# **Algorithm Identification**

In order to efficiently assign and deliver the Packages to Trucks, I used a greedy algorithm to create a delivery sequence of shortest distance between each Package assigned.

* A list of Truck-specific assignable Packages are assigned in the order of shortest distance between each Package “**find\_nearest\_package\_in\_list()**” until no additional Packages could be assigned to the Truck.
* Once the Trucks are assigned the maximum number of Packages they can carry or all Packages have been assigned, the Trucks are ready to perform delivery.
* The Trucks then deliver all their Packages “**deliver\_all\_packages(ht, truck\_list)**” in the order they were added until they run out of Packages and return to the hub.
* If there are still unassigned Packages, the Truck loads as many Packages as possible until it reaches the maximum number of Packages assigned or until all Packages have been assigned for delivery. The Truck then performs another delivery trip (and potentially more, if necessary) until all Packages are delivered.

# **Logic Comments – Algorithm Pseudocode**

Initialize Trucks

Initialize Drivers

Delay the start time of one Truck to the earliest time of delayed Packages arriving at the depot

Assign Packages  
 For each Truck of Trucks

Until the Truck is full or no Packages are available to assign

Find and Assign the Package with the shortest distance to the last added

Package

Deliver Packages

While all Packages are not delivered

For each Truck of Trucks

Until all current Packages loaded onto the Truck is delivered

Deliver the Packages in the order they were added

Return to the Hub

Assign additional Packages

# **Development Environment**

IDE: PyCharm 2019.3.5 (Community Edition)

Build #PC-193.7288.30

Runtime version: 11.0.6+8-b520.66

Python: v.3.8

Hardware:

Processor: Intel i7 8700k CPU @ 3.70 GHz

Memory: 16.0 GB 3600 MHz DDR4

# **Space-Time and Big-O**

Space-Time complexity of major segments of the program are commented above their respective functions.

# **Scalability and Adaptability**

The initial capacity of the HashTable implementation is set for 40 Packages but if there were more than 40 Packages being inserted the solution is scalable because the HashTable would be resized with double the capacity and all Packages from the original HashTable are copied into the new resized HashTable before inserting additional Packages.

# **Software Efficiency and Maintainability**

The software is efficient because it meets all delivery requirements such as delivery deadlines, keeping mileage under 140 miles, and handling edge-cases such as Packages arriving at the depot late or Packages containing the wrong address.

The software is easy to maintain because of the amount of documentation that improves usability and allow for modifying isolated pieces of the application.

# **Self-Adjusting Data Structures**

The strength of the HashTable is being able to insert, remove, and lookup objects with the time-complexity of O(1) on average and stores a convenient table to iterate through objects at O(N) time-complexity.

A weakness of the HashTable comes in the potential need to resize the HashTable. With the table of packages initiated with a fixed initial capacity, any additional Packages inserted once the HashTable is full will require a resize that would double the previous capacity to allow further insertion. With the capacity doubling, the HashTable could be utilizing more space than it needs to if not all buckets are filled.

# **Data Structure**

A HashTable implementation was created as the self-adjusting data structure used in this solution. The HashTable holds a “package\_table” list which is used to store the Package data.

# **Explanation of Data Structure**

The HashTable is used to store Packages and allows Package retrieval via a lookup using the Package ID.

# **Strengths of the Chosen Algorithm**

The algorithm can scale with the number of Trucks, Drivers, and Packages and adapt to objects with unique properties of each object.

For example, the algorithm is scalable can support unique Trucks of different sizes which can hold onto different number of Packages.

The algorithm is able to determine the limiting constraint between the number of Trucks and Drivers to only utilize what would be practical. If there were 3 Drivers and 2 Trucks (and vice versa), the algorithm would plan, assign, and deliver Packages with taking only account of 2 delivery trucks in motion.

The algorithm can also handle delaying a Truck to start at a different time other than 9:05 AM, if a different “delayed on flight” time was indicated on a Package.

# **Verification of Algorithm**

All requirements were met and can be verified by generating a report using the interactive command-line prompt. Entering “1” when prompted for the menu options and providing “2:00 PM” as the designated report time will output a report of the Packages after they have all been delivered.

# **Other Possible Algorithms**

Depth-First Search (DFS) and Dijkstra's shortest path are two other possible algorithms that can be used for the solution.

# **Algorithm Differences**

DFS and Dijkstra’s Shortest path algorithms would find the shortest distance between two vertices in a Graph implementation, whereas the greedy algorithm takes a different approach in which it finds the shortest distance amongst all addresses to find the next nearest Package to assign to a Truck.

# **Different Approach**

If I were to approach this project again, I would separate different pieces of the Package assignment function to be performed in separate pieces in order to decrease the space and time complexity of the **assign\_packages** function. Currently it is not very efficient because each iteration of the loop requires the algorithm to assemble a list of assignable Packages to the Truck in question.

Assigning priority Packages first, then using a method to assign the rest of the “normal” Packages (no special notes, nor delivery deadline) would cut down on the space and time complexity.

# **Overhead**

As the number of Packages grow, the HashTable’s space usage would also grow.

# **Implications**

Increasing the number of Trucks would potentially allow more Trucks to be involved in performing the delivery trips if there were enough Drivers to fill the Trucks. With each additional Truck, it would linearly increase the space usage but the time complexity would not change. Since Trucks are stored in a list, the lookup time would still be O(N).

Increasing the number of cities and distances will not affect the lookup time since the data points are stored in a 2D array, accessed in O(1) time. The space complexity grows at O(N^2) since each city would need to also have a distance data point calculated against all other cities.

# **Other Data Structures**

Graph of nodes and linked list

# **Data Structure Differences**

Using a Graph of nodes, each node can represent an address in the city and the weight between each graph could represent the distance between each node. The path would be useful in representing the delivery routes of the Trucks and the weight of each edge would also be useful in representing the miles between each node. Having the nodes and edges would provide an easier and direct way to calculate the total mileage of each Truck based on the edges they have traversed. The HashTable data structure used in the solution does not contain a data point to represent the distance between each Package’s delivery address nor does it represent the delivery route of a Truck.

Linked Lists would provide a flexible data structure to store the Packages and would not potentially take up unnecessary space in comparison to the HashTable’s package\_table (in cases where the HashTable has to be resized). The order of objects linked together in the List could represent the sequence of delivery for Packages.

# **Sources**

The primary resource used for this project’s data structure implementation was the Zybooks online textbook. The section “**Linear probing**” is cited below.

Zybooks. (n.d.). Retrieved March 20, 2022, from https://learn.zybooks.com/zybook/WGUC950AY20182019/chapter/7/section/8