C950 WGUPS Algorithm Overview

Samuel Miranda

ID #001144463

WGU Email: smira33@wgu.edu

3/28/2023

C950 Data Structures and Algorithms II

# Introduction

This Python program uses a ‘Nearest Neighbor Algorithm’ approach to delivering packages to each address. This proposal will cover this program's method, logic, and reasoning. Please see the code demonstration for an in-depth understanding of this program’s approach to efficiently delivering packages.

# A. Algorithm Identification

In this program, the ‘Nearest Neighbor’ Algorithm is used to deliver all packages in the demonstration program.

# B1. Logic Comments

Func Shortest\_distance()

Get an address ‘a’ and package list []

temp\_list []

For a package in package list[],

goingto = package.address

temp\_list.append(distance(a, goingto))

return: min(temp\_list)

Func Nearest Neighbor()

Start = ‘HUB’

Best\_route = []

Get a list of packages

Remaining\_packages = Copy of list

While remaining\_packages:

Best\_route.append(shortest distance(start, package list)

Remaining\_package.remove(package)

Start = package.address

Return best\_route

# B2. Development Environment

Python 3.11, Mac OS Ventura 13.2.1, MacBook Air M2, 2022

# B3. Space-Time and Big-O

|  |  |  |
| --- | --- | --- |
|  | Time | Space |
| Insert() | O(n) | O(n) |
| Search() | O(n) | O(n) |
| remove() | O(n) | O(1) |
| loadPackageData() | O(n) | O(n) |
| loadDistanceData() | O(n) | O(n) |
| findDistance() | O(n) | O(n) |
| shortDistance() | O(n) | O(n) |
| truck\_shortest\_path() | O(n^2) | O(n) |
| Deliver\_packages() | O(n) | O(n) |
| Print\_all\_all packages | O(n) | O(n) |
| Print a package | O(n) | O(n) |
| Main() | O(n^2) | O(1) |

# B4. Scalability and Adaptability

Logically speaking, the program can accommodate delivering more packages in its current structure. Adaptability-wise, a more segmented (higher index) hash is necessary to maintain efficiency. However, this would have a significant impact on space. As packages are added, the package ID, along with the package object, is stored in a bucket within a list. As more packages are added, those lists will grow longer and longer. A higher index would allow for smaller lists in each array bucket.

The first consideration for scalability would be upgrading the server to handle more packages. If the packages were even 10x, scaling up to take the increase in CPU time would be necessary.

# B5. Software Efficiency and Maintainability

This program seeks to simplify parameters by deriving information from already in-place data. For example, the package class keeps only two datetime data points: departure and delivery times. The program derives the package's status through logic, i.e., ‘En Route’ and ‘At the Hub,’ based on the two DateTime data points. This keeps the program efficient by minimizing data per package object.

Since the program can be broken into essential blocks of code, there is room for other APIs to be swapped in/out for future updates. For example, the ‘finding distance’ method could be replaced by a more robust API that can still provide those same distance inputs into the program to produce a similar result. Any programmer could ‘install’ a change on one code block instead of rewriting the entire program only to accommodate an antiquated variable path. That and comments throughout the program should make for easy maintenance.

# B6. Self-Adjusting Data Structures

Some benefits of the hash table are how space efficient and flexible it is. Since it uses a fixed array, it is efficient when performing inserts (or deletions, when necessary). While still flexible in what it can store since the key-value pair can index any object.

Drawbacks include collisions that can affect efficiency. Null values are not allowed, and hash tables can become full. Also, printing out or manipulating in order can take time, so that sorting can become an issue. (2018; GeeksforGeeks, 2023)

# D. Data Structure

Along with the ‘Nearest Neighbor’ algorithm, package objects are stored in a chaining hash table. Collisions are handled by creating an adjusting list for the indexed array.

# D1. Explanation of Data Structure

In this program, package objects were stored in a chaining hash table. So, the key-value pair, in this case, was the package class package ID (key) and the package object (value) to store all the data points for a particular package.

Each key-value pair is divided into, or assigned, a bucket (index in the array), then organized into lists within the buckets.

# G1. First Status Check

9 am Status Check

Text

Description automatically generated

# G2. Second Status Check

10 pm Status Check

Text

Description automatically generated

# G3. Third Status Check

1 pm Status Check

Text

Description automatically generated

# H. Screenshots of Code Execution

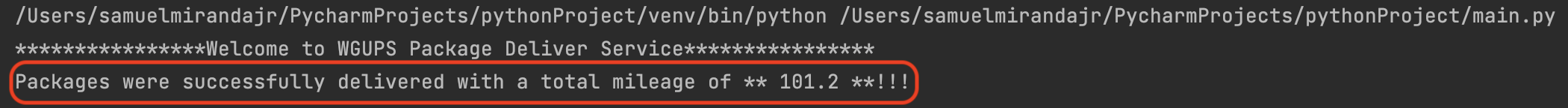
# Text Description automatically generated

# I1. Strengths of Chosen Algorithm

The most relevant strength of the ‘Nearest Neighbor’ algorithm is 1. It changes as needed to accommodate different lists of the same class. 2. It is simple to implement and intuitive. It is mainly driven by one parameter. Since it is so easy to use and adaptive, it makes a good contender for this particular use case of delivering packages. ([www.fromthegenesis.com](http://www.fromthegenesis.com), 2018)

# I2. Verification of Algorithm

The ‘Nearest Neighbor’ algorithm was self-adjusting and could sort a list that accomplishes mileage under 140. It delivers all packages for each truck.



# I3. Other possible Algorithms

Two additional self-adjusting algorithms would be 1. Greed Algorithm, and 2. Dijkstra’s algorithm. These would be able to create short path lists as well to deliver packages with a minimal amount of truck mileage.

# I3A. Algorithm Differences

The greedy algorithm differs in that although it does not find the optimal solution for (in this case) delivering packages, it does provide a reasonable time complexity. Nearest Neighbor takes a toll on time complexity. Greedy may outperform Nearest Neighbor at scale (a more significant amount of packages), although at the expense of not delivering them all at the quickest time possible. (Wikipedia Contributors, 2022)

Dijkstra’s Algorithm finds the shortest path between points on a graph. Nearest Neighbor in this demonstration program does the same, although it could adapt to find the fastest route, which would be more practical for truck delivery. On the other hand, Dijkstra’s Algorithm should be a better fit if this program was built for drone deliveries. Practically speaking, it would be the best fit over Nearest Neighbor. At scale, Dijkstra would also need some adjustments (i.e., Fibonacci’s heap) to prioritize which nearest node to choose. If that were the case, it would have a more competitive time complexity. (Wikipedia contributors, 2023)

In short, Dijkstra’s is best for a graph, while greedy is very efficient in time complexity. Nearest Neighbor finds a reasonable path and yet still is uncomplicated to implement.

# J. Different Approach

A few opportunities for change probably start with the UI. The program could benefit from a graphical interface, using filters for a list in a dialog box. Or a list that is updated in real-time by the minute. By simply (in this specific project) deriving the status of each package using ‘current time’ as the user's chosen time. Also, implementing another self-adjusting structure for the addresses could improve the program's efficiency. It could implement a binary tree to store addresses so each package wouldn’t need to be accessed just to generate an efficient route.

# K1. Verification of Data Structure

The chaining hash is self-adjusting and stores all the package objects. It holds the truck id and the duration from when the truck left to when delivered. A global mileage metric also contains the total mileage for each truck and is summed to provide that the total mileage was less than 140 miles. Any more data points that need to be stored would be updated by the package class to be then stored in the hash. Package class objects will then be efficiently found by package ID using the search (or lookup) function.

# K1A. Efficiency

If the hash array were increased, the lookup function (search and insert) would still maintain the time for finding a package ID, and value itself. The number of steps to access a package from the hash would be the same.

# K1B. Overhead

If more packages were being delivered, the data in each package object would also increase linearly. Scaling out, i.e., getting more servers to handle the space of the data of each package, would be necessary, and updating the hash array index to prevent collision as well.

# K1C. Implications

Truck objects are not saved in this program in the hash data structure. Getting more trucks would not make a significant change in the program at all. Time and space usage would be maintained. The truck info for each package is kept in the package class, and if more servers are provided to keep those data points for each package in the hash table, the truck info will also be reserved.

# K2. Other Data Structures

Other self-adjusting data structures that could be used are binary trees or a simple list.

# K2a. Data Structure Differences

A binary tree would not serve package insertion in the same way that the chaining hash does. However, the binary tree could search and insert with relatively low time complexity. Because the node is not indexed, a hash index array will still outperform it. Using another hash with a different probing method may have been just as efficient but not as easy to maintain. Since the function which probes allocates to a different array, more space is needed. That’s why a chaining hash is a better option.

# L. Sources - Works Cited

G. (2018, September 25). *Pros and Cons of K-Nearest Neighbors - From The GENESIS*. From the GENESIS. https://www.fromthegenesis.com/pros-and-cons-of-k-nearest-neighbors/

GeeksforGeeks. (2023, March 28). *Applications  Advantages and Disadvantages of Hash Data Structure*. https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-hash-data-structure/

Wikipedia contributors. (2022, November 23). *Greedy algorithm*. Wikipedia. https://en.wikipedia.org/wiki/Greedy\_algorithm

Wikipedia contributors. (2023, March 3). *Dijkstra’s algorithm*. Wikipedia. https://en.wikipedia.org/wiki/Dijkstra%27s\_algorithm