Core Algorithm Overview

A: Identify a named self-adjusting algorithm

For this program, I used a Greedy Algorithm to deliver the packages. I use the distance information given in the WGUPS Distance File to determine the shortest path from where the delivery truck is currently at to the addresses of the packages in the truck. The truck will go to the nearest address, drop off the packages (removing them from the list of packages in the truck), and then run the algorithm again to find the next nearest address from the trucks new position.

B1: Explain the algorithm’s logic using pseudocode

The parameters that the algorithm takes are the following:

1. distanceData = the distance information showing the distance from one address to another.
2. addressData = the list of addresses in the same order as the distance information. This is used to find the index of a specific address.
3. Truck = the current truck the algorithm is being used on.
4. totalMiles = the total amount of miles traveled by the truck.
5. Packages = the list of packages currently on the truck.

First, the algorithm checks to make sure the list of packages isn’t empty, if it is empty, it will return the empty list of packages. Next, if the list isn’t empty, it will assign the current address of the truck to a variable (add1). Then, for all of the packages in the list of packages in the truck, it will check the distance from the delivery address of the packages to the current address of the truck and compare them to each other to find the shortest distance. After the shortest distance is found, the algorithm changes the truck’s current address to the packages address, removes the package from the truck’s package list, and updates the package to tell when the package was delivered. Finally, the algorithm is run again, finding the next shortest destination from the trucks new address. This is ran continuously until the package list is empty, then returns the empty list and the total distance traveled for that truck.

minDistanceBetween(distanceData, addressData, truck, totalMiles, packages):

if packages is empty:

return packages

else:

add1 = truck’s current address

add2 = 0

minimum distance = 50

for all packages in package list:

if distance between current address and package address is less than minimum distance:

minimum distance = distance from address

add2 = package address

truck’s current address = package address

totalMiles = totalMiles + minimum distance

remove package from package list

minDistanceBetween(distanceData, addressData, truck, totalMiles, packages)

return totalMiles

B2: Programming Environment

The programming environment that I used was PyCharm 2022.2.4 (Community Edition). I used this programming environment for creating the program and testing it. This version of PyCharm IDE was running Python version 3.11.1.

B3: Evaluate the space-time complexity

The Greedy Algorithm has a space-time complexity of O(N^2). The algorithm used to load the trucks with the packages has a complexity of O(N). The function that updates the package information with the time it was delivered has a complexity of O(1). The function that determines the amount of minutes to get from one address to the next has a complexity of O(N).

B4: Explain the capability of your solution to scale and adapt

This algorithm will be able to scale to multiple trucks and addresses and packages. Since the algorithm works with a specific truck and that truck’s packages, you would only have to update the distance files and package list. The algorithm will read from that information and will still be able to determine the shortest distance from the truck’s current position and the packages delivery address.

B5: Discuss why the software is efficient and easy to maintain

This software is efficient because it can read information from a CSV file to quickly access package information. It also can quickly give you the information that you are needing on a specific package, or on all packages in the list. The software is easy to maintain because I have commented on what variables are exactly for and what each function is meant to accomplish. So if there is an error with a function, it will be easier to find what exactly when wrong by looking at the comments.

B6: Strengths and Weaknesses of the Hash Table

The major strength that I had with the hash table is that it can easily organize the package information and makes it easy to find that information using the packages key. This ultimately made it very simple to store and search for package information. A weakness of the hash table, however, was it made it a little more difficult to find specific information about a single package. Going with a more object-oriented approach had me trying to search for a specific variable of the package, mainly the address, and it was made a little more difficult with the hash table because I had to change the way I would normally get that information. I had to make the search function of the table be able to return specific information about the package, like it’s address.

I: Justify the core algorithm

I think that this algorithm is one of the better ways to do this type of program. One strength of the algorithm is that it will find the shortest path first, and when the program is focused on the amount of miles traveled, that is important to know. This algorithm will also help determine how the packages should be loaded, mainly by looking at the special notes for the packages. If packages needed to be delivered together, or by a certain time, it will determine which truck it should be on.

This algorithm had completed its route in 120.4 miles. This isn’t the most efficient possible, but all packages were delivered on time and following any special notes that were with the packages. Two other algorithms I could have used would be the dynamic programming approach or the self-adjusting heuristic algorithm. The dynamic approach would be different in that I could store paths along the route and check to see if going to a location would be faster by traveling somewhere else first, then to that location. Doing this algorithm, however, would take up a lot of space, but would most likely yield a shorter path. The self-adjusting heuristic algorithm would be different because I could start at the hub and load packages based on the shortest distance from the hub, and as it traveled, the truck would move to the next shortest location. My algorithm and the heuristic are similar in that the shortest path is chosen next.

J: What you would do differently

Looking back, I think I would have changed the distance data and the address data to be CSV files as well. That would definitely help with the expandability of the program, as you would just have to change out the file, instead of redoing the two variables.

K: Justify the Data Structure

I think that the hash table is good for this program because the searching function is a great way to get a specific packages information. It is also very flexible and easy to work with. I think that the expansion of the packages in the list would not effect the searching function of the hash table very much, it will still provide quick access to information about the packages. Using a chaining Hash Table made it very quick for searching, inserting, and removing from the table. Most of the operations even have a constant complexity of O(1). The space requirements would be a little higher because of the change in packages, however the hash table is very space efficient. I think that changes to the number of trucks or the number of cities would affect the search time and space usage a little bit, as the data files and list of truck’s packages would be bigger to search through, but the hash table will still provide a fast look-up time.

Two other data structures that I could have used would be the Binary Search Tree and the Graph. The Binary Search Tree would have been different in that I could have presorted the packages and access them easier through the tree. The Graph would have been different because I could have set the distances as vertices on the graph and could have grouped packages that had similar vertices and traversed the graph from one vertex to the next 16 times to account for the maximum load.

Sources:

Learn.zybooks.com. (n.d.). *zyBooks*. [online] Available at:

https://learn.zybooks.com/zybook/WGUC950AY20182019