Drex Beckman

**#001082313**

C950

C950 Overview Document

1. **Algorithm Identification**

The self adjusting algorithm I used for this project was implemented in the distances\_and\_deliveries file called “deliver()”. The deliver() method is an example of a nearest neighbor algorithm. It takes 3 parameters, truck\_list, a list type, truck\_start\_time, a datetime, and returns\_to\_hub, a boolean. The method utilizes the nearest neighbor algorithm by parsing the addresses of the respective packages within the truck’s list, comparing it to the last delivered address, or the hub in the case of the first package, and finds the address which is the shortest distance from the previous address. When a package has been selected by the algorithm, its delivery status is updated within the hash table, the mileage and time taken are both incremented within their respective variables, and finally the package is removed from the truck’s list.

**B1. Logic Comments**

The pseudocode is as follows:

while size of packages is not 0:

set min\_distance to 0

set current\_package\_id to 0

set previous\_package\_id to 0

set from\_address to "hub address string"

for every package in packages:

set current\_package\_id to package ID

get mileage from distance table between from\_address and package address and set to distance

if distance less than min\_distance:

set min\_distance to distance

set previous\_package\_id to current\_package\_id

set from\_address to package address

add min\_distance to global total\_distance\_travelled

set truck\_seconds to (min\_distance / 18) \* 3600

add truck\_seconds to truck\_start\_time

update package status as delivered at truck\_start\_time

remove package from packages

**B2. Description of programming Environment**

The program environment used was PyCharm Community Edition 2021.2 with the Python 3.9 interpreter. The operating system used was macOS Catalina 10.15.7 on a 1.4 GHz dual-core Intel i5 processor. The program was running on a 2014 Macbook Air.

**B3. Space-Time and Big-O**

1. distances\_and\_deliveries.distance\_between(address\_1, address\_2): O(1)
2. distances\_and\_deliveries.shortest\_distance(from\_address, truck\_list): O(N)
3. distances\_and\_deliveries.deliver(truck\_list, truck\_start\_time, returns\_to\_hub): O(N \* M) deliver method calls shortest\_distance(), which loops through truck\_list, within a while loop iterating through len(truck\_list)
4. distances\_and\_deliveries.deliver\_packages(): O(1)
5. hash\_table.add\_item(self, key, item): O(n)
6. hash\_table.remove\_item(self, item\_key): O(1)
7. hash\_table.search\_item(self, item\_key): O(n)

The overall complexity of these major methods evaluates to O(N \* M)

**B4. Scalability and Adaptability**

Because the nearest neighbor algorithm is capable of evaluating the shortest distance between a given package address and some address within the set of packages in a truck, the ability to optimize the mileage for a truck is easily scalable. A drawback would be that because the packages are assigned manually, using heuristics like special delivery instructions, deadline, and late delivery implies an algorithm to better determine which packages go to which truck would need to be developed. However, since the packages are assigned to a given truck via a list of package ID numbers, the implementation of an algorithm to assign them could be created, assigning the optimal package IDs to a list, and assigning them that way. This would mean the algorithm could be simply appended above the segment of the CSV\_reader file where the package ID lists are assigned to a truck. And because the deliver() method accepts a boolean which determines whether or not the truck returns to the hub, trucks could easily be utilized to run several deliveries in a day easily.

**B5. Software Efficiency and Maintainability**

The hash table uses one method for adding and updating a package’s contents, which makes its use more intuitive. The total distance travelled can be calculated for an increasing load of packages and trucks. Because the program was developed first with regard to basic functions, such as the ability of a truck to deliver packages and then determine the distance and time taken before solutions concerning optimization were implemented, this means that the ability of another programmer to implement more efficient or scalable solutions would not entail having to modify any of the fundamental mechanics of package assignment, distance and time calculations, or use of the hash table. That is to say that because the basic utilities were kept developmentally separated from the optimization portion, it is easier to come up with better, more scalable methods of delivery after the fact.

**B6. Self-adjusting Data Structures**

A previously mentioned strength of the hash table was that the method to add and to modify a package are the same rather than two different methods. This makes it more intuitive to use. Perhaps one drawback of the hash table is the fact that it utilizes chaining to overcome collisions rather than open addressing. A known disadvantage of using chaining in a hash table is that each bucket may contain a list of objects, which would need to rely on a searching algorithm to find the correct item the user is looking for. Such an approach would have a space-time complexity that increases linearly with respect to the number of objects in the bucket. This is contrasted with the open addressing solution for assigning objects in a hash table, where the nearest bucket to the bucket which would have been assigned, but was previously occupied by another object, is given to the current object. The choice of using chaining was one of pragmatism. The implementation of open addressing not only further limits scalability directly to the number of buckets in the hash table, but also requires more work to make. A tangential strength of the program also concerning the hash table is the use of a package class, which means that modifying, adding, or removing package data within the hash table does not require remembering which element in a list needs to be modified, rather all that is required is searching for the package, creating an instance of the package class, modifying that package, and then updating the hash table.

**D. Data Structure**

The data structure used for the project was a hash table containing 10 buckets using chaining as a way of handling collisions.

**D1. Explanation of Data Structure**

A package object has a package ID, a truck ID, an address ID, address information, package weight, delivery instructions, special notes, and most importantly for the deliver() method, a delivery status. The hash table stores this information in package objects, and uses their respective package IDs as keys. When deliver() needs to find the distance between two delivery addresses within a truck’s package list, it only needs the package ID, since the data is stored both within the truck’s list, and the hash table. Once the delivery address that is the shortest distance from the current address is found, the time to drive from one point to the next is calculated as a timedelta, and added to the truck’s start time. From there, the package is removed from the truck’s package list, but it must have its status updated in the hash table. Because the package ID was stored in the truck’s package list, the ID is used as the key to retrieve the correct package in the hash table. A package object is assigned the data corresponding to the package in the hash table using the search\_item(package\_id) method. The package object’s status is updated with a setter function, marking it as delivered, along with the truck’s start time plus the timedelta. The hash table is modified with the add\_item(id, package) function, which updates the delivery status of the package.

**I1. Strengths of Chosen Algorithm**

The first advantage of utilizing the nearest neighbor algorithm in this scenario is that the concept behind it was simple to understand. This means if mileage is too high or delivery took too long, troubleshooting consists only in rearranging the packages. A second benefit is that the space-time complexity of a nearest neighbor algorithm would be quite low compared to other methods of efficiently delivering all the packages.

**I2. Satisfying the Scenario Requirements with the deliver() method**

Identifying the requirements in the scenario, it can be seen that:

* There are an average of 40 packages per day
* Packages have special instructions
* There are three trucks, and two drivers
* There are several packages with deadlines
* Mileage should be kept below 140 miles once all packages have been delivered

The algorithm implemented does satisfy all requirements listed above. All 40 packages are delivered by 12:07, with a total distance travelled by all trucks of 119.3 miles. The package instructions were taken into account when they were manually loaded in each truck. Truck two returns to the hub at the end of the day, and truck one is able to make a second round of deliveries for the remaining eight packages. The deadlines for all packages were met, and the packages which were delayed were loaded into the appropriate truck to accommodate their late arrival to the hub. Proof of the constraints are provided with the included screenshots.

**I3. Other Possible Algorithms**

Dijkstra’s algorithm is the first alternative self-adjusting algorithm which could have been used instead of the nearest neighbor algorithm. An example of another self-adjusting algorithm which could be applied to the travelling salesman problem would be the regular greedy algorithm, of which the nearest neighbor is a subset.

**I3a. Algorithm Differences**

In Dijkstra’s algorithm, instead of simply comparing a set of nodes to find the next closest, a set of unvisited nodes is created, and the distance between vertices in a weighted graph are compared to find the shortest path from the current node to the next. This process is repeated until all nodes have been visited. The difference between nearest neighbor and greedy is that the greedy algorithm does not need to connect one vertex to the next in a linear fashion. Using the distance table, the addresses within a truck’s package list would be compared by their distance to one another, the shortest pairs connecting together, and linking those together to form a route. The main disadvantage being, even with a good package assigning process for the trucks, the search for all possible distances between addresses in the truck list would be resource intensive.Therefore, it is determined that both algorithms meet all the requirements.

**J. Different Approach**

The main issue the project has is the assignment of packages to their respective trucks. As the amount of packages increases, the ability to adequately sort the packages into truck loads will become more complicated, especially if the postal service’s average package load were to increase drastically from 40. I would try to better account for this foreseeable issue by creating an automated heuristic method of sorting packages. The main problem I had was that certain package constraints had different weights as to their importance. The priority always being special instructions, then delays, then deadlines, and finally distance. Because the algorithm I used just looks at distance, and there is no method of automatically assigning packages to trucks, I was able to create a set of truck routes that worked, but it would be too tedious to maintain if the load of packages was too high.

**K1. Data Structure Adherence to Scenario Requirements**

Front the scenario of the project, the hash table has to:

* Retrieve package information given package ID
* Allow for addition of new packages
* Update package data, especially delivery status
* Remove packages from itself

Because the hash table has add, search, remove, and update functions, it satisfies the scenario requirements. Also, because the hash table uses chaining, this ensures that packages are able to be added to the table easily without collisions.

**K1a. Efficiency**

Because the solution uses chaining to overcome the problem of data collision, the time taken to search for a package using a ket will increase linearly, proportionally to the number of packages added. The same can be said for the amount of space used.

**K1b. Overhead**

As the number of packages in the hash table increases the number of packages within a bucket will also increase. This means storage space likewise increases linearly.

**K1c. Implications**

Because the hash table only stores package objects, and those package objects contain city and truck information, there would be no additional space-time complexity, other than that which was previously mentioned, as the number of trucks or cities increased.

**K2. Other Data Structures that Would Meet Scenario Requirements**

A set could have been used for this Scenario, because it allows for searching for an object through a key, allows for updating of an object, and does not allow for duplicate entities. Another potential data structure that would be well suited for the project would be a dictionary.

**K2A. Data Structure Differences**

The difference between a set and a hash table is that there is no need for collision handling, and you can assign subsets to a set. A dictionary, unlike a hash table, uses key-value pairs. A key is unique, meaning that there does not need to be any hashing of the key, nor does there need to be any collision handling.

**L. Resources**

All information was taken from school provided resources. No citations needed.