C950 Task-2 WGUPS Write-Up

(Task-2: The implementation phase of the WGUPS Routing Program).

(Zip your source code and upload it with this file)

Jennifer Murphy

ID #001532028

WGU Email: jmur326@wgu.edu

February 28, 2024

Version II

C950 Data Structures and Algorithms II

# A. Hash Table

Creating the chaining hash table through the use of lists in buckets:

# A screenshot of a computer program Description automatically generated

Defining the function that will load the package data into the hash table:

# A screen shot of a computer Description automatically generated

Calling function in Main to load the hash table:

**A screenshot of a computer screen

Description automatically generated**

# B. Look-Up Functions

The function used the search the hash table for the package ID as key and return the key value pair.

# A computer screen shot of a program Description automatically generated

# C. Original Code

Package Class with method to update package status:A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

Truck Class:

A screen shot of a computer program

Description automatically generated

The function find\_address retrieves the address information from the file. The distance\_between function retrieves the distance data from the data matrix.

**A screen shot of a computer

Description automatically generated**

Nearest Neighbor Algorithm Implementation:

**A screenshot of a computer program

Description automatically generated**

**A computer screen shot of a program code

Description automatically generated**

Loading the packages into the trucks:

A screenshot of a computer program

Description automatically generated

**A computer screen shot of a program

Description automatically generated**

User interface Implementation:**A screenshot of a computer program

Description automatically generated A screenshot of a computer program

Description automatically generated**

# C1. Identification Information

# A screenshot of a computer program Description automatically generatedC2. Process and Flow Comments

# A computer screen shot of a code Description automatically generated

# A computer screen shot of a black screen Description automatically generated

# A computer screen shot of a code Description automatically generated

# A computer screen shot of a black screen Description automatically generated

# D. Interface

Prompt 1:

A screen shot of a computer

Description automatically generated

Prompt 2:

A screenshot of a computer program

Description automatically generated

# D1. First Status Check

Single Package at 08:45:

A screenshot of a computer program

Description automatically generated

All Packages at 08:45:

A screenshot of a computer program

Description automatically generated A screen shot of a computer

Description automatically generated

# D2. Second Status Check

Single Package at 09:45:

A screenshot of a computer program

Description automatically generated

All Packages at 09:45:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

# D3. Third Status Check

Single Package at 12:45:

A screenshot of a computer program

Description automatically generated

All Packages at 12:45:

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

# E. Screenshot of Code Execution

A screen shot of a computer

Description automatically generated

# F1. Strengths of the Chosen Algorithm

The nearest neighbor algorithm has several strengths. One strength is the simplistic implementation and maintenance of the algorithm. It will be easy to understand and edit in the future. The simplicity of the algorithm makes it quite efficient for smaller data sets with a time complexity of O(n^2). The nearest neighbor algorithm does not provide the absolute best path to take, but the path it finds is very effective for a package delivery situation. The algorithm is also scalable to allow for different amounts of packages and addresses.

# F2. Verification of Algorithm

The nearest neighbor algorithm meets the requirements of the provided scenario. It takes that data from the hash table (self-adjusting data structure with insert and lookup functions), extracts the address, and uses the address to return the distances from the CSV file. It uses this distance to calculate the shortest path from the truck’s current location to the next closest package. The algorithm interacts with the truck object by updating the truck’s time and the delivery time of the package. The algorithm is used by the user interface to allow the user to track packages at any entered time and retrieve all the data associated with a specific or all packages. The user entered time also updates the status of the packages after they have been loaded on a truck and once they have been delivered. The package’s truck number is updated and shows for the user when they retrieve information. The algorithm also tracks and keeps the mileage of all trucks under the required 140 miles and delivers all packages within the specified time frames taking into account the address change on package 9.

# F3. Other Possible Algorithms

Other possible algorithms that could be applied to the scenario are the greedy algorithm and the brute force algorithm. Both algorithms would meet the requirements defined in the project.

# F3a. Algorithm Differences

The greedy algorithm chooses the best option currently available, while the nearest neighbor considers all unvisited options. It is also simple to implement but is more applicable to multiple situations while nearest neighbor is best suited for a routing situation. The greedy algorithm also tends to have a better time complexity than nearest neighbor depending on its application.

The brute force algorithm evaluates all possible solutions for delivering the packages and returns the absolute optimal route. The time complexity of this algorithm is O(n!) compared to the nearest neighbor’s complexity of O(n^2) making it significantly more time consuming and impractical for large quantities of data. This is the best algorithm when accuracy is the desired outcome while nearest neighbor gives a close enough solution.

# G. Different Approach

If I complete this project again, I would like to create a function that optimally loads the trucks. I manually sorted and analyzed the package data in the CSV files to determine which truck the packages should be loaded onto. Then I manually loaded each truck using the package ID. I would also try to implement a more sophisticated algorithm, such as Dijkstra’s algorithm to create priority packages while delivering by giving the packages with deadlines a weight. This would make the project more scalable, practical, and beneficial.

# H. Verification of Data Structure

The chaining hash table has an insert and search (look up) option that takes input that allows the package data to be inserted without collisions and returns with all applicable components without errors. The insert function also allows for the address update needed for package 9 before delivery. The total truck mileage is tracked and when added is under 140 miles. All packages are delivered by the time specified in the notes or by the end of the day.

# Efficiency

The time needed to perform the search (lookup) function is affected by the number of packages linearly. The function uses the hash function to translate the package ID into an index that corresponds to a bucket. It then searches the list attached to that bucket for a matching ID. If a match is found, it returns the key value pair. If there are less packages, the list at each bucket would be shorter and take less time to search. Alternately, if there are more packages, the list at each bucket would be longer and take more time to search before returning a matching key value pair. When the number of packages increases so does the time to complete the search (lookup) function.

# Overhead

Similarly to the time needed to search, the space usage would also differ based on the number of packages needing to be delivered. Adding additional packages would lead to an increase in the overall size of the hash table hence increasing the space usage. The lists at each bucket would become longer as the package number increase. Eventually, the number of buckets would need to be increased and a new hash function created to compensate for a larger data set. The memory usage needed to store the hash table would grow and would increase the space usage of the hash table.

# Implications

Increasing the number of trucks or cities would not affect the search (look-up) time or space usage of the hash table. The cities are tied to the packages stored in the hash table. If the number of packages remained constant, but the cities were more varied, it would not affect the hash table. The truck objects are not stored in the hash table and would not affect the time or space usage of the data structure. The number of trucks and the number of cities would not change the look-up time or space usage of the data structure.

# H1. Other Data Structures

Other data structures that would meet the requirements of this scenario are a binary search tree and a linked list.

# H1a. Data Structure Differences

A binary search tree stores nodes in a hierarchical structure. Each node has children and are arranged in a numerical order where the left nodes are less than the right. It stores data using a unique identifier, such as a package ID, where each item is a node in the tree. A binary search tree allows for a more efficient search than a chaining hash table due to the items being stored in order taking less time to find. A disadvantage of a BST over a hash table is that a BST consumes more memory.

A linked list would also use nodes to represent packages. Each node contains a pointer to the next node in the list. To search for a package, the list would have to be traverse sequentially until the desired package was reached adding to the search (lookup) time. The chaining hash table reduces search time by using a hash function to locate an index where it begins its search. Needing a pointer to the next node raises the amount of memory needed for each unit of data stored in the linked list. Unlike a hash table that may need to be reformatted, a linked list’s size changes dynamically with the size of the data set making insertions and deletions simple.

# I. Sources

Not Applicable

# J. Professional Communication