**WGUPS Routing Program Algorithm Overview**

Lydia Strough

ID: #002452624

WGU Email: lstroug@wgu.edu

12/12/2022

C950 – Data Structures and Algorithms II

# Introduction

The scenario is as follows: There are three delivery trucks. Each delivery truck is loaded with a max of 16 packages. There are 40 packages in total. Each package has a delivery address, delivery time deadline, and other constraints. It is assumed that packages are loaded and delivered instantaneously. There are two delivery truck drivers, meaning that only two delivery trucks can transport their associated packages at a time. Each truck drives at an average pace of 18 miles per hour. The goal is to deliver these packages optimally, in 140 miles or less, while meeting all package constraints. The day concludes when all 40 packages have been delivered.

# A. Algorithm Identification

In the program created, package deliveries were optimized via the “nearest neighbor” (greedy) algorithm – the shortest distance was calculated between package deliveries. It is assumed that the shortest distance is equivalent to the optimal solution.

# B.1. Logic Comments

Algorithm logic explained through pseudocode:  A picture containing text, document, screenshot

Description automatically generated

# B.2. Development Environment

**Chip**: Apple M1

**Memory**: 8 GB

**OS**: macOS, Ventura 13.0.1

**Interpreter**: Python 3.11 (using PyCharm 2022.2.3 CE)

# B.3. Space-Time and Big-O

**Big-O calculation**

HashTable.py

| **Method** | **Time-Space Complexity** |
| --- | --- |
| \_\_init\_\_ | O(n) |
| insert (doubles as an update) | O(n) |
| search | O(n) |
| remove | O(n) |
| **Total** | 4 \* O(n) = **O(n)** |

main.py

| **Method** | **Time-Space Complexity** |
| --- | --- |
| distance\_in\_between | O(1) |
| min\_distance\_from | O(n) |
| load\_truck\_packages | O(1) |
| deliver\_truck\_packages | O(n^2) |
| command\_user\_interface | O(log n) |
| main() | 4 \* O(n) + 2 \* O(1) + 3 \* O(n^2) + O(log n) = O(n^2) |
| **Total** | 2 \* O(1) + O(n) + 2 \* O(n^2) + O(log n) = **O(n^2)** |

Package.py

| **Method** | **Time-Space Complexity** |
| --- | --- |
| \_\_init\_\_ | O(1) |
| \_\_str\_\_ | O(1) |
| print\_status\_for\_time | O(1) |
| **Total** | 3 \* O(1) = **O(1)** |

readCSV.py

| **Method** | **Time-Space Complexity** |
| --- | --- |
| load\_package\_data | O(n) |
| print\_package\_table | O(n) |
| load\_address\_data | O(n) |
| load\_distance\_data | O(n) |
| **Total** | 4 \* O(n) = **O(n)** |

Truck.py

| **Method** | **Time-Space Complexity** |
| --- | --- |
| \_\_init\_\_ | O(1) |
| \_\_str\_\_ | O(1) |
| **Total** | 2 \* O(1) = **O(1)** |

**Space-Time Complexity**

(HashTable.py + main.py + Package.py + readCSV.py + Truck.py)

= O(n) + O(n^2) + O(1) + O(n) + O(1)

= 2 \* O(n) + 2 \* O(1) + O(n^2)

= **O(n^2)**

# B.4. Scalability and Adaptability

Scalability and adaptability were core concerns when designing this program. The chaining hash table was implemented to decrease package search time as the number of packages increases (lower search time by scaling the hash table as needed). More truck objects can be created to support the increase in packages. Additional CSV files can be added to the program and read via the ‘readCSV.py’ file.

# B.5. Software Efficiency and Maintainability

The core of this program's algorithm is implemented using an object-oriented architecture. All key components of the program (classes, objects, and methods) are well-commented to ensure readability. Within the methods are several print functions that have been commented out that will assist any future programmer with debugging. Each of these factors will assist heavily in program maintainability and efficiency.

# B.6. Self-Adjusting Data Structures

In summary, a chaining hash table is an array of linked lists. The length of these linked lists affects overall data retrieval. If these linked lists are “never-ending,” data retrieval can be lengthy. However, a positive of a chaining hash table is its ability to grow dynamically. By increasing the hash table’s number of linked lists (buckets), the length of those linked lists decreases, which decreases the number of collisions and overall search time.

# C.2. Process and Flow Comments

All major components of the program (classes, objects, and methods) are well-commented to ensure readability.

# D. Data Structure

The self-adjusting data structure that was chosen to store all package information was a linear probing, chaining hash table.

# D.1. Explanation of Data Structure

The hash table has four functions: insert, update, search, and delete. The hash table acts as an array of linked lists and can retrieve data values as fast as direct hashing. In the first stage of the program, the 40 packages are inserted into the hash table. The hash table is later accessed and searched via the unique package id (the hash table key).

# G.1. First Status Check

Time checked: 08:40:00

Text

Description automatically generated

# G.2. Second Status Check

Time checked: 09:40:00

Text

Description automatically generated

# G.3. Third Status Check

Time checked: 12:40:00

Text

Description automatically generated

# H. Screenshots of Code Execution

H.1. Welcome screen: Asks user to type 1, 2, or 3

Text

Description automatically generated

H.2. User retrieves the information for package #1

Text

Description automatically generated

H.3. User retrieves all package statuses for time: 08:40:00

Text

Description automatically generated

H.4. User chooses to exit the program

Graphical user interface, text

Description automatically generated

# I.1. Strengths of Chosen Algorithm

The overall algorithm is user-friendly and flexible. If a future programmer decided that they wanted to make any changes to the program, due to its object-oriented architecture, the programmer could make these changes easily, as needed. For example, more truck objects could be created, and the load truck function could be adjusted to meet the future programmer's needs.

Also, if the programmer wanted to increase the number of packages, the hash table’s initial capacity could be adjusted to decrease bucket length, thereby decreasing data recovery time.

# I.2. Verification of Algorithm

The algorithm meets the “less than a combined total of 140 miles” traveled cumulatively (at 110.5 miles). The trucks were manually loaded trucks while considering all constraints. After each truck delivered all its packages, the distance between the final package address and the hub was added to the truck's total miles. Before truck three left the hub, at 10:20:00, package number nine’s associated address was changed to the new, correct address. After the trucks completed their deliveries, the hash table search function was used to check that all packages met their specific deadlines. Below is a list of each truck’s package list, as well as the constraints that were considered while loading the trucks. Also, within the code are commented-out print functions that were used to keep track of all aspects of the delivery process. These print functions were (and can be) used to verify the accuracy of the algorithm.

Text

Description automatically generated

# I.3. Other possible Algorithms

The heuristic algorithm and Dijkstra's shortest path algorithm could have both been used in place of the greedy algorithm and met all requirements for this project.

# I.3.a. Algorithm Differences

### Dijkstra's shortest path algorithm

* In contrast to implementing the greedy algorithm with a chaining hash table, Dijkstra's shortest path algorithm implements a graph. The graph’s edges are associated with a numerical value between vertex items. The algorithm finds the shortest path from a start vertex to each vertex in the graph.

Breadth-First Search algorithm

* In contrast to implementing the greedy algorithm with a chaining hash table, the breadth-first search algorithm implements a graph. A starting vertex is initialized, and the graph divides the vertices into two categories: visited and not visited.

# J. Different Approach

If I could do this project again, I would develop a method that would sort the packages based on their constraints, vice manually loading the packages on the truck. Also, I would format the welcome menu differently – I would format option one (specific package information) in a way that would include the specific package’s truck information. I would want to see the time that the truck left the hub.

# K.1. Verification of Data Structure

The implemented chaining hash table meets all requirements for this project. The hash table includes an insert function that takes in all package information. The hash table also contains a search function that returns all required package data elements.

The cumulative truck mileage was less than the required 140 total miles, at 110.5 miles. All packages were delivered within their expected deadlines, and the reporting data can be verified by un-commenting the print functions provided throughout the program. These print functions list specific details regarding truck locations, shortest package distances, and package delivery statuses. The print functions also include details regarding the truck’s “last stop” - determining the distance from the final package delivery back to the hub.

# K.1.a. Efficiency

The chaining hash table is an efficient way to store and look up package information. The chaining hash table uses key-value pairs for managing its data and uses linear probing for its functions. The worst-case space-time complexity for linear probing is O(n). As the number of packages increases, the longer the linked lists to be searched. The longer the linked list, the more time it takes to retrieve the requested data.

# K.1.b. Overhead

As described above in section K.1.a (Efficiency), as the number of packages increases, the longer the linked lists to be searched within the chaining hash table. The longer the linked list, the more time it takes to retrieve the requested data.

# K.1.c. Implications

City addresses are implemented via a list and are only used to calculate the time and distance between packages. Increasing the number of cities would have minimal effect on look-up time and/or space usage. The trucks are merely objects used for carrying the delivery information throughout the program. Increasing the number of trucks would not affect look-up time and/or space usage.

# K.2. Other Data Structures

A binary search tree (BST) and a weighted graph could have both been used in place of the chaining hash table and meet all requirements for this project.

# K.2.a. Data Structure Differences

Binary search tree (BST): A BST is a collection of nodes with edges that connect them. The nodes of the BST are organized by level. The node at the highest level is called the root. Each node can have up to two child nodes (a left and a right node). The BST is split into a left and a right subtree (the children of the root node). In a BST, each node is greater than every node in its left subtree. In a BST, searching through and inserting data always begins at the root.

* Differences: In contrast to a BST, chaining hash tables use key-value pairs and linked lists to insert and search through their data. Generally, hash tables are faster than BSTs with a constant space-time efficiency of O(1), compared to the BST at O(log n). Worst-case, however, hash tables will run slower than a BST at O(n). Although hash tables are generally faster than a BST, BSTs are guaranteed to run at O(log n) in any situation. BSTs also maintain sorted order while hash tables do not. Hash tables also

Weighted graph: A weighted graph is a graph in which edges are associated with a value (weight). A graph’s edge represents a numerical value between vertex items. A graph can be either directed or undirected.

* Differences: In contrast to a chaining hash table, the weighted graph would eliminate the need to calculate and store the distance between package addresses. However, the worst-case space-time complexity for a weighted graph is O(n^2) compared to a hash table’s O(n).

# L. Sources - Works Cited

**C950 WGUPS Project Implementation Steps – Example. Retrieved from**

<https://srm--c.na127.visual.force.com/apex/CourseArticle?id=kA03x000001DbBGCA0>

zyBooks, a Wiley brand. (2020). **C950: Data Structures and Algorithms II.**<https://learn.zybooks.com/zybook/WGUC950AY20182019>(accessed 2022).

Breadth First Search or BFS for a Graph**. Retrieved from**

<https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>