Core Algorithm Overview

**Stated Problem:**

The purpose of this project is to create an algorithm using a common programming language (Python3.7) to develop an effective and efficient way to deliver the package for WGUPS. This application is developed with PyCharm 2021.1.1(Community Edition), Runtime version: 11.0.10+9-b1341.41 amd64. The hardware environment is Windows 10 Pro OS 19041.1052. with Processor AMD Ryzen 7 1700X, RAM 16GB.

The total number of packages in this problem is 40. And there are three trucks, 2 drivers. Packages have some constraints and notes. This algorithm would try to find out the shortest distance to deliver all the packages as well as meet all the requirements. The method this algorithm use is to find the nearest neighbor, and this article would provide more detail in the overview.

**Algorithm Overview:**

The nearest neighbor algorithm works like following:

1. Read package data from the CSV file, and based on the requirement on each package, we put them into three different trucks. For instance, the package with the note” Must be on truck 2” goes to truck 2, the package with the early deadline goes to truck one, the package with the note “Wrong address” goes to truck 3, etc.
2. After putting the packages with special notes into trucks, we look for the packages that are not loaded but share the same address with those already in the truck. We find them and put them in the truck with those who share the same address. (We have a limit of 16 packages per truck)
3. Put the rest of the packages in truck 3.
4. Sort the package on each truck base on the nearest neighbor algorithm. Nearest algorithm: the start point is always hub, set the hub as current address, then we find the closet address on a list of the package, append it to a new list called sorted\_truck, removed that address from the original truck list, and set that address as current, repeat the process(find the closest address, append the address, set the address as current) until all of the packages is sorted.
5. With the distance and speed of the truck, we could find out the time for each package when they are delivered and find out the total miles trucks have traveled.

The space and time complexity of this algorithm is O(N+N^2+N^2+N^2) ->O(N^2):

Pseudocode:

1, initial 3 empty dictionary for 3 trucks (first\_truck, second\_truck, third\_truck)

2, go through each package to see requirements and put them into 3 different trucks:

While truck 2 is not full:

        If package notes contain “on truck 2” -> item goes to truck 2

        If package notes contain “Delay” -> item goes to truck 2

     While truck 1 is not full:

If package notes contain “must delivery with ‘x’, ‘y’ package” -> item goes to truck 1

        Put ‘x’, ‘y’ id package in truck 1

             If package notes contain “Wrong address” -> item goes to truck 3

     Check all remain package, if the package address shares same address as the package in truck 2:

        If truck 2 is not full:

The item goes to truck 2.

       Check all remain package if package address shares same address as the package in truck 2:

        If truck 1 is not full:

The item goes to truck 1.

       Check all remain package,

               If the truck is not full:

          The item goes to truck 3.

3 Sorted the package base on distance (nearest neighbor):

        Create 3 new truck named sorted\_trick1, sorted\_truck2, sorted\_truck3:

        For each truck, the start point would be the hub:

Find the closest address to the hub from all packages in truck 1.

Append that package on sorted\_truck1.

Delete a package from the original truck 1.

Make that closest address as current and find the next closest address from the remaining package on truck 1.

        Repeat the process until all packages from truck 1 are sorted.

        Repeat the same process for truck\_2 and truck\_3.

        We get three trucks with sorted order: sorted\_trick1, sorted\_truck2, sorted\_truck3.

4 deliver the packages assorted truck order and get total miles.

Space and time complexity of each part in the pseudocode:

1) O (1)

2)O(N^2) 3) O(N^2)

4)O (1)

**Delivery\_time.py:**

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Line | Space Complexity | Time Complexity |
| update\_time | 9 | O(N) | O(N^2) |
|  |  |  |  |

**Distance.py:**

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Line | Space Complexity | Time Complexity |
| sort\_package\_by\_distance | 16 | O(N) | O(N^2) |
| find\_closest | 38 | O(N) | O(N^2) |
| get\_address | 77 | O(1) | O(1) |
| get\_distance | 83 | O(1) | O(1) |

**Hashmap.py:**

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Line | Space Complexity | Time Complexity |
| \_\_init\_\_ | 9 | O(N) | O(N) |
| insert | 16 | O(1) | O(N) |
| search | 32 | O(1) | O(N) |
| remove | 43 | O(1) | O(N) |
|  |  |  |  |

**main.py:**

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Line | Space Complexity | Time Complexity |
| None | 49 | O(N) | O(N) |
|  |  |  |  |

**Package.py:**

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Line | Space Complexity | Time Complexity |
| \_\_init\_\_ | 7 | O(1) | O(1) |
| set\_delivered | 22 | O(1) | O(1) |
| set\_leave\_hub | 27 | O(1) | O(1) |
| \_\_str\_\_ | 32 | O(1) | O(1) |
| get\_address | 37 | O(1) | O(1) |

**readData.py:**

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Line | Space Complexity | Time Complexity |
| None | 8 | O(N^2) | O(N^2) |
| correction | 116 | O(1) | O(1) |
| update\_third\_truck | 126 | O(1) | O(1) |
| getHashTable | 136 | O(1) | O(1) |
| get\_second\_truck | 141 | O(1) | O(1) |
| get\_first\_truck | 147 | O(1) | O(1) |
| get\_third\_truck | 153 | O(1) | O(1) |
|  |  |  |  |

**Advantage of this algorithm:**

This algorithm is capable of handling a large number of packages, as long as we have enough trucks to load all of the packages. This algorithm could also find out the distance to the next destination for the driver. We could find out the estimated delivered time for each package, so if there is an adjustment, we could change to package order very quickly. However, if one package changes its order, it may affect other packages' estimate time.

**Efficient and maintain:**

This program is considered very efficient, it has the best time complexity of O (1) and the worst case of O(N^2). It also meets all special requirements of some packages, delivered all packages on time, and delivered the package with special instructions. The algorithm includes very detail explanation of the code, which make it easy to understand and make adjustment of the code if needed. It is very flexible to do the modification when there is a change to some element (truck capacity, total packages, additional address), it is very easy to maintain for future users.

**Other algorithms might work with this problem:**

**1 Dijkstra’s algorithm:**

This requires a graphic and connection between each node. And them find out the closet path from one node to another. With Dijkstra’s algorithm, there might be a better solution (shorter distance traveled), but it would require drawing nodes on the graphic, which would be a lot of effort if there is a large amount of address.

**2 Greedy algorithms:**

Base on the zip code of each address, sort the address into three different groups and put the same group of zip code address package at the same truck. This algorithm might work well when there is not a special requirement for each package. However, when there are constraints for some packages, the package might have to be a specific truck, which would add much complexity to the algorithm.

**How would I do differently:**

I would try to find a different algorithm to solve the problem. The code I implement is kind of messy and there are too many attributes in a dictionary as well as in the package class. As number of package increase, it would take a lot of space and time to store and adjust the information for the package. I would like to write a separate function to change these attributes (Ex. Package status, time delivered).

**Algorithm meet all requirements:**

The total miles traveled for all three trucks is 119 miles, which meets the requirement of under 140 miles. The Screenshot\_for\_G4 file shows all packages delivered on time. The algorithm I used put all the packages in the correct truck as required. Second truck leave hub at 09:05 am as flight delay. The wrong address package is modified. This project meets all requirements in the scenario.

**Data Structure:**

The data structure I used in this project include: HashMap to store package Data, dictionary to store truck information, another dictionary to store sorted truck information. This meets the requirement in the scenario, the screenshot is provided for total miles, and all other requirement is verified in the Algorithm meet all requirement section. The HashMap includes a function named search() which is used to look up any package with package ID as the search keyword.

K1A. Efficiency:

Since search() method has a time complexity of O(N), as the number of package increase, the lookup time might increase slightly. There are ten buckets in the HashMap, every 10 items added to HashMap would increase one additional look-up unit in the HashMap.

K1B. Overhead:

The data structure is using HashMap and dictionary, the changes in the number of packages would not affect the HashMap. But for the dictionary, it has a capacity of 16 (due to 16 packages per truck), the user might need an additional dictionary(truck) if all three trucks are full.

K1C. Implications:

The truck and HashMap implemented in this project are in the separate data structure, increase or decrease of the truck will not affect the look-up time. But increase or decrease would increase or decrease the space complexity, respectively. The number of cities would not affect either look-up time or space usage in the data structure.