**C950 WGUPS Routing Program**

Prepared by

Arthur J. Amende

Student ID: 002180137

# A: Algorithm Identification

The program was created using a Nearest Neighbor algorithm which selects the next closest delivery address of packages on a truck.

# B1: Logic Comments

Packages are manually sorted and loaded onto the trucks. The packages were manually sorted due to the high number of packages affected by special instructions, 15 out of 40 packages.

Each truck determines its path in the following manner:

if status of truck is driving:

drive truck along current route

if truck arrives at destination

deliver packages (take off truck)

find next closest delivery address from packages remaining on truck

if truck has delivered all packages

return to HUB for more packages

# B2: Development Environment

Python 3.10.0

IDE: Visual Studio Code

Home Personal Computer:

* Processor: AMD FX(tm)-8350 Eight-Core Processor 4.00 GHz
* Installed RAM: 8.00 GB
* Windows 10 Version 21H1 for x64-based Systems (KB5006738)

# B3: Space-Time and Big-O

Nearest Neighbor Algorithm:

Single Address Search: O(N)

Delivery Route: O(N^2)

Each time a truck needs to determine its next destination requires it search through all packages still loaded on the truck.

Package Hashing Complexity:

Single Insert O(1)

Single Search O(1)

The package hash map has sufficient storage for all packages to be placed in a single ‘bucket’. No collisions will occur.

Distance Hashing:

Single Insert O(1) to O(N)

Single Search O(1)

The distance hash map has a total of 100 buckets by default and there are 351 distance values to store. Collisions will occur increasing the time complexity. Complexity may rise to O(N) for a single insert or search. But spread of data through the hash map is sufficiently wide time complexity remains low. Using the built in Python hash() function, which is randomly seeded, the average packages in a used bucket was 4.6.

# B4: Scalability and Adaptability

The program is limited to scalability and adaptability mainly because the packages are hand sorted for each truck. The program does not determine which packages should be loaded on which truck. Currently the program is limited to only three delivery cycles. But the hashing and path finding portions of the program are easily scaled.

The number of packages stored in the package hash map can be scaled up without any changes to the code. Time complexity will increase as more collisions occur. Collisions can be minimized by increasing the size of the hash map.

Similarly the number of address/distance pairs can be increased without any changes to the code. Time complexity will increase as more collisions occur. Collisions can be minimized by increasing the size of the hash map.

The number of packages on a truck can be increased with no change to the code. The Nearest Neighbor algorithm that determines the next delivery address for the truck is independent of package size. But time complexity will increase with the number of packages on a truck with no mitigation possible.

# B5: Software Efficiency and Maintainability

The program is efficient and maintainable due to its simplicity. A Nearest Neighbor algorithm is easy to implement and understand. While the program does require packages to be hand loaded that is the only user input required. Once done the program quickly determines the routes of each truck without further user input.

The program is easily maintained because it is logically divided, commented, and relatively simple.

# B6: Self-Adjusting Data Structures

The greatest strength of the Nearest Neighbor algorithm is it will find a solution given enough time. By simply finding the next closest delivery address each truck will eventually delivery all packages and return to the HUB. Additionally, a Nearest Neighbor algorithm is very simple to implement and understand.

The largest draw back of a Nearest Neighbor algorithm is it very unlikely to determine an optimal solution. The route each truck travels is not likely to be the most efficient. Also, it is guaranteed to find a solution *given enough time*. Potentially the amount of time could be unacceptable.

Hash mapping used for the packages and address/distance pairs is highly scalable and efficient. Given a sufficiently large hash map there would be no collisions when inserting or searching for data leading to a fixed time complexity. But a large hash map may also have large empty sections. Reducing the size of the hash map will minimize empty buckets but increase collisions resulting in increased time complexity.

# D: Data Structure

The program creates a Package Class which stores all the information of a single Package including:

* Id Number
* Delivery Address
* Delivery City
* Delivery Zip Code
* Package Weight (kg)
* Delivery Deadline Time
* Package Status
* Delivery Time

A Package hashing data structure is created in the PackageHash class. The package ID number is used as the key for the hash table. Collisions are resolved by chaining.

# D1: Explanation of Data Structure

A Package hashing data structure is created in the PackageHash class. Upon initialization a table is created with a user passed number of ‘buckets’. If the user does not pass a number for buckets it defaults to 10.

The insert method of the PackageHash class uses the ID number of the package as the key for the hash table. The key MOD number of buckets determines what bucket the package will be put in. The hash table accounts for collisions by chaining packages in the same bucket. The insert method will first search if the package already exists in the table, and if so, override the previous data with the new package data. If the package does not exist it is added to the hash table.

The search method of the PackageHash class accepts either the Package ID number or the Package instance. The package ID number is used as the key to determine what bucket the package should be in. That bucket is searched for the package and the package returned if found or None if not found.

# I1: Strengths of the Chosen Algorithm

Two major advantages of the Nearest Neighbor algorithm that apply to this problem are:

* Finding the next closest address will always result in a solution
* Simple to implement

# I3: Other Possible Algorithms

Other algorithms that could be used to solve the problem include:

* Greedy Algorithm
* Dijkstra Algorithm

# I3A: Algorithm Differences

The Greedy Algorithm finds the best solution at the current moment in time. In a travelling salesman problem it would find the next delivery address that can be driven to in the shortest amount of time. A Greedy Algorithm is also simple to implement and easy to understand. In this scenario as speed is constant, there are no collisions, there is no traffic, and the distance between two points is always constant regardless of direction travelled, the Greedy Algorithm and Nearest Neighbor Algorithm are functionally identical. If any other variable could change over time then the use of a Greedy Algorithm would be preferred to account for additional variations.

The Dijkstra Algorithm finds the shortest path from a start vertex to each vertex in a graph. If we think of each delivery address as a vertex and each route as a path between vertexes we could use the Dijkstra Algorithm to find the shortest route to visit each delivery address. At each vertex (delivery address) the algorithm would weigh the distance to each adjacent vertex choosing the shortest distance to travel. Again, because our given situation is greatly simplified with no variables other than distance between delivery addresses the Dijkstra algorithm would also simplify to a Nearest Neighbor algorithm.

# J: Different Approach

If I were to attempt this project again I would create a second hash table to store packages by address instead of package ID number. I could then use the address hash table to write an algorithm to determine which packages should go on which truck to optimize package delivery. Of course the packages would still require some manual sorting to account for special delivery notes.

# K1A: Efficiency

The number of packages stored in the package hash map can be scaled up without any changes to the code. Time complexity will increase in hashing the packages, both insert and search, as more collisions occur. Collisions can be minimized by increasing the size of the hash map.

The time complexity for the Nearest Neighbor algorithm will not increase with the total number of packages to be delivered. The algorithm’s time complexity is dependent on the number of packages on a truck, not the total number of packages to be delivered.

# K1B: Overhead

To keep time complexity at a reasonable level will require increased overhead space usage. As the number of packages grow collisions will become more common in the hash table. The length of a chain in a given bucket will become longer increasing time complexity. Expanding the size of the hash table to include more buckets will reduce collisions and chaining thus reducing time complexity.

# K1C: Implications

An increase in the number of trucks would have the largest impact on the efficiency of the program because packages are manually entered for each truck. User sorting and input will always be much slower than an algorithm. But look-up time and space usage of the hash table would not be affected. The hash table is not dependent on the number of trucks or the location of a delivery address.

# K2: Other Data Structures

Other data structures that could be used in lieu of a hash table include:

* Ordered Array
* Set

# K2A: Data Structure Differences

An ordered array could be used to store packages. The packages would be sorted by their id number. A binary search could then be performed to insert and search for packages. The time complexity would vary from O(1) to O(log n) for each insert or search. A hash table with even a small number of buckets would be able to easily out perform an ordered array and binary search.

A set of unordered package data could also be used. It would have a time complexity of O(1) for each insert assuming you never tried to enter duplicate packages. But would have time complexity of O(1) to O(N) for any search with no way to optimize. We effectively use an unordered set in the hash table to account for collisions. Any collision in a bucket generates a set that must be searched through. Using an unordered set would be effectively the same as using a hash table with a single bucket.