**NHP2 — NHP2 TASK 1: WGUPS ROUTING PROGRAM**

**Brandon Jones**

**A: ALGORITHM IDENTIFICATION**

The algorithm used to deliver all packages is an adaption of the popular “nearest neighbor” algorithm that functions by calculating the nearest point from each package.

**B1: LOGIC COMMENTS**

The packages are loaded onto each truck individually on each truck and once the trucks are loaded, the “runRoute” function is ran to deliver the current truck load. These steps are repeated until every package is delivered

The logic behind is as follows:

function runRoute(truck):

firstLocation = HUB

nextDistance = 20

nextPackage = 0

while there are packages on truck

for each package on truck:

calculate distance between firstLocation and package address

if distance between firstLocation and package address is less than nextDistance

nextDistance = distance between firstLocation and package address

nextPackage = current package in the iteration of this for loop

calculate total distance driven by truck using nextDistance

calculate delivery time by using nextDistance and avg truck speed of 18 mph

set firstLocation = nextPackage

remove nextPackage from truck

**B2: DEVELOPMENT ENVIRONMENT**

This program was developed on 64-bit Windows 10 machine with a Coffee Lake architecture Intel processor. The IDE used was PyCharm Community Edition 2021.3

**B3: SPACE-TIME AND BIG-O**

The first major segment of this program reads in data from the package list csv and instantiates our package class with said information then inserts the package along with a natural index into a hash table. The time complexity for this segment is O(n) where n is each a package.

The next segment of the program creates a distance array and an address array from our distance csv. The complexity for each of these operations are O(n) where n is each line being appended to an array.

The most intensive segment of this program is our runRoute function. Since there is a while loop that will execute for every package on each truck, the time complexity for that would be O(n) where n is a package on the truck. Inside of this while loop is a for loop that executes for each package on the truck as well, so together the time complexity of this function would be O(n2).

Adding each package to the truck would be considered constant time, however, once the packages are loaded onto the truck, a for loop executes for each package on every truck and changes the delivery status to “en route” so this would be considered O(n) time.

Finally, the segments that print status reports loop through the entire hash table so this is O(n) time where n is a package in the hash table.

**B4: SCALABILITY AND ADAPTABILITY**

Since the hash table implemented in this program uses natural indexes to hash, this solution could work for any number of packages so long the package IDs were unique. No changes would have to be made to the route function to accommodate more packages.

**B5: SOFTWARE EFFICIENCY AND MAINTAINABILITY**

Since the time complexity of this program is O(n2), it is significantly more efficient with lesser amounts of packages.

Maintenance for this program is minimal, as the only thing that needs to change with new sets of packages is the order of which packages are to be loaded on which trucks.

**B6: SELF-ADJUSTING DATA STRUCTURES**

Hash tables can be tricky to implement as you need a hash function and collision handling functions put into place to store and retrieve data, if your collision handlers are not efficient, your data structure could become very inefficient. However, because of these functions we are able to store and retrieve data much more efficiently than can be expected from a normal array.

**D: DATA STRUCTURE**

This program uses a hash table in order to store the package data and interact with the algorithm that delivers the packages.

**D1: EXPLANATION OF DATA STRUCTURE**

The hash table developed in this program contains an insertion, deletion, and search function that allows the user to manipulate the data stored in the table. Since the package information that is being stored in the hash table contains a natural index (package ID), I was able to make the hash table more efficient by omitting a hash function.

**G: STATUS REPORTS OF THE PACKAGES**

1. **Between 8:35am and 9:25am**

**Text

Description automatically generated**

1. **Between 9:35am and 10:25am**

**Text

Description automatically generated**

1. **Between 12:03pm and 1:12pm**

**Text

Description automatically generated**

**L1: STRENGTHS OF CHOSEN ALGORITHM**

A major strength of the nearest neighbor algorithm developed for this program is it’s simplicity. Since it is easy to understand, implementation is simple and maintenance for the program can be easily performed by any developer. Another advantage of this algorithm is its efficiency. It determines a relatively efficient path very quickly, while it is rarely the most optimal path.

**L2: STRENGTHS OF CHOSEN ALGORITHM**

The algorithm can be verified by looking at the third status check screenshot in part G. All packages were delivered on time to the correct address.

**L3: OTHER POSSIBLE ALGORITHMS**

I considered using an algorithm similar to the one I implemented, however, it would have incorporated the idea of dynamic programming as the function would calculate locations in groups of 3. Rather than computing the next location from a current location, this algorithm would take the distance summation of 3 separate locations and compare that to all other possible set of three locations and repeat that until all packages are delivered. Hypothetically, this could produce a more optimal path, however it would be much less time efficient.

Another algorithm that could solve this problem would be a heuristic algorithm. Specifically a self-adjusting heuristic algorithm. This algorithm could prove to be useful by organizing the packages in a way that minimizes the time to search for each package. It’s not likely that this algorithm would result in a more optimal path than my chosen algorithm, however, it is likely to reduce overall time complexity of the program.

**J.  Describe what you would do differently, other than the two algorithms identified in I3, if you did this project again.**

I would implement an interface class full of functions to handle time, converting time, adding time, calculating time for each delivery. Having an interface like this implemented would have made status checks much simpler.

**K.  Justify the data structure you identified in part D:**

The hash table that I implemented in this program meets the requirements because it stores all of the package information and completes the search, insert, and delete, functions properly. The hash table functions in liner time, O(n), and uses space at the rate of O(n). This means that the time needed to complete the table’s functions as well as the space it requires is directly proportional with the amount of packages stored in the table grows linearly with the amount of packages. Increasing the amount of trucks or cities used in this problem would not affect the time or space of this data structure, unless you were also increasing the amount of packages to handle.

Two other data structures that could have handled this problem are: Dictionary and 2-dimensional array. A dictionary functions very similarly to the hash table that was implemented in this program, although it would have likely used a hash function which could have possibly made the program less efficient than using the direct hash table. A 2-dimensional array could have been implemented to store the packages, however, look up times would have been significantly longer, as you would need to loop through the entire array, and the program would run less efficiently.