C950 Task-1 WGUPS Algorithm Overview

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

Alexander Fair

ID # 001574781

WGU Email: afair26@wgu.edu

2/25/2024

C950 Data Structures and Algorithms II

# Introduction

The purpose of this assignment is to create a program that will use algorithms to route 3 trucks with 2 drivers to deliver 40 packages while staying under a total of 140 miles traveled for 2 of the trucks.

# A. Algorithm Identification

I implemented a programming solution using the greedy nearest neighbor algorithm for optimizing package delivery routes. This algorithm efficiently selects the closest next destination from a current point, prioritizing speed over finding the absolute shortest path. Although this approach doesn't ensure the optimal solution in all cases, it significantly accelerates the computation process, providing a sufficiently effective route quickly. By focusing on minimizing the immediate distance to the next point, the algorithm simplifies the complexity of route planning, making it a suitable choice for applications where rapid route calculation is essential. This method exemplifies a compromise between computational efficiency and solution accuracy, ideal for scenarios where immediate execution is more valuable than exhaustive optimization.

# B. Data Structure Identification

For the package list, create a hash table where the key is the package ID, and the package info is stored in the buckets.

For the address list, create a dictionary where the key is an address identifier (unique address ID number) and the value is the full address details.

For the distance table, create a dictionary where each key is a tuple of address identifiers representing a pair of addresses, and the value is the distance between them.

# B1. Explanation of Data Structure

The hash table provides constant time complexity *O*(1), for search insert and delete operations on average, which is crucial for quickly accessing package details based off their ID’s during the routing process. Hash tables also have well-defined methods for handling collisions ensuring that even in the cases of key conflicts, package data remains accessible and distinct. Hash tables also allow for flexible handling of package data, including addresses, delivery deadlines, delivery status, and weight. The flexibility is essential for dynamically updating package statuses or requirements as routing decisions are made.

# C1. Algorithm’s Logic

BEGIN optimized\_delivery WITH truck

CREATE an empty list for unsorted\_packages

FOR EACH package\_id in truck's load

FIND package using package\_id

ADD package to unsorted\_packages list

END FOR

EMPTY truck's load

WHILE there are packages in unsorted\_packages

SET closest\_package as the first package in unsorted\_packages

FOR EACH package in unsorted\_packages

IF package is closer to truck's current location than closest\_package

SET closest\_package to this package

END IF

END FOR

MOVE closest\_package from unsorted\_packages to truck's load

UPDATE truck's location to closest\_package's destination

UPDATE truck's mileage and time based on distance to closest\_package

MARK closest\_package as delivered

END WHILE

END optimized\_delivery

# C2. Development Environment

PyCharm Version: 2023.3.4

Python Version: 3.11.6

Hardware: Dell Latitude 5580 running Windows 10 version 10.0.19045 Build 19045

Using stock PyCharm with the Rainbow Brackets and CSV Editor plugins.

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

The overall space time complexity of the program is O(n2). Space complexity is dominated by the storage requirements of the distance matrix. And Time complexity is for the optimized delivery process. The analysis is below.

The space time complexity for the method *optimized\_delivery* is O(n) & O(n2).

* Space Complexity: O(n) for storing the unsorted list of packages.
* Time Complexity: O(n2) it is due to the iterative search for the closest package for each delivery step, where n is the number of packages.

The *distance\_calculations* methods are both O(1).

* Space Complexity: it calculates and returns a single value.
* Time Complexity: direct access to the matrix.

The *package\_delivery* method are both O(1).

* Space Complexity: O(1) operating on an existing object without additional significant storage.
* Time Complexity: O(1) with direct assignments with simple arithmetic operations.

The *find\_closest\_package* method is O(1) & O(n).

* Space Complexity: O(1) for using a fixed number of variables.
* Time Complexity: O(1) iterating once through the list of packages

The *reading and load data* methods are both O(n).

* Space Complexity: O(n) for storing package and address data linearly.
* Time Complexity: O(n) for reading from CSV files where n is the number of lines.

# C4. Scalability and Adaptability

The scalability of the program would be challenged on a performance perspective after a significant number of packages are being delivered due to the quadratic time complexity O(n2) of the optimized delivery method which would degrade the system’s performance. In addition, due to the trucks being manually loaded there would need to be additional code written to handle the assignment of packages to trucks for handle increased package quantities. The code would have increased difficulties to handle various situations of the notes and other special requirements for packages.

# C5. Software Efficiency and Maintainability

The delivery optimization method could be further improved to increase efficiency as currently it runs at a O(n2) complexity which is quadratic. But overall, the remaining code keeps to a polynomial runtime or better for the remaining methods for decent overall efficiency. I refactored the code into smaller methods to allow for more modularity and easier to discern different portions of the code for future expansion or maintenance, in addition there are comments and documentation throughout to explain the methods for others.

# C6. Self-Adjusting Data Structures

The hash table has noticeable strengths and weaknesses. One of the weaknesses of the hash table is that it can require additional memory to maintain efficiency in operations compared to other alternatives. Another weakness is that only immutable types can be used as keys which limits the usage of mutable types (dictionaries or lists). Their strengths on the other hand outweigh the weaknesses. One such strength is that they provide a constant time complexity O(1) for lookup, insert and delete operations on average, making data access remarkedly fast. Another strength of the hash table is that they store data in key-value pairs, enabling direct access to values through unique keys, which simplifies data retrieval and manipulation.

# C7. Data Key

I used the package ID as my choice of key for the hash table. Using the package ID as the key allowed for an identifier to distinguish all the packages from each other. There shouldn’t be any 2 packages with the same ID number. The other options left ambiguity and room for potential overlap for duplicate keys, multiple packages having the same delivery address, weight or deadline.

# D. Sources

R. Fair, personal communication, February 20, 2024 (25+ years’ experience as programmer)

Lysecky, R., & Vahid, F. (2018, June). *C950: Data Structures and Algorithms II*. zyBooks.

Retrieved February 10, 2024, from <https://learn.zybooks.com/zybook/WGUC950AY20182019/>

# E. Professional Communication

Nothing to write. Run your document through <https://www.grammarly.com/>