C950 WGUPS Algorithm Overview

Jacob Sanchez

ID: 001242735

Data Structures and Algorithms II

**Stated Problem:**

The Objective of this project was to find the best route for delivering packages for the Western Governors University Parcel Service (WGUPS). The programming language used was Python (3.11). This task involved delivering 40 packages. Some packages had specific requirements, such as having the incorrect address, experienced delays, or needing to be on truck two. The way we solved this is by first, loading the truck manually based on the distance data provided. Secondly, a greedy algorithm was used to optimize the delivery of the packages. The greedy algorithm will determine the shortest route from the current location or “hub”. In this overview, I will describe the use of this algorithm and the programs methods.

**Algorithm Overview**

The greedy algorithm is carried out by:

1. Loading the packages onto the three trucks, the trucks contain the current location, truck speed, truck packages, and the time that the truck will take off for delivery. The location of the truck will always be the same but can always be updated if the hub changes in the future. This information is passed to our delivery function.
2. The location of the trucks is going to be compared/examined with the other truck packages to determine the shortest destination.
3. Once the package has been identified, the function will update the miles, truck time, departure time, and delivery time.
4. Then the package is removed from the trucks list and the truck's location is updated of the delivered package.
5. Loop will continue once all packages have been delivered and returns the mileage of the truck.

The greedy algorithm has a worse-case time complexity of O(N^2) and best-case would-be O(N). To achieve a best-case scenario, packages would need to be sorted in a way that minimizes the total distance of our trucks.

**Greedy Algorithm Pseudocode**

1. Truck Objects (Taking values as parameters)
   1. TruckAddress – The starting location for the truck
   2. TruckSpeed – The speed which the vehicle travels
   3. TruckPackages – The list of packages the truck will store
2. Deliver Packages function (Greedy Algorithm)

The function processes a list of packages for delivery along the trucks route. With each iteration, it computes the distance and delivery time to the nearest package, updates the trucks progress and package information. This will continue until all packages have been delivered. In the end it will return the total distance of the truck

**While the number of items in truck.TruckPackages is greater than 0**

**AddressNotVisited, getPackageID, getDist = minDistance(truck.TruckAddress,**  **truck.TruckPackages)**

**truck.TrackMiles = truck.TrackMiles + getDist**

**DeliveryTime = (getDist / truck.TruckSpeed) \* 60 \* 60**

**var = convertToTimeDelta(DeliveryTime)**

**truck.Time = truck.Time + var**

**packageObject = searchPackage(getPackageID)**

**packageObject.DeliveryTime = Truck.Time**

**packageObject.DepartureTime = truck.TruckTimeRemaining**

**remove getPackageID from truck.TruckPackages**

**truck.TruckAddress = addressNotVisited**

**Return truck.TrackMiles**

**Space and Time Complexities**

Table Below show the time and space complexities and their total.

Hash table class in main.py

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Space Complexity | Time Complexity | Line Number |
| \_\_init\_\_ | O (1) | O (1) | 11 |
| insert | O (1) | O (1) | 20 |
| search | O (N) | O (N) | 41 |
| remove | O (N) | O (N) | 57 |

Total Space Complexity: O(N) Time Complexity: O(N)

Class Package in main.py

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Space Complexity | Time Complexity | Line Number |
| \_\_init\_\_ | O (1) | O (1) | 75 |
| \_\_str\_\_  packageStatus | O (1)  O (1) | O (1)  O (1) | 90  96 |

**Total:** Space Complexity: O (1) Time Complexity: O (1)

Main.py methods

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Space Complexity | Time Complexity | Line Number |
| loadPackageData | O (N) | O(N) | 112 |
| distanceBetween | O (1) | O (1) | 149 |
| minDistance | O (1) | O (N) | 166 |
| truckDeliverPackages | O (1) | O (N^2) | 186 |
| loadDistanceData | O (N) | O (N) | 211 |
| loadAddressData | O (N) | O (N) | 220 |

**Total:** Space Complexity: O (1) Time Complexity: O (N^2)

Main.py

|  |  |  |  |
| --- | --- | --- | --- |
| Non-Methods | Space Complexity | Time Complexity | Line Number |
| PackageHash = ChainingHashtable()  loadPackageData('wgupspackagefile.csv', PackageHash)  distanceData = []  loadDistanceData  AddressData  TruckOnePackages  TruckOneMiles  TruckTwoPackages  TruckTwoMiles  TruckThreePackages  TruckThreeMiles  TotalTruckMiles  While True: | O (N)  O (N)  O (1)  O(N)  O (1)  O (1)  O (1)  O (1)  O (1)  O (1)  O (1)  O (1)  O (1) | O (1)  O (N)  O (1)  O(N)  O (1)  O (1)  O (1)  O (1)  O (1)  O (1)  O (1)  O (1)  O(N) | 71  138  234  238  243  249  252  257  261  266  270  274  278 |

**Total**: Space Complexity: O(N) Time Complexity: O(N)

Class Truck in main.py

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Space Complexity | Time Complexity | Line Number |
| \_\_init\_\_ | O (1) | O (1) | 75 |

**Total**: Space Complexity: O (1) Time Complexity: O (1)

The entire program has a time complexity of O (N^2). This puts less stress on resources such as CPU and RAM. This program works on a local machine so there are no network limitations. That said, this makes our program scalable.

**Advantages of Chosen Algorithm:**

The greedy algorithm chosen fulfills the project requirements to deliver packages under 140 miles. Firstly, the greedy algorithm finds feasible solutions by choosing the nearest available packages for delivery, making it efficient in time and resources with a time complexity of O (N^2). Secondly, the greedy algorithm ensures timely delivery of packages as certain packages have specific deadlines.

Another algorithm I could have used that would meet the requirements is a self-adjusting heuristic algorithm. “A self-adjusting heuristic is an algorithm that modifies a data structure based on how that data structure is used” (Zybooks, 3.1). It will essentially re-arrange the data that is used frequently making your program much more efficient as data is accessed much faster. The difference between a greedy algorithm and a self-adjusting heuristic is that greedy algorithms make a locally optimal choice whereas self-adjusting heuristics is more dynamic as it can find patterns and adapt over time. A second algorithm that would meet the requirements is Dijkstra's algorithm. “Dijkstra's shortest path algorithm, created by Edsger Dijkstra, determines the shortest path from a start vertex to each vertex in a graph” (Zybooks, 6.11). A key difference between Dijkstra’s algorithm and greedy algorithm is that Dijkstra’s is specifically used to find the shortest path between vertices. This would allow our trucks to deliver packages based on the shortest distance to take.

**Programming Environment:**

This program was built for local environments such as Windows, Mac, etc. The program was written in Python 3.11 and was processed using PyCharm IDE. There are zero communication protocols as the information needed for this program comes from CSV data, located in the project folder. Our data is read in our code by using a csv reader function which then stores this information in a Python list. This program is limited to the host machine only and does not require internet connectivity.

**Adaptability:**

The program overall is highly adaptable and scalable to the number of growing packages. We can achieve this because we implement a hash table. For example, if we increase the number of packages, the hash table can assign packages to their specific buckets and utilizes chaining to prevent collisions. For delivery, the greedy algorithm is also not dependent on a fixed number of packages, which makes it dynamic to the growing number of packages, while keeping the time complexity mostly the same.

**Efficiency and Maintainability:**

My program is efficient as my core algorithm has a time complexity O(N^2). It can scale with the number of growing packages in each truck easily. It is also maintainable as I have comments, structure, descriptive functions, and variables which makes it easier to update code in the future.

**Data Structures:**

The self-adjusting data structures used in this program were a hash table and lists. These data structures are easy to implement and are quite flexible. I was also able to incorporate a hash table for one of my lists for search, deletion, and insertion. The time complexity for most of my hash table operations was O (1). My hash table was specifically used for package data and utilized chaining, where each bucket in the table has a linked list to store key-value items. There is an initial capacity set, however, collisions are avoided because multiple keys hash to the same bucket in the table. The hash table is organized as a list of buckets, where each bucket contains a key-value pair. Collisions are managed using the chaining method, if there is already a key-value pair in the bucket, the pair will be added to the linked-list for that bucket. Adding additional trucks or cities would not affect the lookup time because adding additional trucks or cities will still yield a space and time complexity of O (N).

Now that I have completed my project, I would like to do different is have used a dictionary, which would map key values pairs, for example, each package ID could be used as a key for the associated package object. This would be a more semantic choice than a list. Secondly, an array could have been used instead of a list, the advantage would be that there is less overhead in memory, but an array is typically fixed in size.

**Sources:**

Zybooks. (n.d.). <https://learn.zybooks.com/zybook/WGUC950AY20182019/chapter/6/section/11>

Zybooks. (n.d.). <https://learn.zybooks.com/zybook/WGUC950AY20182019/chapter/3/section/1>