Charles Brixey

Student ID: 004546211

Data Structures and Algorithms II - C950

July 29, 2024

**NHP3 — NHP3 TASK 2: WGUPS ROUTING PROGRAM IMPLEMENTATION**

**A.  Develop a hash table, without using any additional libraries or classes, that has an insertion function that takes the package ID as input and inserts each of the following data components into the hash table:**

**A computer code on a dark background

Description automatically generated**

A screen shot of a computer code

Description automatically generated

**B.  Develop a look-up function that takes the package ID as input and returns *each* of the following corresponding data components:**

A screen shot of a computer code

Description automatically generated

A blue background with text

Description automatically generated with medium confidence

**C.  Write an original program that will deliver *all* packages and meet all requirements using the attached supporting documents “Salt Lake City Downtown Map,” “WGUPS Distance Table,” and “WGUPS Package File.”**

A screen shot of a computer code

Description automatically generated

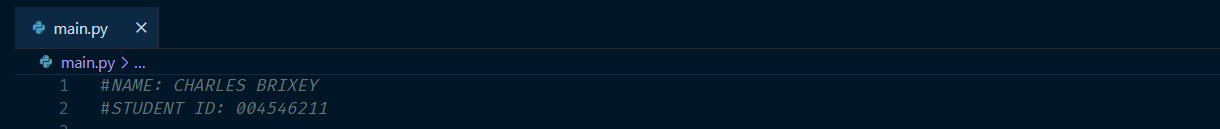
**A computer screen with text on it

Description automatically generated**

**A screen shot of a computer code

Description automatically generated**

**1.  Create an identifying comment within the first line of a file named “main.py” that includes your student ID.**

****

**2.  Include comments in your code to explain both the process and the flow of the program.**

(See code)

**D.  Provide an intuitive interface for the user to view the delivery status (including the delivery time) of any package at any time and the total mileage traveled by all trucks. (The delivery status should report the package as at the hub, en route, or delivered. Delivery status must include the time.)**

A screen shot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**1.  Provide screenshots to show the status of *all* packages loaded onto *each* truck at a time between 8:35 a.m. and 9:25 a.m.**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**2.  Provide screenshots to show the status of *all* packages loaded onto *each* truck at a time between 9:35 a.m. and 10:25 a.m.**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**3.  Provide screenshots to show the status of *all* packages loaded onto *each* truck at a time between 12:03 p.m. and 1:12 p.m.**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**E.  Provide screenshots showing successful completion of the code that includes the total mileage traveled by *all* trucks.**

A screenshot of a computer program

Description automatically generated

**F.  Justify the package delivery algorithm used in the solution as written in the original program by doing the following:**

**1.  Describe two or more strengths of the algorithm used in the solution.**

One of the main strengths of the ‘greedy’ determineNextDestination() algorithm is it’s simplicity. determineNextDestination() compares the destination address of all packages with the truck’s current address and chooses the option with the least cost. This is intuitive and the code isn’t something a beginner developer would have trouble understanding.

Another strength of the determineNextDestination() algorithm is its efficiency. One might mistakenly think that an “easy solution” would produce subpar results. However, the total mileage of 107.7 easily beats the 140 target. To be fair, I manually loaded the packages to game the system and lower the mileage. I would be curious to see how determineNextDestination() would fair in a randomly loaded packing scheme.

**2.  Verify that the algorithm used in the solution meets *all* requirements in the scenario.**

* The total mileage after all packages have been delivered is 107.7, which is less than 140
* All packages with specific time deadlines have been met
* Delayed packages (6, 25, 28, 32) are accounted for
* Package 9 with the address change is accounted for
* Packages that can only travel aboard truck 2 are accounted for
* Mileage from truck1 returning to the hub to swap drivers with truck 3 is accounted for

**3.  Identify two other named algorithms that are different from the algorithm implemented in the solution and would meet *all* requirements in the scenario.**

1. Brute Force

Another algorithm that could be implemented to solve this task would be the brute force method. Currently, determineNextDestination()looks for the lowest cost of the next destination (referred to as greedy approach). However, we could create a method that creates all possible permutations of all destination routes. Once all permutations have been created we would select the route with the overall lowest cost. Theoretically, this would give us the most efficient route. However, it would be resource intensive. In a hypothetical scenario with only 40 packages, it may suffice. But this algorithm would scale poorly.

2. Nearest-Neighbor

The nearest-neighbor algorithm works by traversing to the nearest destination addresses if they are unvisited by that truck. You continue this process until all nodes are visited, then return to your original starting point (the hub). Once you’ve finished visiting all nodes, all packages must have been delivered. If every truck had at least one package to every destination address this strategy might seem feasible. However, in our scenario, not all trucks needed to travel to each destination address. Therefore, the nearest-neighbor method would be too slow.

**a.  Describe how *both* algorithms identified in part F3 are different from the algorithm used in the solution.**

The brute force method differs from our greedy algorithm because it looks at all possible routes, as opposed to the route between the current address and next destination. Understandably, the brute force method is more resource intensive, however it will reveal the fastest route every time.

The nearest-neighbor method looks at immediate cost of destinations just like our greedy algorithm, however it also traverses to every unvisited location. This is problematic because we potentially would be needlessly accumulating miles for locations which had no associated package. Therefore, our greedy algorithm will be much faster.

**G.  Describe what you would do differently, other than the two algorithms identified in part F3, if you did this project again, including details of the modifications that would be made.**

If I was to do this project again, I’d probably make a more sophisticated determineNextDestination() algorithm. Currently, determineNextDestination() just looks at the nearest location without considering overall mileage. I would have liked to expand determineNextDestination() too possibly look at 2-3 ‘steps’ out. This wouldn’t be as cumbersome as a brute force method, that looks at all possible permutations of the routes. However, it would be more optimal at reducing mileage.

Another change I would have implemented would be an algorithm to perform adding packages to trucks. Presently, the trucks are manually loaded, which makes this project dramatically easier to achieve the less than 140 miles constraint. However, this was tedious because I spent a lot of time shuffling the packages to get an acceptable result.

Finally, it would have been cool to add some type of regex scanning on the notes of the packages. That seemed beyond the scope of this project and honestly quite time-consuming. However, it would be beneficial to group certain packages on certain trucks depending on their “late arrival time” for instance.

**H.  Verify that the data structure used in the solution meets *all* requirements in the scenario.**

* The hashMap class has an add method takes the package ID as input and inserts each of the following data components into the hash table: delivery address, delivery deadline, delivery city, delivery zip code, package weight, delivery status (includes delivery time)
* The hashMap class has a lookUp method that takes in package ID as input and returns each of the following corresponding data components: delivery address, delivery deadline, delivery city, delivery zip code, package weight, delivery status
* No python dictionaries were used in this program
* No additional libraries were used

**1.  Identify two other data structures that could meet the same requirements in the scenario.**

* Array
* Stack

**a.  Describe how *each* data structure identified in H1 is different from the data structure used in the solution.**

You can store package objects in an array. This would work, but trying to retrieve packages would take O(n) runtime instead of O(1). This is because the array index would not be mapped to the package id like in our hash table. Also, deletes would be O(n) as well because it would cause a shifting of the array. Adding to an array would take O(1) time if you always appended the next package to the end of the array. However, if you wanted to do insertion somewhere in the middle of the array or at the start it would take O(n) runtime. Overall, an array is definitely feasible but slower.

A stack would also work when we scan packages in from our CSV file. You can picture placing incoming packages in a stack at some warehouse. The last package in would be the first package placed on a particular truck. From there you could deliver the packages using a queue or another stack. It would be necessary to record which packages got placed on which trucks. Because once a package is popped off a stack or queue, it will be gone from the data structure. Obviously, you can see the pitfalls with this data structure compared to our hash table. Our insertions for a stack would always been O(1) runtime because we’d be appending the package to the end of the stack. Likewise, our gets would also be O(1), but we’d only be able to grab the last package in. This would be a nightmare for finding an optimal route to deliver our packages because we would travel to the next destination of the package on top of our stack.

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

Kuo, Marc. “Algorithms for the Travelling Salesman Problem.” *RSS*, 18 Apr. 2024, www.routific.com/blog/travelling-salesman-problem.