WGUPS ROUTING PROGRAM

Aaron Balke

College of Information Technology, Western Governors University

C950: Data Structures and Algorithms II

Dr. Amy Antonucci

May 31st, 2023

A: ALGORITHM SELECTION

The self-adjusting algorithm I used in this project is the Nearest Neighbor Algorithm. When additional packages are loaded onto the trucks, the calculation to find the shortest path does not have to be restructured or written, it automatically adjusts to the input change. This is because of the WHILE statement, looping as long as packages are in the truck, and the FOR statement, loops through those packages to compare distances.

B1: LOGIC COMMENTS

Nearest Neighhbor Pseudocode:

chosen_package

chosen_distance

while packages_in_truck:

---for each package in packages_in_truck:

------travel_distance = distance between truck_location and package_delivery_location

------if travel_distance < chosen_distance:

---------chosen_distance = travel_distance

---------chosen_package = package

------Subtract chosen_package from packages_in_truck

1. Define blank variables chosen_distance and chosen_package
2. While packages are in the truck do the following:
    a. Loop through packages in the truck and find the distance between the truck and that package delivery location

   b. If this package has a distance shorter than any previous one, set it as the chosen package, and the chosen distance.
   c. Subtract this chosen package from the truck
3. This will end when all packages in the truck have been removed

B2: DEVELOPMENT ENVIRONMENT

This program is written in Python 3.10 using Pycharm Community Edition 2023.1.2. The program was originally written on a custom Windows 10 64-bit desktop.

B3: SPACE-TIME AND BIG-O

The first major block of code in my program is the Hash Table. The Hash Table is a data structure that stores data in an associative KeyValue Pair. The Hash Table's functions can be run at O(1) time complexity, making it preferable in data-heavy environments. For our case, keeping track of packages in a Hash Table is effective since it is called 10+ times, if each call required O(n) time complexity, the program would slow down substantially. (Iyer)

The next major block of code is the Nearest Neighbor Algorithm calculation. While packages are in the truck, the algorithm will search package addresses to find the closest one. This portion has a time complexity of O(n^2) which comes from the while statement checking for packages in the truck, and the for statement looping through those packages to find the shortest distance. There are additional statements inside this algorithm that affect the time, such as looping through the location data to find the id for a string name, but this is not affected by the input size

B4: ADAPTABILITY

The Hash Table will scale great and be adaptable in almost any package load due to its O(1) time complexity. Regardless of the number of packages, finding a value is always the same amount of time.

The Nearest Neighbor portion will scale, however, it won't be fast at higher inputs. As we add more inputs the time complexity of O(n^2) begins to get bogged down. On the bright side, it is still significantly faster than exact brute-force methods with time complexities of O(n!).

B5: SOFTWARE EFFICIENCY AND MAINTAINABILITY

The program is efficient because its main algorithm and data structure run in O(n^2) and O(1). To be efficient it has to be O(n^2) at minimum. It is maintainable because of many enhancements to the code. The main enhancement is the use of Object-Oriented Programming. With OOP, we can define many different objects and classes to simplify the repeating structures such as Trucks, and Packages. By doing so, the code becomes much easier to maintain, and future developers can easily make adjustments because of abstraction and encapsulation. Additionally, the use of comments to provide descriptions for all methods and variables gives direction to someone looking at the code for the first time.

B6: SELF-ADJUSTING DATA STRUCTURES

The Hash Table implemented in this program provides many advantages in the process of accessing the packages. The main advantage is through the use of hash functions, the time complexity of accessing this structure is O(1). Additionally, unlike trees, pointers are not required to be stored increasing storage efficiency. The big disadvantage of Hash Tables is as they get bigger, we begin to get more and more collisions, slowing performance. Luckily for our small-scale use, this was not applicable. (aayuski2402) Another disadvantage is that the order is

not maintained between elements. In our case, we could not store the order the packages were delivered in, within the Hash Table. This aspect had to be stored separately. (Data Structures Handbook: Hash Tables)

C: ORIGINAL CODE

See attached

D: DATA STRUCTURE

The self-adjusting data structure that can store the package information is a Hash Table, adapted from the "Webinar - 1 - Let's Go Hashing", published 12/24/2022. The Hash Table works by using the "Division Method" of associating a hash to a Key-Value pair for O(1) lookups and insertions. In my implementation the Division Method is referred to as "bucket", the division method calculation is h(k) = k mod M, where k is the key, our Package ID, and M is the size of the Hash Table. The hash is a unique associated value created to directly associate a key with a corresponding value, which is significantly faster to calculate at larger scales compared to linear searching. (Data Structures Handbook: Hash Tables)

By entering a package ID, we can get the package data in the same amount of time regardless of total packages, which would not be the case with linear searching.

E: HASH TABLE

The hash table insertion function is called "add" in my implementation.

F: LOOK-UP FUNCTION

The hash table lookup function is called "get" in my implementation.

G: INTERFACE

8:35 a.m. and 9:25 a.m. See attached (screenshot_g1.jpg)

9:35 a.m. and 10:25 a.m. See attached (screenshot_g2.jpg)

12:03 p.m. and 1:12 p.m. See attached (screenshot_g3.jpg)

H: SCREENSHOTS OF CODE EXECUTION

See attached (screenshot_h1.jpg, screenshot_h2.jpg)

I1: STRENGTHS OF THE CHOSEN ALGORITHM

The two main strengths of the Nearest Neighbor Algorithm are its simplicity and efficiency. My original methods, Brute Force and Dijkstra, required handling the problem in its entirety at some point. Brute Force required comparing each entire path to each other, and the Dijkstra method required sorting all possible path segments to find the shortest. In the Nearest Neighbor Algorithm, you are only concerned with a single node and the paths between it and all other points at a time, making it a smaller local problem to fix and then build toward the bigger problem's solution. This makes the implementation very simple.

Also, compared to exact algorithms, Nearest Neighbor is incredibly efficient. The time allocated to finding a solution was effectively instant, whereas with my original Brute Force method I had to shut it down after 40 minutes of calculating. In our simulation, the 40+ minutes it took figuring would’ve easily been better served to get packages out earlier on a less exact path.

I2: VERIFICATION OF ALGORITHM

Verify Total Miles: Available on Program Execution

Verify All Packages Delivered on Time: If you request all packages at any time, the delivery time for each is printed as well

Verify Delivery Specifications: When all packages are printed at the required delivery times, the required deliveries are marked as “Delivered” with their delivery times available. For truck placement, a list of packages on each truck is printed.

I3: OTHER POSSIBLE ALGORITHMS

When I originally started this project I made the poor assumption I could brute force the truck paths. A Brute Force Algorithm would have provided the optimal path by running every possible permutation. The main problem with this is the time complexity of O(n!). For every new package added to a truck, the number of permutations just grew by the factorial. When I started I assumed this would not become too large with at most 16 packages a truck, however, after 40 minutes of calculating and 98% RAM usage (I also made the mistake of storing all 20,922,789,888,000 permutations in a list), I realized even at this small scale, a Brute Force Algorithm is not remotely practical. (Oolawanle)

Another method I attempted was using the Dijkstra Algorithm to find the smallest paths and add those to all vertices/addresses until each address has been reached. This method has a time complexity of O(E log V), where E is our total Edges or path segments and V is our total vertices or addresses. This method is very similar to Nearest Neighbor, except it finds the shortest path for all vertices, not just the current vertex. Both are Greedy Algorithms, algorithms that take a global problem and split them into smaller local problems. For me, implementing a Dijkstra Algorithm was significantly more complex than Nearest Neighbor, since a Graph, and Vertex class would have to be created.

J: DIFFERENT APPROACH

One different approach that could enhance the project would be the automatic loading of packages onto each vehicle. My current implementation requires each truck to manually have packages added to them. Creating a method of loading packages based on required delivery time, and required start time would make the process easier and may render a more efficient path than the ones possible based on my chosen ‘packages on truck’.

K1: VERIFICATION OF DATA STRUCTURE

Verify Total Miles: Available on Program Execution

Verify All Packages Delivered on Time: If you request all packages at any time, the delivery time for each is printed as well

Verify Delivery Specifications: When all packages are printed at the required delivery times, the required deliveries are marked as “Delivered” with their delivery times available. For truck placement, a list of packages on each truck is printed.

K1A: EFFICIENCY

Additional packages in the program do not affect the Hash Table lookup function. Because of the 1-1 mapping, the only additional work at lookup is calculating the hash, which is not based on the input size, and therefore does not affect time complexity O(1).

K1B: OVERHEAD

The space complexity of the Hash Table is O(n), since for every additional package input, an additional portion of memory is required to store the KeyValue pair.

K1C: IMPLICATIONS

The addition of trucks or cities would not affect the Hash Table time or space complexity, since both of these variables are not inputs into the Hash Table. However, these do affect the actual calculation times and storage used by the program, such as when each truck location has to be found in the location data, and distance data. Time complexity and Space Complexity are strictly based on the input variables of the data structure or algorithm.

K2: OTHER DATA STRUCTURES

The main structure adaptation that I would take into consideration would be a modified Hash Table that deals with hash collisions more effectively. The current package input load does not necessitate this, however, if more packages were required to be stored, we would need some form of collision resolution. The simplest implication would be Linear Probing which tries to take the very next available position in the table. Another method to reduce collisions is to change the modulus value of the Division Method hash function. Currently, the modulus is the length of the table. Ensuring this number was a prime number would increase the uniformity of the structure, decreasing the chance of collisions. (Data Structures Handbook: Hash Tables)

A structure I would not recommend would be a linked list. The main positives of a linked list compared to a Hash Table are duplicate values and quick insertions and deletions. If we were a very specific delivery company, only delivering a few types of nonunique packages, having the ability to deliver Package A to Address 1, or 2 would make this more lucrative. Additionally, if we were needing to remove and add packages from the main Hash Table structure it may be more lucrative to use a Linked List. In our situation, neither of these cases apply making the Hash Table a better option. (Data Structures Handbook: Linked Lists) (Open4Tech Team)

L: SOURCES

aayuski2402 (2022, June 7). Applications, Advantages, and Disadvantages of Hash Data Structure.GeeksforGeeks.https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-hash-data-structure/

Data Structures Handbook: Hash Tables. (n.d.). Data Structures Handbook. https://www.thedshandbook.com/hash-tables/

Data Structures Handbook: Linked Lists. (n.d.). Data Structures Handbook. https://www.thedshandbook.com/linked-lists/

Iyer, J (2021, December 6). Time and Space Complexity of Hash Table Operations. OpenGenus IQ: Computing Expertise & Legacy. https://iq.opengenus.org/time-complexity-of-hash-table/

Oolawanle, J. (2022, October 5). Big O Cheat Sheet – Time Complexity Chart. FreeCodeCamp.org. https://www.freecodecamp.org/news/big-o-cheat-sheet-time-complexity-chart/

Open4Tech Team (2019, August 5). Array vs. Linked List vs. Hash Table. Open4Tech. https://open4tech.com/array-vs-linked-list-vs-hash-table/