C950 - TASK 1: WGUPS ROUTING PROGRAM PLANNING

Fatma Sena Ekiz – Student ID: 011013854

**A. Algorithm for the WGUPS Routing Program**

The algorithm that could be effectively used for the WGUPS routing program is the **Nearest Neighbor Algorithm (NNA)**. This algorithm is beneficial in scenarios where the goal is to find a close-to-optimal path through multiple points (delivery locations in this case) by always moving to the nearest unvisited point from the current location. Given the constraints and requirements of this task, the NNA offers a straightforward approach to reducing the total distance traveled while ensuring all packages are delivered.

**B. Data Structure for the Routing Program**

A **hash table** would be an effective self-adjusting data structure to use alongside the NNA for managing package data.

**1. Relationship Accounting in the Data Structure**

* **Keys**: Each package's ID can serve as a unique key.
* **Values**: Associated values could include package-specific details such as delivery address, delivery deadline, delivery status, and any special notes.
* The hash table facilitates quick access (constant time complexity on average) to any package's data, which is crucial for real-time updates on delivery status and handling exceptions like the change in delivery address for package #9.

**C. Overview of the Program**

**1. Algorithm's Logic in Pseudocode**

initialize hash\_table to store package data

initialize hash\_table to address data

initialize hash\_table to store distance data

load\_packages\_into\_package\_hash\_table()

load\_address\_into\_address\_hash\_table()

load\_distance\_into\_distance\_hash\_table()

for each truck:

while truck is not empty or packages remain undelivered:

current\_location <- hub

nearest\_package=find\_nearest\_unassigned\_package()

load\_truck\_with\_nearest\_packages() # adhering to max capacity and special notes

while truck has packages:

nearest\_location <- find\_nearest\_neighbor(current\_location)

deliver\_package(nearest\_location)

update\_package\_status\_in\_hash\_table(nearest\_location)

current\_location <- nearest\_location

return\_to\_hub()

def find\_nearest\_neighbor(current\_location):

min\_distance <- infinity

for package in truck:

distance <- distance\_hash\_table.get (current\_location, package.location)

if distance < min\_distance:

min\_distance <- distance

nearest\_package <- package

return nearest\_package.location

**Pseudocode Explanation:**

**Initialization**:

* + Initialize hash tables to store package, address and distance data.
  + Load all those data (reading from CSVs) into the hash table.

**Truck Delivery Loop**:

* + For each truck:
    - While the truck is not empty or there are undelivered packages:
      * Set the current location of the truck to the hub.
      * Load the truck with packages, considering its maximum capacity and any special notes.
      * While the truck has packages:
        + Find the nearest neighbor location to the current truck location.
        + Deliver the package to the nearest location.
        + Update the package status in the hash table.
        + Update the current truck location to the nearest location.
      * Return the truck to the hub once all packages are delivered or the truck is empty.

**Finding Nearest Neighbor**:

* + For each package in the truck, calculate the distance to each location.
  + Find the location with the minimum distance from the current truck location.

**2. Programming Environment**

* **Software**: Python 3.9+, Visual Studio Code to edit and run the code.
* **Hardware**: Any standard computer with sufficient RAM (at least 8GB) and processing power to handle script execution and potential data visualization processes. I’m using a MacOS Sonoma 14.3.

**3. Space-Time Complexity**

**Space Complexity:**

The space complexity refers to the amount of memory space required by the algorithm as a function of the input size (number of packages, denoted as N).

* **Best Case**: O(N)
  + The space complexity is best when the hash table has enough space to store all package data without any collisions, resulting in a linear space requirement proportional to the number of packages.
* **Worst Case**: O(N)
  + The worst-case scenario also occurs when the hash table has to handle N packages, but with potential collisions, leading to additional space required for handling collisions and maintaining the integrity of the hash table.

**Time Complexity:**

The time complexity refers to the amount of time taken by the algorithm to complete its execution as a function of the input size (number of packages, denoted as N).

* **Best Case**: O(N^2)
  + Each package's data is stored in the hash table, all hash table operations (lookup, insert, delete, update) have O(1) time complexity at best and O(N) at worst if hash collisions occur.
  + The best-case scenario occurs when the nearest neighbor algorithm consistently selects the nearest location in each iteration, resulting in N iterations to process all packages. However, within each iteration of while loop, finding the nearest neighbor requires checking the distance to each package, resulting in a time complexity of O(N) for each iteration. Therefore, the overall time complexity becomes O(N^2).
* **Worst Case**: O(N^2)
  + The worst-case scenario also involves O(N) iterations to process all packages, but with additional time required within each iteration due to potential hash collisions and the need to calculate distances to all other packages. Therefore, the overall the time complexity remains O(N^2).

**4. Scalability and Adaptability**

* The program can scale to handle more packages or additional routes by adding more trucks and drivers as the core nearest neighbor algorithm and data handling would not change significantly.
* Adaptable to different cities and states, assuming distance tables and maps are updated accordingly.

**5. Software Design Efficiency and Maintainability**

* Modular design: Separating functionalities like loading trucks, finding the nearest neighbor, and updating the delivery status into distinct functions increases maintainability. Also having separate classes for Package, Truck and HashTable operations increases efficiency and maintainability by applying abstraction and object-oriented design.

**6. Strengths and Weaknesses of the Hash Table**

* **Strengths**: Fast access times for reading and updating package data; well-suited for scenarios with frequent look-ups.
* **Weaknesses**: Poor performance in scenarios with many hash collisions; space efficiency can be an issue if the load factor is not well managed.

**7. Key Choice for Efficient Delivery Management**

* **Package ID** as a key is most effective since it remains unique per package, allowing for efficient indexing and retrieval in the hash table. This is essential for managing real-time updates on the status and location of each package during delivery operations.

**8. Sources**

* No external sources used.