Core Algorithm Overview

Stated Problem:

This task is the planning phase of the WGUPS Routing Program.

The Western Governors University Parcel Service (WGUPS) needs to determine an efficient route and delivery distribution for their daily local deliveries (DLD) because packages are not currently being consistently delivered by their promised deadline. The Salt Lake City DLD route has three trucks, two drivers, and an average of 40 packages to deliver each day. Each package has specific criteria and delivery requirements that are listed in the attached “WGUPS Package File.”

Your task is to determine an algorithm, write code, and present a solution where all 40 packages will be delivered on time while meeting each package’s requirements and keeping the combined total distance traveled under 140 miles for two of the trucks. The specific delivery locations are shown on the attached “Salt Lake City Downtown Map,” and distances to each location are given in the attached “WGUPS Distance Table.” The intent is to use the program for this specific location and also for many other cities in each state where WGU has a presence. As such, you will need to include detailed comments to make your code easy to follow and to justify the decisions you made while writing your scripts.

The supervisor should be able to see, at assigned points, the progress of each truck and its packages by any of the variables listed in the “WGUPS Package File,” including what has been delivered and at what time the delivery occurred. (WGU Performance Assessment, n.d.)

Algorithm Overview:

I chose the Greedy algorithm because of its ability to make locally optimal choices at each step, taking into account the provided information and selecting the most suitable path for package deliveries. Its efficiency and simplicity make the Greedy algorithm well-suited to address the delivery problem for (WGUPS) in Salt Lake City, allowing for the determination of an efficient route and effective distribution of packages for the daily local deliveries.

The Greedy Algorithm code is created in the following manner:

1. Clearly state the problem the greedy algorithm aims to solve, such as determining an efficient route and delivery distribution for daily local deliveries.
2. Identify the criteria for making locally optimal choices at each step, such as package priorities, distance to the package location, and truck capacity.
3. Construct the greedy solution by iteratively adding elements/packages based on the identified criteria.

4. Check for feasibility and optimality by verifying if the solution meets all constraints and requirements of the problem, such as on-time deliveries and adherence to truck capacities and distances.

B.

Using a Priority Queue with the Greedy algorithm efficiently handles the relationship between data components by organizing packages based on priority criteria. The Greedy algorithm can easily access and process high-priority packages first, ensuring optimal choices for an efficient route and delivery distribution.

C.

<u>C.1</u>

The provided pseudocode presents a Greedy algorithm for efficient package delivery management by (WGUPS) in Salt Lake City. It utilizes two empty trucks, truck1 and truck2, along with a priority queue to handle packages based on their priority criteria. The algorithm prioritizes high-priority packages and assigns them to the available trucks, updating their locations and delivery times accordingly. With a time complexity of O(n*logn) or O(n), this Greedy approach optimizes the delivery process to ensure timely distribution of packages.

```
▪ Function greedy_algorithm(locations, packages)
    ▪ Initialize truck1 as an empty list
    ▪ Initialize truck2 as an empty list
    ▪ Create priority_queue and insert packages based on priority criteria

    ▪ Set current_location_truck1 to 'Starting Point'
    ▪ Set current_location_truck2 to 'Starting Point'

    ▪ While priority_que is not empty:
        ▪ Set highest_priority_package to get_highest_priority_package
          from priority_queue

    ▪ If truck1 has capacity for highest_priority_package:
        ▪ Set distance_to_package to
          calculate_distance(current_location_truck1,
          highest_priority_package.delivery_address)
```

```
            ▪ Update truck1's current_location and delivery time
            ▪ Add highest_priority_package to truck1
        ▪ Else:
            ▪ Set distance_to_package to
              calculate_distance(current_location_truck2,
              highest_priority_package.delivery_address)
            ▪ Update truck2's current_location and delivery time
            ▪ Add highest_priority_package to truck2
        ▪ Mark highest_priority_package as delivered in delivery_status
    ▪ Return truck1, truck2
▪ End Function
```

<u>C.2</u>

- Software: Pycharm 2023.2 Community edition, 3.9
- Hardware: Apple Macbook Pro M1, 16GB, MacOS Ventura 13.4.1

<u>C.3</u>

- The time complexity of each segment:
- Initializing truck1 and truck2: O(1)
- Creating the priority queue and inserting packages: O(n*logn) or O(n) depending on the priority queue implementation.
- Setting current_location_truck1 and current_location_truck2: O(1)
- While loop: O(n) as it iterates through the priority queue.
- Getting highest_priority_package: O(1) or O(logn) depending on the priority queue implementation.
- Checking truck capacity and updating locations: O(1)
- Adding packages to trucks: O(1)
- Marking packages as delivered: O(1)
- The overall time complexity of the Greedy algorithm is dominated by the priority queue operations, giving us a final time complexity of O(n*logn) or O(n), depending on the priority queue's implementation.

<u>C.4</u>

The proposed solution using the Greedy algorithm with a priority queue demonstrates good scalability, efficiently handling a growing number of packages. The priority queue allows for constant-time retrieval of the highest priority package, ensuring efficient decision-making during deliveries. While the algorithm's simplicity offers a scalable approach, for extremely large numbers of packages, more advanced algorithms could be explored to optimize performance further. Nonetheless, the current implementation provides an effective solution for the WGUPS to meet their future delivery demands.

C.5

The software design is efficient and easy to maintain due to the Greedy algorithm's fast execution with a priority queue, ensuring quick access to the highest priority package. The straightforward logic and modular nature of the design make it easy to read and debug, simplifying maintenance. Additionally, the solution is adaptable and scalable, capable of handling varying package demands and larger datasets efficiently. Overall, the design is a reliable and effective solution for parcel delivery, providing flexibility for future needs.

C.6

The self-adjusting data structure, exemplified by the priority queue employed in the Greedy algorithm, has its own set of strengths and weaknesses. On the positive side, such data structures provide efficient operations, enabling quick insertion, deletion, and retrieval of elements, thereby optimizing the algorithm's time complexity. Additionally, their dynamic nature allows them to adapt flexibly to changing data, ensuring optimal performance even with varying input sizes. Notably, priority queues excel at handling elements based on priority, making them well-suited for tasks like package delivery, where priorities determine processing order. However, there are also drawbacks to consider. Self-adjusting data structures may introduce complexity overhead in terms of memory usage or computational complexity compared to simpler data structures. Moreover, their implementation and maintenance can be more intricate and prone to errors than traditional data structures. Therefore, a thoughtful evaluation of these strengths and weaknesses is crucial to effectively and appropriately employ self-adjusting data structures in addressing real-world problems.

C.7

The most suitable key for efficient delivery management from the provided components would be the "package ID." The package ID is a unique identifier assigned to each package, making it easy to access and track specific packages throughout the delivery process. Using the package ID as the key allows for quick retrieval and manipulation of package-related data, enabling efficient organization and monitoring of packages at every stage, from the hub to their final destinations. This choice ensures streamlined package handling and simplifies the implementation of the delivery algorithm.

D.

- *WGU Performance Assessment*. (n.d.). https://tasks.wgu.edu/student/001425632/course/30860017/task/4041/overview, Scenario Cited