C950 WGUPS Algorithm Overview

Jennifer Cole

ID # 001265274

WGU Email: jcol627@wgu.edu

Date 10/10/2021

C950 Data Structures and Algorithms II

# Introduction

This program is designed to find an optimal solution for delivering packages, given a specific number of trucks, drivers and mileage allowed. The program should be able to deliver all 40 packages with three trucks and travel no more than 140 miles. The program should also allow for certain constraints, such as delivery times and late arrivals.

# A. Algorithm Identification

Nearest Neighbor Algorithm was used in this program.

# B1. Logic Comments

The Nearest Neighbor Algorithm works as follows:

1. Takes an initial value and a set of data and loops through to find the nearest point to the initial value.
2. The nearest point now gets reassigned as the initial value and is simultaneously removed from the set of data.
3. The program loops through the remaining data again, to find the next nearest point to the initial value.
4. This reassignment, removal, and looping continues until all data are accounted for.

Time complexity for this algorithm as used in this program is O(n), as discussed below in B3.  
 **minDistanceFrom method O(n)**  
 minimum = *value to hold distance between 2 address, initialized to 1000*  
 time\_delivered = *initialized to the time the truck left the hub* current\_address = *initial address to compare all other addresses to*  
 minimum\_address = current\_address  
 pkg\_to\_deliver\_next = *holds package ID for which package will be next (package ID holds all package info, such as address)*

for index in range(*amount of packages on truck*): **O(n) based on number of packages on truck** pkgID = *package ID of current index, initialized to 0*  
 next\_address = *address associated with pkgID*  
 a1\_a2 = *calls distanceBetween method to calculate distance between current\_address and next\_address*

if a1\_a2 < minimum:   
 *replace values with current index’s information as the nearest neighbor*  
 minimum = a1\_a2  
 minimum\_address = next\_address  
 pkg\_to\_deliver\_next = pkgID

time\_delivered = *call timeDelivered method to timestamp delivery*  
call isDelivered method  
return\_values = [minimum\_address, minimum]  
return return\_values

**distanceBetween method O(1)**  
 returns distance between two addresses

**getAddress method O(1)** returns address of a package

**isDelivered method O(1)** returns time package was delivered and removes package from the truck

**timeDelivered method O(1)** calculates delivery time based on seconds from current address and mph  
 returns delivery time, time object

# B2. Development Environment

Python 3.9 and PyCharm Community Edition 2021.2

# B3. Space-Time and Big-O

Major code segments include:

1. Hash Table functions (insert, search, remove methods) – O (1)
2. Deliver Packages (deliverPackages method) – O(n)
3. Min Distance From (minDistanceFrom method) – O(n)

The program as a whole has a time complexity of O(n). The basic segments all have a time complexity of either O(1) or O(n), where n is the number of packages and their corresponding data, being read into the program as lists. The Hash Table’s insert, search and remove functions all have O(1) since it does not need to loop through lists to find corresponding values. It takes one key and decides which bucket to go to, based on the key. Nearest Neighbor Algorithm is O(n^2). The algorithm first calls Deliver Packages O(n), which then calls Min Distance From O(n), to loop through all packages each time it is called. This results in a time complexity of O(n^2). Ultimately, the number of packages (n) is going to determine the time complexity for the entire program.

# B4. Scalability and Adaptability

My software is scalable and adaptable in the sense that it would be easy to add more trucks as the number of packages increases, simply by creating another truck object. Each truck is independent of other trucks. A major downside is that the software currently runs based on manually deciding which package goes on which truck. While this is doable on a small scale, the package sorting/loading process would need to be revisited as the number of packages increases.

# B5. Software Efficiency and Maintainability

The software is easy to maintain because most data is read in from csv files. As is, the only thing needed to manually adjust in the code itself is the number of trucks (if needed) and to update which packages go on which truck. It is efficient because all the lists used throughout the program will adjust automatically as the list grows so, as the number of packages increase, so to will the size of the lists.

# B6. Self-Adjusting Data Structures

One major strength of the hash table is its speed. If the number of entries is known from the get-go (aka maximum number of packages that will ever be delivered in a day) then the modular algorithm can be decided in advance. It would always have a constant time complexity. A weakness is the probably for collisions as the number of packages grows. This can eventually become too inefficient to continue using this program as is.

# C. Original Code

pkgDelivery included in zip file.

# C1. Identification Information

pkgDelivery included in zip file.

# C2. Process and Flow Comments

pkgDelivery included in zip file.

# D. Data Structure

A hash table is used in this program to store the package data. This table is quickly accessible at its current size, which allows for ease of use with the “Nearest Neighbor” algorithm implemented.

# D1. Explanation of Data Structure

The relationship between my data structure and my data points is accounted for using the package’s ID number. The ID number is not special to the package in any way other than making it unique. By using a hash table, I’m able to store the data in an easily identifiable and easily accessible way using this unique identifier. Memory access is one of the slower processes for a computer or program to run so having an access time of O(1) for the data points makes this data structure ideal.

# E. Hash Table

pkgDelivery included in zip file

# F. Look-Up Function

pkgDelivery included in zip file

# G. Interface

pkgDelivery included in zip file

# G1. First Status Check

# 

# G2. Second Status Check

# 

# G3. Third Status Check

# 

# H. Screenshots of Code Execution

# 

# I1. Strengths of Chosen Algorithm

Two strengths of my chosen algorithm is that it is simple to understand and does not need to be adjusted when more data is fed in (as long as the data fed in is formatted correctly).

# I2. Verification of Algorithm

The algorithm used meets all requirements. See pkgDelivery included in zip file.

# I3. Other possible Algorithms

Two other algorithms that could have been used are Dijkstra’s algorithm and Brute Force algorithm

# I3A. Algorithm Differences

1. Dijkstra’s Algorithm stores its information in a graph (vs. a list, like in Nearest Neighbor Algorithm). Another difference is Dijkstra’s discovers the shortest path from beginning node to end node, whereas Nearest Neighbor just loops through the list to find the next nearest option.
2. Brute Force algorithm typically has a slower run time than Nearest Neighbor because as the number of packages grow, it becomes more and more inefficient. Another difference is that Brute Force seeks the most optimal solution while Nearest Neighbor seeks the most efficient solution.

# J. Different Approach

The major thing I would have done differently is I would have implemented a package sorting algorithm to choose which packages should go on each truck, instead of manually loading them.

# K1. Verification of Data Structure

The data structure meets all requirements. See pkgDelivery included in zip file.

# K1A. Efficiency

Depending on the number of packages that are added, time needed to complete the look up function is not affected by the addition of packages. Currently, the lookup time is O(1). A threshold would need to be established to determine the appropriate package capacity to adjust the modulus value in the hash table to prevent collision, but ultimately, lookup time should not be affected.

# K1B. Overhead

Space usage is affected by the number of packages added because the chance of collision increases as the number of packages increases.

# K1C. Implications

The number of trucks and cities would not affect the lookup time of the packages or the space usage of the data structure. The lookup time is still O(1) with this specific data structure.

# K2. Other Data Structures

Two other possible data structures to use are a Hash Map and Binary Search Tree (BST).

# K2a. Data Structure Differences

1. A Hash Map is different than a Hash Table because you cannot have null values in a Hash Map. You also cannot have null keys like you can in a Hash Table. Hash Maps are also slower than Hash Tables.
2. BST is different from a Hash Table in that they have a lookup/access time complexity of O(log n) whereas a Hash Table is O(1). BST also allows for easier sorting than Hash Tables do.

# M. Professional Communication

Professional communication guidelines followed.

# L. Sources - Works Cited

* C950 - Webinar-1 - Let’s Go Hashing – [Complete Python Code](https://westerngovernorsuniversity-my.sharepoint.com/personal/cemal_tepe_wgu_edu/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fcemal%5Ftepe%5Fwgu%5Fedu%2FDocuments%2FMyDocs%2FC950%2FWebinar%2FW%2D1%5FChainingHashTable%5FzyBooks%5FKey%2DValue%2Epy&parent=%2Fpersonal%2Fcemal%5Ftepe%5Fwgu%5Fedu%2FDocuments%2FMyDocs%2FC950%2FWebinar&originalPath=aHR0cHM6Ly93ZXN0ZXJuZ292ZXJub3JzdW5pdmVyc2l0eS1teS5zaGFyZXBvaW50LmNvbS86dTovZy9wZXJzb25hbC9jZW1hbF90ZXBlX3dndV9lZHUvRVhhWGJqS0FjaTVFaG5hV2pQYWI2aU1CYzB6T1ViX2RPYV9iLUZ3WTR6ZXVtZz9ydGltZT0yTFQtYzhXVjJVZw)
* C950 - Webinar-3 - How to Dijkstra – [Complete Python Code](https://westerngovernorsuniversity-my.sharepoint.com/personal/cemal_tepe_wgu_edu/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fcemal%5Ftepe%5Fwgu%5Fedu%2FDocuments%2FMyDocs%2FC950%2FWebinar%2FW%2D3%5FChainingHashTable%5FzyBooks%5FKey%2DValue%5FCSV%5FGreedy%5FDijkstra%2Epy&parent=%2Fpersonal%2Fcemal%5Ftepe%5Fwgu%5Fedu%2FDocuments%2FMyDocs%2FC950%2FWebinar&originalPath=aHR0cHM6Ly93ZXN0ZXJuZ292ZXJub3JzdW5pdmVyc2l0eS1teS5zaGFyZXBvaW50LmNvbS86dTovZy9wZXJzb25hbC9jZW1hbF90ZXBlX3dndV9lZHUvRVlVNzdrSDFrZHBGbDVBeTdKNGc3eDRCeFVqTWlTTzRKZzkwNllSM0dZWlBJdz9ydGltZT10TWNxRjhXVjJVZw)