WGUPS Routing Program Algorithm Overview

# Algorithm Identification

The Greedy Algorithm was used to route and deliver the packages in this program.

# Logic Comments

**For I in length of dfPackages:**

**Package = package with id of i**

**If package is not delivered and either the truck is not defined or the truck is equal to the truck specified:**

**If earliest availability time is after current time, skip this package**

**If package has delivery dependency, set the soonest dependency to deliver**

**Otherwise, set the soonest package to deliver by deadline**

**Add the location of the package to deliver to the locations list**

**For var in [SoonestDependencyPackage, soonestPackage]:**

**If var is not none:**

**If distance to location is not null, deliver to this location**

**Otherwise iterate over location column and find distance to the location and deliver**

**For I in length of dfLocations:**

**If location of id I is in locations:**

**If distance to location is not null and shorter than the current shortest distance:**

**Shortest = distance**

**If location is equal to the current location:**

**Shortest = 0**

**If location is equal to the current location:**

**CurrentLocationIndex = I – 1**

**If shortest is not 0:**

**For index, (name, value) in enumerate(current location columns):**

**If index > 1:**

**If location with id index – 1 is in locations:**

**If distance to location is not null and the distance is shorter than current shortest:**

**Shortest = value**

**If shortest is not none:**

**For I in length of dfPackages:**

**If package address is the shortest distance address and the package has not been delivered:**

**Deliver package**

# Development Environment

Text Editor: Visual Studio Code

Language: Python 3.10.7

Operating System: Windows 10

# Space-time and Big-O

Time complexity:

* Make list of locations: O(N)
* Deliver to soonest deadline: O(N)
* Find shortest distance from current location: O(N)
* Find shortest distance to current location: O(N)
* Deliver to shortest distance: O(N)
* Overall program time complexity: O(N)

As more packages and cities are added, the time needed to run the algorithm increases linearly.

Space complexity:

* Make list of locations: O(1)
* Deliver to soonest deadline: O(1)
* Find shortest distance from current location: O(1)
* Find shortest distance to current location: O(1)
* Deliver to shortest distance: O(1)
* Overall program space complexity: O(1)

O(1) As more packages and cities are added, the space needed to run the algorithm remains constant.

# Scalability and Adaptability

The utilization of iterating over the length of packages and locations allows for an unlimited growth in the number of packages and locations. The hash table used significantly increases the speed of finding an element.

# Software Efficiency and Maintainability

Only necessary data is used and updated throughout the algorithm, and loops are limited and optimized. Comments throughout the code help to explain what certain sections of code do and enable an environment in which bug fixing and updating is simple.

# Self-adjusting Data Structures

The hash table enables a fast way to locate data without having to iterate over an entire list. However, data stored within the hash table is not organized, rather stored by the hashing algorithm.

# Data Structure

Package and location data are stored within hash tables to allow for fast lookup. Data are stored within lists in the hash table buckets.

# Screenshots

Text

Description automatically generated

Text

Description automatically generated

A picture containing text

Description automatically generated

# Strengths of Greedy Algorithm

1. Chooses the shortest distance to travel
2. Able to optimize by multiple parameters

# Other Possible Algorithms

Instead of using a greedy algorithm, a backtracking algorithm could be used instead. Instead of choosing the best case at each step, the backtracking algorithm would attempt to find a shorter distance between multiple locations at once. A mile count could be specified as a goal for a specific number of locations, and if at any point that mile count is passed, the backtracking algorithm would backtrack to find a more optimal solution. For example: a mile count of 15 could be set for 5 locations, and the algorithm would attempt to find a solution to deliver to 5 locations within 15 miles.

Another algorithm that could be used is dynamic programming. The algorithm would track the distance between multiple locations at every state to develop a sense of direction. If the distance between one location increases from location A to location B, we know that the truck moved away from that location. If we know that that location will later be on the path back to the hub, we can choose to deliver to that location later when the truck is returning to the hub.

# Different Approach

If I were to attempt the project again, or improve upon it, I would implement either a backtracking algorithm or a dynamic programming solution. The greedy algorithm is easiest to implement but might not be the most efficient algorithm for this problem.

# Other Data Structures

Instead of using a hash table, data could be stored in a binary search tree. Instead of using a hashing algorithm, the binary tree insertion would traverse the tree by moving to the right if the id is greater than the node value and left if the id is less than the node value.

If there aren’t likely to be many packages or locations, a simple list would work just fine. Since lists are native to python, an insertion algorithm wouldn’t need to be created.