A: Algorithm Identification:

The algorithm I chose to use is the “nearest neighbor algorithm” because it allows for a scalable package delivery that finds the shortest distance to travel to next. When the packages are loaded onto the truck, they are manually loaded, so the deliverTrucks(truck, start) function calls other functions to get the nearest neighbor. getDistances(truck, start) is used to get the distance of all possible locations from the starting location. Then, getMinDistance(truck) is called to get the minimum distance; which is then returned to the deliverTrucks(truck, start) function to actually deliver the package.

B1:Logic Comments:

The “nearest neighbor algorithm” is used to solve the delivery problem. Shown below is a pseudo code representation of the algorithm. First, the algorithm takes two parameters, the truck number and the starting address position. Then it determines the truck the package is one, sets the count to the package list length and determines if the list length is greater than zero. If it is, then it will call a series of functions to determine the next minimum distance address. Once it does, the package will be delivered and removed from the truck. Then the function will be called again. This will iterate until the package count is zero. Once the count is zero, the truck will determine the distance back to the hub and add it to the truck’s distance list.

- Determine the truck
    - Set the package count equal to the length of the truck list
        - If package count greater than 0
            - Set new starting address
            - Call the distance function
            - Call the minimum distance function
            - Set returned values to new variables
            - Add the minimum distance to the truck minimum distance list
            - Calculate the delivery time
            - Set delivery time to datetime format
            - Add delivery time to truck delivery time list
            - Call function to get current time for delivery
            - Lookup the package being delivered in hash table
            - Remove package from truck
            - Update package in hash table with delivery time
            - Update package in hash table with status
            - Clear the truck distance list
            - Call truck delivery function again
        - If truck package not greater than 0

- Look up the distance from the start address to the hub
- Add the distance to the truck minimum distance table
- Determine delivery time to hub
- Set the delivery time to timedelta format
- Add delivery time to truck delivery time list

B2: Development Environment:

The integrated development environment (IDE) utilized for the creation of the program was PyCharm Community Edition 2022.3. The program is written in Python 3, on a laptop running Windows 11 OS.

B3: Space-Time and Big-O:

Space-Time and Big-O shown in code comments

B4: Scalability and Adaptability:

The delivery algorithm is scalable and adaptable because getDistances(truck, start), getMinDistance(truck) functions can scale to the size of the list they are initially given. Based on the packages manually loaded into the trucks, the getDistances(truck, start) function will iterate through all the packages on each truck to find the distance of each starting point to all possible end points. The getMinDistance(truck) function then works off the getDistances(truck, start) function to determine the next best location to go.

The loadTrucks(run) function is not scalable or adaptable, because the packages are manually entered into a list beforehand, then iterated through. If more packages were to be added to the delivery, then the lists would have to be manually updated before the program can correctly run.

B5:Software Efficiency and Maintainability:

The entire program is efficient because it runs at an O(n^2) time complexity. The deliverTrucks(truck, start) function calls other functions that have an O(n^2) time complexity, which makes the overall time complexity O(n^2).

The program is easily maintainable because all the major code blocks have comments of what is performed in the preceding lines of code. I also compartmentalized the code by breaking the delivery function into smaller functions. This allows for the change of single function to be updated for the entire delivery process.

B6:Self-Adjusting Data Structures:

One strength of the hash table is that it can automatically resize itself according to the input given to it in the package file. If there are more than 40 packages, then the hash table will

be able to handle those additional packages without much change to space efficiency. Another strength of the hash table is that it provides constant time for searching, insertion and deletion operations. This is beneficial because for the delivery algorithm, searching for packages is an important action that needs to be taken.

Some weaknesses are inefficiency if there are collisions and they do not support null values. In the case of this project, there are only 40 packages, so there are no real collisions. However, if there were more packages, the probability of a collision would increase if the number of buckets did not increase, which would slow down the searching process. Also regarding this project, there are no null values, but if there were, the program would break because the search function would not be able to return a key or value.

D: Data Structure:

The self-adjusting data structure used in the project is a hash table. It is able to scale to the number of packages that are read from the package_file.csv. It also has an insertion and lookup function that respectively allow for new packages to be added or existing packages to be looked up by their key. Both the insertion and lookup function can respectively store and retrieve the ID, address, city, state, zip code, deadline, weight, status, timestamp of the package.

D1: Explanation of Data Structure:

The hash table stores package information based on a key and value sequence. The key is the package ID and the values contain the remaining package information for that particular package. It gets the key by looking at the value for each row in the package_file.csv. The values are pulled from the preceding columns in that same row. Data is able to be retrieved from the hash table by using the lookup function. It takes the key as a parameter and searches through the hash table to find that same key. Then returns the values associated with that key.

The hash table is more efficient at retrieving data than a linear search because searching through a hash table has a constant space complexity.

G1: First Status Check:

- Shown in screenshots folder - labeled firststatuscheck1.jpeg and labeled firststatuscheck2.jpeg
- Ran for 9:00:00

G2: Second Status Check:

- Shown in screenshots folder - labeled secondstatuscheck1.jpeg and secondstatuscheck2.jpeg
- Ran for 10:00:00

G3: Third Status Check:

- Shown in screenshots folder - labeled thirdstatuscheck1.jpeg and thirdstatuscheck2.jpeg
- Ran for 12:30:00

H: Screenshots of Code Execution:

- Shown in screenshots folder - labeled codecompletion1.jpeg and codecompletion2.jpeg

I1: Strengths of the Chosen Algorithm:

One strength of the "nearest neighbor algorithm" used is that it is adaptable. Once the trucks are loaded, the functions called in the deliverTrucks(truck, start) function can resize based on the input they are given. The getDistances(truck, start) loops through each package given to it after the trucks are loaded and does so as long as there are packages. There is no size restraint. The getMinDistance(truck) will loop through all distances from the starting to point, which it gets after getDistances(truck, start) is called. Due to getDistances(truck, start) not having a size restraint, getMinDistance(truck) does not have one either. These allow for the delivery algorithm to scale to the number of packages given.

Another strength of the "nearest neighbor algorithm" is its simplicity. It looks through a list of packages, determines the next minimum distance from the current location, delivers the package, then repeats the process until the package list is empty. I was able to break these processes down into individual functions because they can be compartmentalized easily, due to the simple nature of their function.

I2: Verification of Algorithm:

The total mileage once all packages have been delivered is 122.6 miles. All packages were delivered on time per the specifications in the WGUPS Package File.xlsx file.

Using the interface, all parameters can be proven if the user selects option 1. Once the output is produced, all packages will show a delivered status with their appropriate delivery timestamp. The total mileage is shown at the bottom as the last printed line before the user is asked to select another option in the interface.

I3: Other possible Algorithms:

Two additional algorithms that could work to solve the package delivery problem are *Random Insertion* and *Nearest Insertion.*

I3A: Algorithm Differences:

Random Insertion:

- The random insertion algorithm starts with two locations and picks a random location that has not been visited yet to be visited next (Weru, 2021). A disadvantage to this algorithm is that it creates a random delivery path. To minimize the travel distance, this would be a hard algorithm to use, because it does not care about the distance between addresses. An advantage to this algorithm is that it could potentially create a shortest delivery route, but with it being random everytime, the likelihood is low.

Nearest Insertion:

- The nearest insertion algorithm starts with two locations then looks for the next closest location to add between the two points (Weru, 2021). A disadvantage to this algorithm is that the route that is determined may end up being longer than the nearest neighbor algorithm because it looks for the nearest insertion point to any existing location already in the route, not the nearest location to the current position. Though the path may end up being slightly longer, the time complexity of the nearest insertion algorithm is the same as the nearest neighbor algorithm, so it is equally efficient at run time.

J: Different Approach:

If I were to do the project again, I would change how the trucks are loaded and use a Greedy Algorithm. Instead of manually loading the trucks, I would iteratively go through the package list, have the program determine the minimum distance from the hub, and add the package to a respective truck. Doing this would allow the trucks to deliver the packages in the same order they were loaded.

K1: Verification of Data Structure:

The total mileage once all packages have been delivered is 122.6 miles. All packages were delivered on time per the specifications in the WGUPS Package File.xlsx file. The hash table has a lookup function called def lookup(self, key).

Using the interface, all data can be proven if the user selects option 1. Once the output is produced, all packages will show a delivered status with their appropriate delivery timestamp. The total mileage is shown at the bottom as the last printed line before the user is asked to select another option in the interface. All data is accurately represented in the output. The lookup function can be seen in the hash_table.py file.

K1A: Efficiency:

The efficiency of the lookup function for the hash table would not be greatly affected by the number of packages. If more packages were to be added to the hash table, it would take the same amount of steps to search each new package based on the key search. The function would

have to look through more packages, which could mean more iterations are required, but the time complexity would not change.

K1B: Overhead:

If more packages are added to the hash table, the space usage will increase. If more buckets were added for the storage of data, the chance of a collision would increase. However, as the number of buckets increases, so does the space required to hold them. This could potentially slow down the searching process when trying to retrieve data from the hash table.

K1C: Implications:

Adding more cities or trucks would not affect the lookup time or space usage of the hash table because the hash table does not relate to them. The hash table is strictly for the package data. All processes pertaining to the trucks and the addresses are done outside of the hash table.

K2: Other Data Structures:

Alternative data structures that could be used in place of a hash table are a *graph of nodes* and *Binary Search tree*. Both could have been used because they can store the correct data and allow for a lookup and insertion function.

K2A: Data Structure Differences:

Graph:

- An advantage of using a graph would be that it allows for better scalability with larger data because multiple data points could be grouped together in adjacent vertices. A disadvantage of a graph is the space complexity. The space complexity of a graph is O(n^2), so it is slower than a hash table.

Binary Search tree:

- An advantage of using a binary search tree is that the lookup time would decrease. This is because the packages would already be presorted. Another advantage of the binary search tree is the space complexity. A hash table has a space complexity of O(n), while a binary search tree has a space complexity of O(1). This means that the binary search tree takes up less space.

L: Sources:

Weru, L. (2021, August 24). *11 animated algorithms for the traveling salesman problem*. STEM Lounge. Retrieved January 26, 2023, from https://stemlounge.com/animated-algorithms-for-the-traveling-salesman-problem/