Charles Brixey

Student ID: 004546211

Data Structures and Algorithms II - C950

July 29, 2024

**NHP3 — NHP3 TASK 1: WGUPS ROUTING PROGRAM PLANNING**

**Stated Problem:**

Implement an algorithm to route delivery trucks that will allow you to meet all delivery constraints while traveling under 140 miles.

**A. Identify a self-adjusting algorithm**

To solve this task I created a greedy algorithm called determineNextDestination(), which executes during every iteration of deliverPackages(). determineNextDestination() compares the truck’s current address against the destination addresses of all packages onboard. Upon arriving at the new address, relevant metrics are updated for both the truck and packages in deliverPackages(). The packages that have a destination address matching their current address are then removed from the truck and considered delivered.

The reason the algorithm is ‘greedy’ is because it will find the lowest cost(miles) of all destinations and traverse to that address without considering other implications. For instance, travelling to the nearest package destination address might not be advantageous for overall mileage.

determineNextDestiation() is self-adjusting because it must find the closest package destination, which changes depending on the truck’s current location and the packages onboard that truck.

Algorithm Overview:

1. While packages exist on the particular truck execute determineNextDestination()
2. Compare the current address of the truck against all package destination addresses
3. Use the determineCost() method to find the lowest cost(miles) of the package destination addresses
4. Traverse to the lowest cost destination
5. Remove packages that have a destination address of the new current address using the removePackges() method

A computer screen with text

Description automatically generated

A computer screen shot of a program code

Description automatically generated

**A computer screen shot of a program code

Description automatically generated**

A computer screen shot of text

Description automatically generated

**A computer screen shot of a program code

Description automatically generated**

**B.   Identify a self-adjusting data structure, such as a hash table**

The self-adjusting data structure I used was a hash table by creating a HashMap class. HashMap’s add method takes the following parameters: self, Id, destinationAddress, city, state, zip, deadline, weight, and notes. From these parameters HashMap creates a package object that it maps to a map array index using id-1. This is powerful because HashMap methods like add, get, and delete operate in O(1) runtime. We won’t be using delete in our project but add and get will be essential in creating packages and grabbing data pertaining to those packages.

The hash table is self-adjusting because the HashMap class takes a size parameter. Size is derived from the number of packages scanned in from the CSV file. In this case, we had 40 packages. However, the HashMap class could accommodate much more.

A computer screen shot of text

Description automatically generated

**C.  Write an overview of your program in which you do the following:**

**1.  Explain the algorithm’s logic using pseudocode.**

while packages exist on a truck:

determineNextDestination():

truck’s current address = some address

lowest cost = placeholder data

next destination = placeholder data

loop through all packages:

compare truck’s current address to package destination address

store cost between current address and package destination address

if(new cost < lowest cost):

replace lowest cost with new cost

replace next destination with new destination

return new destination with the lowest cost

truck travels to new destination

update truck/package metrics

package(s) are now delivered

find index of delivered packages on the truck and remove them with removePackagesFromTruck()

**2. Describe the programming environment you will use to create the Python application, including *both* the software and hardware you will use.**

The coding environment I will use is Visual Studio Code on my Home PC. I am also using the Python extension for Visual Studio Code to run python files.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**3.  Evaluate the space-time complexity of *each* major segment of the program and the entire program using big-O notation.**

determineNextDestination(): Looping through all the packages onboard the truck to compare their destination address using determineCost() with the truck’s current address is an O(n^2).

determineCost() – O(n): Determining the cost from two locations involves first seeing which address array is longest. This is hard to visualize without looking at the CSV Distance Table, but essentially the longest array has the route to the shorter array. Once this is determined, you loop through the addresses to find the index of the new or old address depending on which array you chose. Then index helps calculate the mileageCost, which is then returned. This operation is O(n).

removePackagesFromTruck() – To remove packages we use a while loop to cycle through all packages onboard of the truck and see if their current address equals their destination address. If that is true, we record that index, and then pop off the package at then index and return iterating variable i back to zero and begin the process again. This is an O(nlogn) operation because every package deleted resets variable i back to 0, thus making the runtime worse than O(n).

deliverPackages() – deliverPackages() will execute as long as there are packages on the truck, which by itself is an O(n) operation. However, deliverPackages() is calling determineNextDestination(), determineCost(), handleUpdates(), removePackagesFromTruck(), and handleLastTruckRide()(if you’re truck 1) per each package. Because determineNextDestination() is O(n^2), we can say that deliverPackages() is O(n^3).

**4.  Explain the capability of your solution to scale and adapt to a growing number of packages.**

determineNextDestination() has a runtime O(n^2) and is nested inside delieverPackages(), which has a runtime of O(n^3). If we are delivering on average 40 packages a day, this algorithm will be sufficient. However, this algorithm would not scale well with a company like Amazon that delivers an average of 1.6 million packages a day. Assuming an input of 1.6 million, that would be 1.6million^3 or 4.096e+18 operations!

**5.  Discuss why the software design would be efficient and easy to maintain.**

This program would easy to maintain because the deliverPackages() and determineNextDestionation() methods are fairly straightforward. They loop over the packages in a particular truck comparing route costs. Once you have a route selected, you traverse to that destination. Then metrics are updated for the truck and its remaining packages. Finally, all packages whose current address matches their destination address are considered delivered and are removed from the truck. Conceptually, the most challenging aspect to understand would probably be the address table arrays derived from the Distance Table CSV file. But this would quickly make sense once the software developer could see distance table.

**6.   Describe *both* the strengths and weaknesses of the self-adjusting data structure (e.g., the hash table).**

The main strength of the hash table is its O(1) runtime for add, get, and delete methods. Performing these tasks in an array is much slower. For instance, a get method in an array would have a runtime of O(n) because you’d have to scan the entire array looking for a specific package. Likewise, a delete method would also be O(n) in an array (unless you deleted the last element) because it would cause elements to shift. An add method in an array would be more-forgiving with an O(1) runtime but only if you added elements to the end of an array. If elements were inserted somewhere in the array besides the end, it would result in O(n) runtime because of shifting. But if you had millions of packages that could be problematic.

The weakness of the hash table is the lack of order amongst the packages. For instance, in a traditional array the first package added would be in the front. Likewise, the last package added would be at the end of the array. In the hash table we are adding packages based on their id, which works in this case because the package ids are in order from the CSV file. However, imagine if the package ids added were scanned out of order. There would be no way to know which packages were added first or last.

Another weakness of this hash table is inability to deal with collisions. If a package was mistakenly given the same id as another, the add method would override the initial package with that id. Thus, packages would be erased.

**7.  Justify the choice of a key for efficient delivery management from the following components:**

•   delivery address = destination address. I wanted to be clear this was a destination with a cost of travelling to.

•   delivery deadline = deadline.

•   delivery city = city.

•   delivery zip code = zip.

•   package ID = id.

•   package weight = weight.

•   delivery status = status. All package statuses are initially set to “AT THE HUB.” Although, you’ll see in my code that I change timestamps for late packages to alter this initial setting.

A computer screen shot of text

Description automatically generated

**D.  Acknowledge sources, using in-text citations and references, for content that is quoted, paraphrased, or summarized.**

“Applications, Advantages and Disadvantages of Hash Data Structure.” *GeeksforGeeks*, GeeksforGeeks, 28 Mar. 2023, www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-hash-data-structure/.

Chresfield, Kaelyn. “Pros & Cons of Hash Tables.” *Medium*, Medium, 21 Jan. 2019, medium.com/@kaelyn.chresfield/pros-cons-of-hash-tables-bc5d3097ffa7.