C950 Task-1 WGUPS Algorithm Overview

(Task-1: The planning phase of the WGUPS Routing Program)

Aaron Ballesteros

ID #011019047

WGU Email: abal387@wgu.edu

04/11/24

C950 Data Structures and Algorithms II

# A. Algorithm Identification

I plan to use the Greedy Algorithm to determine the optimal route for delivering all packages within the constraints of the program requirements.

# B. Data Structure Identification

A hash table serves as a suitable data structure the self-adjusting data structure that will be used in collaboration with the greedy algorithm for storing package data as it will provide an efficient lookup, insertion, and delete functionality.

# B1. Explanation of Data Structure

A hash table will be the data structure I will use to store package data. By using a hash table, we are simplifying package data retrieval and accessibility. Chaining will resolve hash collisions; where each table index points to a list of entries that hash to the same index. Chaining will handle dynamic relationships and collisions by mapping package IDs (as keys) to their matching package details (values) and storing them in a linked list at each index of the additional package details information such as the address, weight, status, deadlines, and more. We are simply linking the ID (key) and other components of data in the linked list (values) and storing the in the hash table buckets, or, in other words, the bucket list. The chaining hash table will maintain associations between the relationships of data components and streamline search, insertion, and deletion processes of the components of data.

# C1. Algorithm’s Logic

A screenshot of a computer

Description automatically generated

# C2. Development Environment

**Apple MacBook Pro**

**Processor**: Apple M1 Max

**Memory**: 32 GB

**Operating System**: macOS Sonoma 14.2

**Interpreter**: Python 3.11

**IDE**: PyCharm 2024.1 (Professional Edition)

# C3. Space and Time complexity using Big-O notation

The chaining hash table is for collision handling, offering average-case time complexities of O(1) for insert, search, and deletion operations. In the case of high load factors or uneven key distribution, increases to O(n). The process to load package data iterates over each row in the CSV, parses package details, and inserting each package into the hash table. This process demonstrates a linear time complexity of O(n). The greedy algorithm, determining the nearest next delivery location iteratively. The comparison between distances for each package per delivery iteration demonstrates a time complexity of O(n^2). The function to calculate distances between two point; given the distances in a matrix, the time complexity for accessing distances is O(1), implying constant time complexity. Updating the status of packages and logging delivery metrics do not significantly alter the overall time complexity, remaining within a linear scale, O(n), for updates across all packages. Interface sections for user input and menu navigation operate with a constant time complexity, O(1), for each action taken by the user. Overall, the entire program operates within O(n^2) due to the nested iterations required for calculating delivery routes and the sequential handling of package deliveries.

# C4. Scalability and Adaptability

The program is designed with scalability and adaptability, utilizing a hash table with chaining for efficient package management, regardless of the growing number of packages. This data structure ensures that search, insertion, and deletion operations remain efficient even as the data magnifies. The flexible design of the Trucks class further enhances scalability, allowing for the addition of more trucks to accommodate an increase in deliveries. The delivery simulation algorithm, while maintaining a time complexity of O(n^2) due to its nested iterations, is crafted to handle an increasing workload without degradation in performance speed. Overall, the program’s architecture and selected data structure supports seamless scaling and adaptation to handle more packages and extend delivery capabilities if the need arises.

# C5. Software Efficiency and Maintainability

By using object-oriented principles, the program segments into well-documented classes and functions, enhancing both readability and maintainability. Each component, from the chaining hash table for package storage to the trucks and packages classes for delivery management, is carefully annotated to facilitate understanding and ease of modifications. The hash table's average-case time complexity of O(1) for operations reinforces the program's efficiency, minimizing resource utilization and maximizing performance. Ensuring that the program utilizes resources efficiently while delivering optimal results, making future updates, and troubleshooting straightforward.

# C6. Self-Adjusting Data Structures

Some of the strengths that the chaining hash table excels in is simplicity, ease of implementation, and performance efficiency, particularly in handling collisions through linked lists. However, one of the weaknesses is the dependency on linked lists for collision resolution. This introduces additional memory overhead, as empty slots are necessary to preserve efficiency. The tradeoff results in an increase in storage requirements but is justified by the significant performance benefits and the flexibility it offers in managing dynamically changing data.

# C7. Data Key

The most efficient component for data management is the package ID because of its uniqueness. The package ID guarantees that each package can be identified and managed within the hash table, avoiding the ambiguities and inadequacies that could arise from non-unique attributes like addresses, zip codes, or delivery status. Using the package ID as the key ensures that each entry in the hash table is individual, enabling rapid access and updates. This key choice focuses on efficiency and accuracy in package management, streamlining the delivery process irrespective of other overlapping package attributes.

# D. Sources

Tepe, C., MBA. C950 - Webinar-1 - Let’s Go Hashing. College of Information Technology, Western Governors University. Retrieved from <https://westerngovernorsuniversity-my.sharepoint.com/:b:/g/personal/cemal_tepe_wgu_edu/EWjwOOwEwCdHsW83oQ2az00BkTCjFFiGXk8a6eXls74wLg?e=EhpKrF/>

Tepe, C., MBA. (Year). C950 - Webinar-1 - Let’s Go Hashing - Complete Python Code. College of Information Technology, Western Governors University. Retrieved from <https://westerngovernorsuniversity-my.sharepoint.com/:u:/g/personal/cemal_tepe_wgu_edu/EXaXbjKAci5EhnaWjPab6iMBc0zOUb_dOa_b-FwY4zeumg?e=1EN3Bl/>