**Project Overview**

**Stated Problem:**

This project was undertaken to create a fully functional program to assist WGUPS with their daily local deliveries. They are experiencing problems with consistently delivering packages by their assigned deadlines. Currently, the route has three trucks but only two drivers. An average of 40 packages per day are to be delivered and each package may have specific delivery requirements. The program must allow the user to look up package status at certain times while also giving a total mileage for all deliveries. A Greedy Algorithm was used for this project because of its simplicity and usefulness when applied to problems with a small data set.

**Algorithm Overview:**

The following is a brief outline of the greedy algorithm used:

1. Packages were manually assigned to each respective truck based on the delivery criteria.
2. Using the truck’s current location, a comparison to each other location associated with each package in each truck is made.
3. The algorithm will select the closest possible location for the packages in each truck’s list.
4. The package associated with the location is then “unloaded” from the truck.
5. The process then repeats for each package on each truck’s list.

Operation time for the greedy algorithm used will be O(N^2) unless the algorithm is provided with empty lists, in which case it would have a runtime of 0(1). For this project, there will always be deliveries/packages provided in a list. See below for a pseudocode representation of the algorithm.

**Greedy Algorithm**

**If truck priority list > 0 or truck list > 0:**

**For distances in destinations:**

**Closest location = distances**

**Closest distance = distance data for distances**

**Return closest location, closest distance**

**While destination list length > 0:**

**Destination = closest location**

**Distance traveled += closest distance**

**Remove destination from destination list**

**distance.py:**

|  |  |  |  |
| --- | --- | --- | --- |
| Method Name | Line Number | Space Complexity | Time Complexity |
| none | **8** | **O(N)** | **O(N)** |

**hash.py:**

|  |  |  |  |
| --- | --- | --- | --- |
| Method Name | Line Number | Space Complexity | Time Complexity |
| \_\_init\_\_ | **2** | **O(1)** | **O(1)** |
| \_\_get\_hash | **5** | **O(1)** | **O(1)** |
| add | **11** | **O(N)** | **O(N)** |
| get | **26** | **O(N)** | **O(N)** |
| delete | **32** | **O(N)** | **O(N)** |

**package.py:**

|  |  |  |  |
| --- | --- | --- | --- |
| Method Name | Line Number | Space Complexity | Time Complexity |
| \_\_init\_\_ | **22** | **O(N)** | **O(N)** |

**prompt.py:**

|  |  |  |  |
| --- | --- | --- | --- |
| Method Name | Line Number | Space Complexity | Time Complexity |
| optionOne | **6** | **O(1)** | **O(1)** |
| optionTwo | **12** | **O(1)** | **O(1)** |
| optionThree | **18** | **O(1)** | **O(1)** |

**truck.py:**

|  |  |  |  |
| --- | --- | --- | --- |
| Method Name | Line Number | Space Complexity | Time Complexity |
| findDestination | **10** | **O(N)** | **O(N)** |
| distanceHome | **33** | **O(1)** | **O(1)** |
| individualPackageReport | **38** | **O(1)** | **O(1)** |
| packageReport | **46** | **O(N)** | **O(N)** |
| \_\_init\_\_ | **93** | **O(1)** | **O(1)** |
| truckAttributes | **98** | **O(1)** | **O(1)** |
| getDistance | **146** | **O(1)** | **O(1)** |
| loadPackages | **153** | **O(N)** | **O(N)** |
| unloadTruck | **165** | **O(N)** | **O(N)** |
| deliveryMethod | **181** | **O(N)** | **O(N)** |

**Algorithm Advantages:**

Some advantages of the greedy algorithm are its simplicity and efficiency. The provided data set is small, which minimizes the disadvantages of the greedy algorithm. Because it only works one step at a time, it can end up being less efficient than other choices, especially with larger data sets. Analyzing the run time for a greedy algorithm is also much easier than other choices because of its simplicity.

**Development Environment:**

This project was completed using Python 3.9 in the Pycharm IDE on a local workstation. Data for the project was taken from provided Excel files and converted to CSV files. The project uses only built-in Python libraries and no third-party software or tools besides the IDE.

**Scalability and Adaptability:**

Changes to the data sets would require manually updating the package lists for each truck. Because of this, it currently does not scale or adapt well to changes. It would function well with new data provided the package lists for each truck were manually assigned to find the most efficient solution. Assuming this is the case, it could adapt and scale well to larger or different data sets as the underlying methods would continue to work well.

**Software Efficiency and Maintainability:**

Because of how small the program is, maintaining it will not be difficult. Most of the components of the program are separated and can be maintained/worked on separately which should make maintenance easier. It would require manually adding packages to the trucks, but otherwise, it would need very little maintenance. The greedy algorithm used is not the most efficient solution to the problem. Considering that the trucks can only hold 16 packages and there are only three trucks at this time, efficiency should not be an issue unless WGUPS scales up considerably. Improving the hash function used or possibly using a different data structure could improve efficiency by reducing possible collisions.

**Self-Adjusting Data Structure:**

This project uses a hash table as the self-adjusting data structure. It can adjust itself to more packages without changing the hash table’s structure. The hash function could be improved which could further increase the efficiency of the program.

**Data Structure:**

The data structure used is a list, which allows the hash function to have all of the necessary functions (lookup, add, delete). Different keys were used depending on which CSV file was used to create the hash table. Because of this, constant lookup time is possible (O(1)).

**Alternate Data Structures:**

Using other data structures like graphs or deques would allow the use of different, more effective algorithms as well. Using a deque would make it easier to add and remove items. Unlike lists, a deque would give us O(1) when adding and removing items, however, lookup time could potentially be longer (O(N)) than that of a list. A graph could be used only for the distances and locations, making the distance between locations the edges and the locations themselves the vertices. The information could be stored as an adjacency matrix, which would have a higher space complexity but give us constant lookup and removal time.

**Different Approach:**

Using Dijkstra’s shortest path algorithm could be another possible solution to the problem. It could be used with the provided data as the distances between delivery locations would become the while the vertices would be the locations themselves. This solution shares a shortcoming with the greedy algorithm as it would potentially require the packages to be manually assigned to each truck before starting, especially if there were packages with specific requirements. Another possible solution would be the A\* algorithm, which is limited by its increased space complexity. Using other data structures like graphs would allow the use of different, more effective algorithms as well. However, in our situation, this shouldn’t pose an issue. Because the number of trucks and packages will remain small, and the solution could be reevaluated if WGUPS grows much larger.

**Implications:**

Because the algorithm has a time complexity of O(N), it should scale well if additional trucks or cities were added. This would be complicated by the fact that the packages need to be manually assigned to each truck before running the algorithm. The program would be unlikely to scale up to a point where space complexity would be a concern.