Mark Wilson
C950 Performance Assessment
Western Governors University
October 11, 2022

A: Algorithm Identification

A greedy algorithm using the nearest neighbor method was developed in order to efficiently deliver the packages based on the shortest distance between two addresses.

B1: Logic Comments

The aforementioned greedy algorithm is contained within the function named deliver_packages. The current location is initialized as the Hub. The while loop ensures that the algorithm will continue to run as long as there are items in the list. The minimum distance is set to a number that is much larger than any distance that would be found. The next_location and index variables are set to an empty value.

The for loop iterates through the list of items. If the distance to the item's address is less than the current minimum distance, it is set as the new minimum distance. The item's location and its index are stored in next_location and index, respectively.

When all items in the list have been compared, the current_location takes the value of next_location and the item is removed from the list. The algorithm then begins again from the while loop until the algorithm reaches its base case, an empty list.

```
current_location = HUB

While list is not empty
    min = 9000.0
    next_location = None
    index = None
    For all packages in list
        If distance to item_location <= min
            min = distance
            next_location = item_location
            index = item_index

    Current_location = next_location
```

Remove list[index]

B2: Development environment

The application was developed in Python on a MacBook Air (M1 2020) with 8GB of memory. The operating system in use is macOS Monterey version 12.5. The IDE that was used is PyCharm Community Edition 2022.2.1

B3: Space-Time and Big-O

This information is included in the comments of the code itself.

B4: Adaptability

The nearest neighbor greedy algorithm does not take the return trip back to the hub into account when finding an optimal route. There is a greater chance for inefficiency as the number of addresses and the distance between them increases.

The hash table's capacity, currently set at ten, can easily be increased in order to decrease the look up time for additional packages.

B5: Software Efficiency and Maintainability

The program runs in polynomial time (n^2).

Packages and trucks are stored as objects, so adjustments to their properties can be easily made. For instance, if WGUPS added additional trucks that could travel at a faster rate than 18 miles per hour, those trucks could have those attributes set differently. The code is also thoroughly commented and makes good use of whitespace for readability. The use of discrete functions allows for their reuse in future components of the application.

B6: Self-Adjusting Data Structures

For the hash table, the code is easy to read and understand. Future developers should have little issue determining exactly how the algorithm works.

Another strength is that the hash table is not overly complicated. It would not be too difficult for future developers to make adjustments or additions to account for new scenarios, such as additional packages.

A weakness of the implemented hash table is there is no flexibility for insertion. New packages can only be added to the end of the chained list. Also, as the amount of packages increases, the amount of time it takes to search through each chained list will also increase.

A strength of the nearest neighbor algorithm is its simplicity. Other more efficient methods require additional objects and computations, which may require additional resources for computation and maintenance. The added complexity also adds more potential vectors for error.

Another strength is its reusability. Components of the algorithm could also potentially be used in other parts of the program, such as an automated sorting and loading method to replace the manual method currently in use.

A drawback of using this algorithm is that the algorithm finds the next closest address from the current address, but this may not be the most optimal path to take overall.

Another downside is that the algorithm is unidirectional and does not account for the return trip to the hub. An algorithm that accounts for the return trip would be able to drop off packages on the way back if it results in a more optimal path.

D1: Explanation of Data Structure

The hash table is instantiated as a list of lists with an initial capacity of 10. As each package is added to the table, the index is determined by getting the remainder of the hash value of the key divided by the capacity of the table. The application then iterates through the chained list that is located at the index that corresponds with the hash value.

This implementation is more efficient than a simple linear search. A package's information can be accessed by utilizing the hash method to find the index where the package information is located. The algorithm would then only have to utilize a linear search in the chained list at that index, which is much faster than performing a linear search through all packages.

G1 - G3 & H: Status Checks and Screenshots of Code Execution

These image files can be found in the folder named "Screenshots" within the root of the zip file.

I1: Strengths of the Chosen Algorithm

A strength of the nearest neighbor algorithm is its simplicity. Other more efficient methods require additional objects and computations, which may require additional resources for computation and maintenance. The added complexity also adds more potential vectors for error.

Another strength is its reusability. Components of the algorithm could also potentially be used in other parts of the program, such as an automated sorting and loading method to replace the manual method currently in use.

I2: Verification of Algorithm

All packages were delivered on time, according to their specifications, within 121.20 miles. The total mileage is displayed upon starting the application. The truck number, time deadline, and time of delivery are displayed in the output of the application.

I3: Other possible algorithms

Other possible algorithms that could have been used include Dijkstra's algorithm and nearest insertion.

I3A: Algorithm Differences

Utilizing Dijkstra's algorithm requires additional Graph and Vector objects. The algorithm takes all possible routes into account before returning the most optimal. It requires more computational power than the nearest neighbor algorithm used. The number of computations increases drastically as more vertices are added to the graph.

The nearest insertion algorithm accounts for a round trip, unlike the chosen method and Djikstra's algorithm. As it only compares distances as needed, it does not require as much computational power as Djikstra's algorithm.

J: Different Approach

Package sorting was done manually. First loading the packages according to their requirements. Then, ensuring that as many packages with the same address were loaded onto the same truck as possible. After that, any packages leftover were placed on trucks that were already traveling to that general area.

In hindsight, an algorithm could have been used to sort based on the given requirements. The greedy algorithm that was developed to deliver the packages could have been utilized to more efficiently sort the items onto trucks as well.

K1: Verification of Data Structure

All packages were delivered on time, according to their specifications, within 121.20 miles. The total mileage is displayed upon starting the application. The truck number, time deadline, and time of delivery are displayed in the output of the application. An efficient hash table with a look-up function is present and is used to hold instances of the Package class. The package statuses and information can be verified through the user interface. All information displayed through the user interface is accurate.

K1A: Efficiency

The hash table utilizes chaining to store lists within lists. While the key lookup is instantaneous, a linear search is utilized for the chained list that corresponds to that key. As more packages are added, the longer the chains get, as more keys share the same hash value. As such, an increase in packages will increase the amount of time needed to potentially find an individual package.

K1B: Overhead

Each additional package increases the size of the chained list corresponding to its hash value by one.

K1C: Implications

Adding trucks and cities by themselves will not affect the performance of the hash table, as those entities are not stored in the hash table. If extra packages are added along with the trucks and cities, then performance will be affected. As the chained list utilizes a linear search, the time to traverse through each chained list increases with the amount of packages stored within it.

K2: Other Data Structures

Other data structures that could have been used instead of the hash table with chaining include a hash table with a linear probing method and a hash table that utilizes linked lists.

K2A: Data Structures Differences

A hash table utilizing linear probing deals with collisions by storing a key value pair with the same hash as an existing pair in the next available slot in the table. For this method to work properly, the capacity of the table must always be greater or equal to the number of key value pairs.

A hash table with chaining linked lists allows for greater flexibility than chaining with a standard list. Items can be inserted anywhere in the list. A linked list uses pointers to link items together, which results in an increase in memory usage as those pointers must also be stored in memory.

L: Sources

No outside sources were used.