly related to the number of packages to be delivered.

This is taken from Section E Hash Table

`insert(package):`

`insert` is a function that takes in 1 argument, a package object and adds/updates it to the `hashtable(self.storage)`.

Args:

`package`: This is a package object that will be inserted into the `hashtable`

Returns:

This function doesn't return data

Time complexity: Because the `insert` function calculates the hash key, uses a for loop to access a list that will always be size 2, and does value assignment, the time complexity is $O(1)$

Space complexity: Because the number of data that has to be referenced can be up to the size of the number of packages the space complexity is $O(n)$

If the number of packages is n , the space taken up by that data is also equal to n memory spaces. As more packages are added, more data is stored, therefore increasing total memory consumption. The space complexity of keeping this hash table data structure open is $O(n)$

K1c. Implications

If n is the number of packages, and m is the number of addresses to deliver to, then m can never be a number greater than n unless the user has uploaded addresses that will not be visited. In the worst case scenario for space complexity in my program, m is equal or higher than n . The `load_distance_data` creates a 2D list that can have as many as m^2 entries.

`load_distance_data(distance_data_array):`

`load_distance_data` is a function that takes in 1 argument, the list where the distance data will be represented. It utilizes the csv reader to read data from a csv file, change the data from string to a float, and add distance information into a 2D structured list.

Args:

`distance_data_array`: This is a list argument. This list should be empty and is the tool the main program will use to calculate distances

Returns:

This function doesn't return anything

Time complexity: Because the `load_distance_data` function reads through the rows m times for m times in the nested for loops, where m is the number of addresses which will always be a number equal or less than n (the number of packages) the time complexity is $O(n^2)$

Space complexity: The employment of a 2D list structure to hold the distance data utilizes m^2 memory spaces, where m (the number of addresses) is a number always less than or equal to n (the number of packages) therefore the space complexity is $O(n^2)$

If more cities were added, it translates to more addresses being added to the csv file with address data. This won't affect the look-up time of packages or distance data because only addresses found in the package file will be searched. On the other hand the space complexity of the program would increase the size of the 2D array that holds distance data, and the size of the list that holds all address data.

K2. Other Data Structures

As discussed in section I3, a different solution like Dijkstra's algorithm would require a graph data structure. The second structure that can help solve this problem is Circular linked list for it's parallels to actual addresses

K2a. Data Structure Differences

Graph Structure: This structure is composed of vertices and edges which connect to other vertices. In this example, every address would be converted to a node, and the cost of traversing an edge to get to a different node is the distance to that address. The algorithm would find the shortest path to travel to visit all nodes. The amount of memory of utilizing a graph structure is higher than a chained hash table as utilized in this example. This is because of the increased amount of data that is needed to be stored to implement this structure. In order to maximize the advantages of using a graph structure, the algorithm used would have to change as well, for example this program would change from a chaining hash table and greedy algorithm to a graph structure and Dikstras' algorithm.

Circular linked list: Every address added to the program represents a singular node. These addresses are physically connected by roads, which are represented by the link. The list being

circular and in delivery order means that after delivering the last package, the truck will end up at its starting position, the Hub. If the requirements of the total truck mileage took into consideration driving the truck back to the Hub, this would be the ideal data structure implementation for the added realism. This data structure is similar to the chaining hash table in memory consumption. In order to realize the same insert, search and remove functions from a hash table, a circular linked list would need the implementation of an indexing system. This structure can be built with the same data and data processes(data parsing) that used for the chaining hash table.

L. Sources

1. WGU. (2021, September 29). C950 WGUPS Project Implementation Steps. Western Governors University. Retrieved July 1, 2022, from <https://srm--c.na127.visual.force.com/apex/coursearticle?Id=kA03x000001DbBGCA0>