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**

1. **Initialization Segment**

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()

**• Space Complexity:** Assuming we have N packages, M addresses, and L distances:

**• Packages hash table:** *O(N)*

**• Addresses hash table:** *O(M)*

**• Distances hash table:** *O(L)*

**• Overall:** *O(N + M + L)*

**• Time Complexity:** Each initialization is a constant-time operation O(1) , but loading data into hash tables would take O(N) , O(M) , and O(L) respectively:

**• Overall:** *O(N + M + L)*

1. **Main Loop for Each Truck**

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

**• Space Complexity**: Mainly involves variables and data structures already initialized. Additional space for variables like current\_location and nearest\_package is O(1) .

**• Overall:** *O(1)*

**• Time Complexity:**

**•** Outer loop for each truck:O(T) where T is the number of trucks.

• Inner loop involves finding the nearest unassigned package and loading the truck, which is dependent on the number of packages N .

1. **Inner While Loop for Delivery**

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()

**• Space Complexity:** Additional variables like nearest\_location are O(1) .

**• Overall:** *O(1)*

**• Time Complexity:** The while loop runs for each package in the truck:

• Finding the nearest neighbor: O(N) for linear search through the packages.

**• Delivering the package:** *O(1)*

**• Updating package status:** *O(1)*

**• Total per package***: O(N)*

**• Overall:** *O(N^2) per truck (assuming the truck can hold all packages).*

1. **Find Nearest Neighbor Function**

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

**• Space Complexity:** *O(1) for variables min\_distance and nearest\_package.*

**• Overall:** *O(1)*

**• Time Complexity:** *Looping through all packages in the truck:*

**•** *O(N) per call*

**Overall Complexity**

**• Space Complexity:**

• Dominated by the hash tables and other data structures: *O(N + M + L)*

**• Time Complexity:**

*• Initialization: O(N + M + L)*

*• Main truck loop: O(T)*

*• Inner loops: O(T x (N + N^2)) = O(T x N^2)*

*• Overall: O(T x N^2)*

**• Best Case:** If the number of trucks T is significantly smaller or if each truck only needs to deliver a small number of packages, the time complexity could approach *O(N^2)* dominated by the truck with the most packages.

**• Worst Case:** All trucks are utilized to their maximum capacity, and each must deliver all N packages, leading to *O(T x 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**

* **Choice of Data Structure:** Using hash tables for storing package, address, and distance data ensures quick lookups and updates, typically O(1) on average. This reduces the time complexity for operations, making the system more efficient.
* **Modularity**: The design promotes modularity by breaking down the system into smaller, manageable parts such as HashTable, Truck, Package classes etc. and separates distinct functionalities (initialization, package loading, route finding, and delivery) into modules, making each part easier to understand, optimize, and debug. Each class or function has a single responsibility, making it easier to modify or replace without affecting the entire system. Abstraction hides complexity, allowing for improvements without affecting other parts of the system.
* **Encapsulation**: By encapsulating data and functions within appropriate classes or modules, the design reduces dependencies and minimizes the impact of changes. Encapsulation ensures that the internal workings of a module are hidden from other parts of the system.
* **Readability and Clarity**: The pseudocode is structured in a way that is easy to read and understand. Clear naming conventions and a logical flow make it easier for developers to follow the code and identify where changes need to be made.
* **Reusability**: Functions like find\_nearest\_neighbor and update\_package\_status\_in\_hash\_table are designed to be reusable. This reduces code duplication and makes it easier to maintain and extend the system.
* **Object Oriented Programming:** By modeling entities like packages, trucks, and the hub as classes, the design leverages the power of object-oriented programming. This allows for better organization of data and behavior, making the code more intuitive and easier to manage. These OOP concepts can be used to extend the functionality of the system without altering the existing codebase. For instance, different types of packages or trucks can be introduced by extending base classes. OOP allows for dynamic method binding, which can simplify the process of adding new functionalities and handling various edge cases.

**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.