C950 Task-1 WGUPS Algorithm Overview

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

Andrew Hilleary

ID: 011315946

WGU Email: ahil783@wgu.edu

Date: 4-2-2024

C950 Data Structures and Algorithms II

# Introduction

The task of determining an algorithm to apply to a package delivery service for the quickest usage of a limited number of trucks and packages.

# A. Algorithm Identification

A greedy algorithm would work best for the quickest way to solve a problem without demanding a perfect solution. Of those, a nearest neighbor algorithm would be able to identify the nearest or “greediest” option at each stop in the package route and will be used in my program to do so.

# B. Data Structure Identification

To store data for use in the algorithm, I will use a hash table due to the nature of the structure and the self-adjusting needs.

# B1. Explanation of Data Structure

A hash table will allow an easily scalable way to store the weight of all the routes. The constant lookup time will allow the algorithm to execute quickly and access the stored keys from the buckets. I intend to use chaining to enable ease of removal and insertion into the table. The package ID will serve as a key for sorting into the various buckets due to the uniqueness property.

# C1. Algorithm’s Logic

PackageList = {

Id1: address1,

Id2:address2,

….

}

Packages = (id1,id2…)

Next\_package = 1

Miles\_driven = 0

Time = 8:00 # AM

Speed = 18 #mph

While Packages is not Empty

Destination = Find nearest destination by distance

Next\_package = packages[destination]

If package has requirements

change according to requirements

Deliver(Next\_package)

Distance = PackageList[Next\_package]

Miles\_driven = Miles\_driven + Distance

Time = Time + ( Distance/Speed)

Remove Next\_package from Packages

# C2. Development Environment

My environment will be my home computer running Windows, with hardware containing a GTX 3080, an Intel i7, and 16 Gb of RAM. The software I will be using is VS Code Editor version 1.87.2 running Python version 3.10.11.

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

The limiting time complexity overall will be O(n^2) due to the algorithm requiring the while loop to occur for every package, (n), and within the while loop executing the find nearest distance portion iterating over another list of distances, (m). Using a hash table here will allow constant, O(1) lookup and removal time making the change functions constant. This makes the deliver function operate in O(1). Updating the time and the miles will of course operate in O(1) as well. Therefore, the time will be O(n^2). Space complexity will be simple as we only have two lists to iterate, packages (n) and distances (m) making the space complexity O(n+m).

# C4. Scalability and Adaptability

My program should be able to scale to suit whatever needs the customer has. The program operates independently of whatever number of packages or trucks that it requires and will not be affected by an increase. I will implement a class that will allow the addition of new trucks and new packages with accompanying requirements like time.

# C5. Software Efficiency and Maintainability

The maintenance of the code should be minimal as I will implement classes for the packages and trucks that should disallow any oddities that would disrupt the algorithm. The methods to add packages will be simple to avoid confusion and will check for formatting that is required. This also makes the algorithm efficient to implement by having a defined way of adding new records to the lists therefore simplifying the process.

# C6. Self-Adjusting Data Structures

The self-adjusting data structure works incredibly well with quick inserts and removal of records working in constant time. It is also inherently scalable due to the self-adjusting nature of it. The efficient search time for records leads to quicker application of the structure.

While the typical lookup time is O(1), collision issues could occur to slow that down to a worst-case O(n). This can be avoided using proper chaining methods. The overhead space from hash tables also suffers if the load factor of the buckets is too high.

# C7. Data Key

The package ID fits the criteria for an optimal key. The package ID is the most suitable choice for a key because address, deadline, city, zip code, weight, and status all would have multiple packages with potentially the same value which would cause more issues rather than just having a unique ID for each package.

# D. Sources

I did not use any sources

# E. Professional Communication

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