WGU C950 DSA2

NHP2 – NHP2 TASK1: WGUPS ROUTING PROGRAM

## A. Identify a named self-adjusting algorithm

I used a **genetic algorithm** to determine a near ideal route by randomly guess a subset of the total permutations of routes and mix the best routes (shortest) together and repeated this for a set number of times. The more iterations of this the shorter the route, but the longer it will take to run.

## B. Write an overview of your program

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

**Note: to make the application reproducible, a numpy.random.seed of 42 is used in Genetic.py**

Input parameters of truck, distance matrix, address dictionary (address are keys and index are values), hash table and an initial list of routes randomly selected from a numpy permutation of the truck’s package ids

Create a list of package ids to be loaded in a list.

Convert the list of package ids to the index of the address in distance matrix.

Randomly select 25 permutations of the route

Loop for N iterations:

Evaluate the route by taking a list of routes and calculating the total distance of the route, but the route will get a 100-mile penalty if any package is delivered after its deadline.

The shortest route will be added to the best list and to the parent list and a list of vector probabilities will be returned and will randomly be picked to add to the parent list until there are 4 routes.

Once there are 4 routes in the parent list, a mutation may be done (probably of 90%) by swapping the first part of one route with the second part of another route without repeating a stop, then possibility a swapping any of the two stops may occur (probability of 90%).

Loop

The best route will be returned along with the distance of the route.

This is repeated for each truck to determine the best route

1. **Describe the programming environment you used to create the application.**
   1. The application was developed with Python 3.11 with a virtual environment of pipenv in pycharm professional on a MacBook pro with an M1 processor. **\*\*Please run pipenv install in the terminal to ensure that all packages are installed (numpy, prettytable) or run “pipenv install numpy prettytable” in the terminal of pycharm.\*\***
2. **Evaluate the space-time complexity of each major segment of the program and the entire program using big-O notation.**
   1. Main.py
      1. Filling the hash table, line 54
         1. Time & Space = O(n)
      2. Distance matrix creation, line 57
         1. Time & Space = O(n)
      3. Address Index (dictionary), line 60
         1. Time & Space = O(n)
      4. Set up the parameters and create truck objects, line 64, 66 and 68
         1. Time = O(1)
         2. Space = O(n)
      5. Load truck easy, line 71
         1. Time & Space = O(1)
      6. Fill truck ids in the packages, line 74-76
         1. Time = O(n)
         2. Space = O(1)
      7. Determining the truck routes for 3 trucks, run and if the route is not good enough increase iterations, line 80 through
         1. Time = O(n), just for the while loop, it will have more complexities broken down below
         2. Space = O(1), only 1 total score will be stored
      8. Truck package indexes (convert\_pacakge\_id\_to\_address), lines 83, 97 and 116
         1. Time = O(n)
         2. Space = O(n)
      9. Genetic algorithm, lines 85, 99 and 118
         1. Time = O(n^3)
            1. Fitness is the only method O(n^3) because of the translation of address index to package id. Crossover and address\_index\_to\_package\_id are O(n^2)
         2. Space = O(n)
      10. Update the truck finish time, line 93
          1. Time & Space = O(1)
      11. Setting the delivery times of each package lines 139-141
          1. Time = O(n^2)
          2. Space = O(0), it is just updating memory that is already reserved
      12. Display the data at a time, line 175
          1. Time = O(n)
          2. Space = O(n^2), create table\_list and the package\_id\_list, both which are the size of the number of pacakges.
3. **Explain the capability of your solution to scale and adapt to a growing number of packages.**
   1. The program can scale on the time complexity with the genetic algorithm by trading the number of iterations for time. It is overall an O(n^3) time complexity, but it is strongly related to the number of iterations that is asked in the beginning. The data matrix will grow space complexity at O(n^2), but this is not bad for small to medium sized areas.
4. Discuss why the software is efficient and easy to maintain.
   1. The software produces a pretty good route distance that will be more optimal than nearest neighbor. With just 100 iterations most of the routes, if the seed of 42 is not used, will be under 95 miles. The software does not determine what packages go in each truck, but packages with the same address are grouped in the same truck then there will be fewer stops. The maintainability of the software is ok, since there is type hinting and some tests that can be used to make sure that the program can be tested.
5. Discuss the strengths and weaknesses of the self-adjusting data structures (e.g., the hash table).
   1. A hash table is a good data structure for insert and lookups with O(1) being the best possible time complexity and O(n) for the worst for lookups. If the table is under sized or the size of a number that has many divisors, then a weakness will be that a collision. The simple way that I handled collisions was to create a list of items at that hash, but with a size of a prime number there will be minimal collisions.

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

1. Create an identifying comment within the first line of a file named “main.py” that includes your first name, last name, and student ID.
2. Include comments in your code to explain the process and the flow of the program.

## D. Data structure, such as a hash table, that can be used with the algorithm identified in part A to store the package data.

1. Explain how your data structure accounts for the relationship between the data points you are storing.
   1. The hash table is used to hold all the packages that are available to the genetic algorithm. It is used to convert the package ids to address indexes and back to package ids. This is used to generate the route and then to quickly access each package object and fill in the delivery times and what truck was used to deliver the package.

## E.  Develop a hash table, without using any additional libraries or classes, that has an insertion function that takes the following components as input and inserts the components into the hash table:

•   package ID number

•   delivery address

•   delivery deadline

•   delivery city

•   delivery zip code

•   package weight

•   delivery status (e.g., delivered, en route)

## F.  Develop a look-up function that takes the following components as input and returns the corresponding data elements:

•   package ID number

•   delivery address

•   delivery deadline

•   delivery city

•   delivery zip code

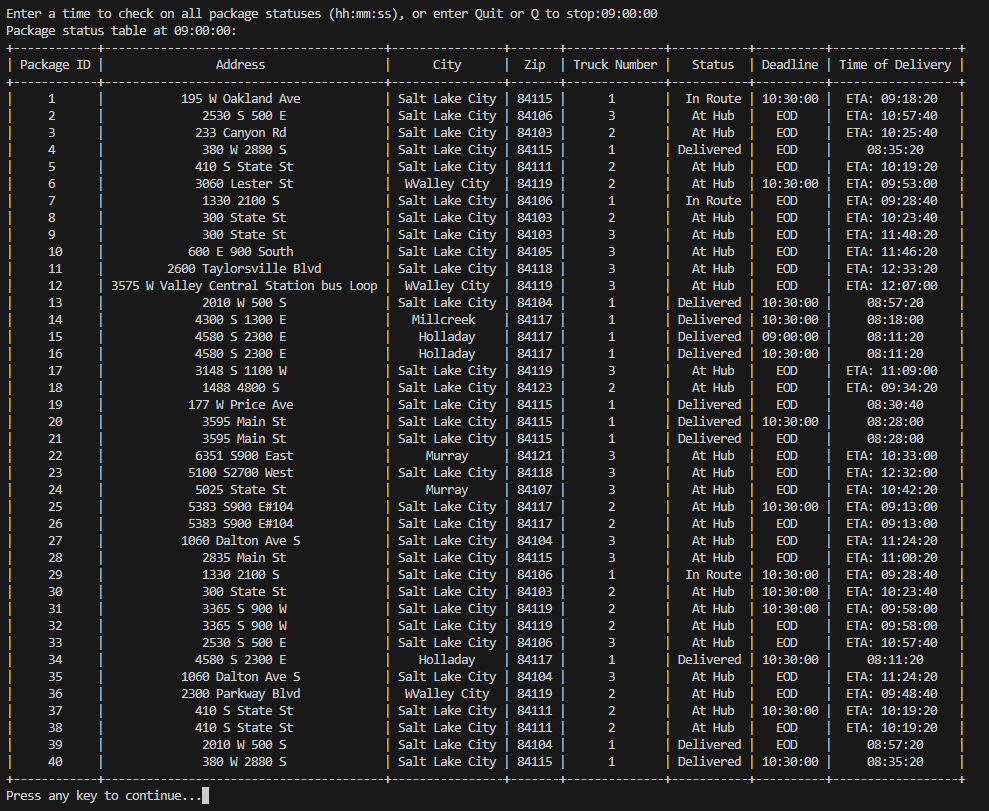
•   package weight

•   delivery status (i.e., “at the hub,” “en route,” or “delivered”), including the delivery time

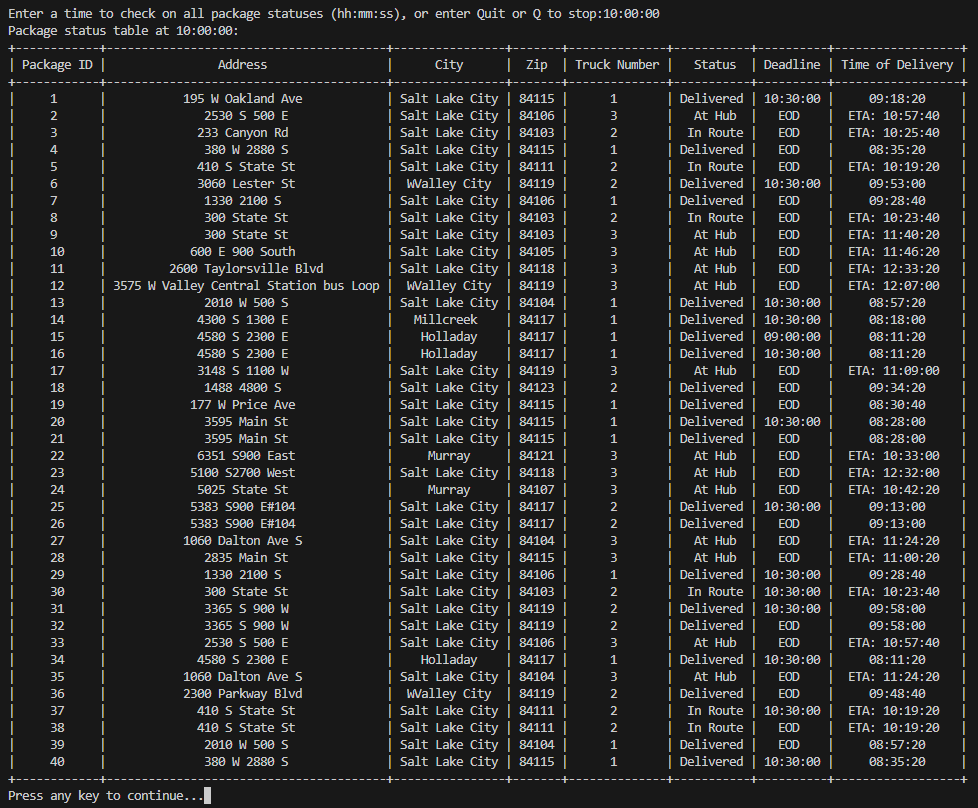
*Note: Your function should output all data elements for the package ID number.*

## G.  Provide an interface for the user to view the status and info (as listed in part F) 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.)

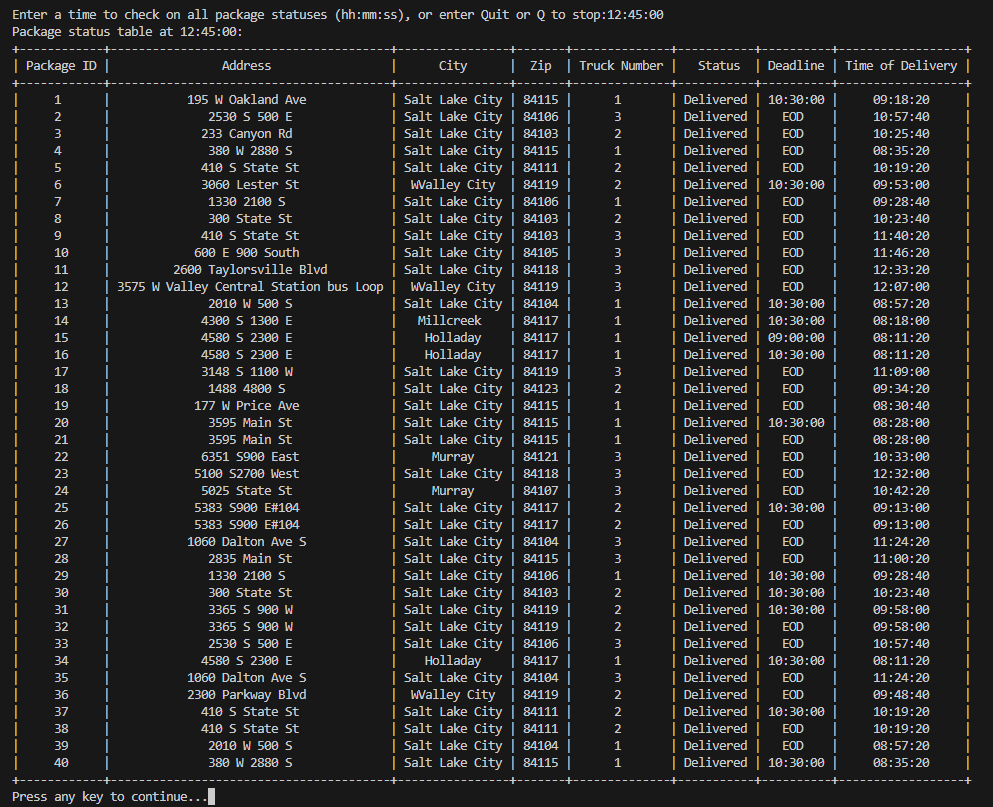
1.  Provide screenshots to show the status of all packages at a time between 8:35 a.m. and 9:25 a.m.



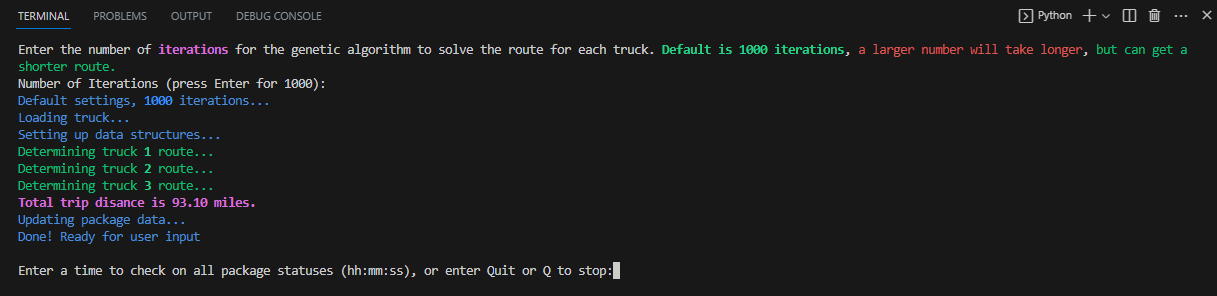
1. Provide screenshots to show the status of all packages at a time between 9:35 a.m. and 10:25 a.m.



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



## H.  Provide a screenshot or screenshots showing successful completion of the code, free from runtime errors or warnings, that includes the total mileage traveled by all trucks.



## I.  Justify the core algorithm you identified in part A and used in the solution by doing the following:

1.  Describe at least **two** strengths of the algorithm used in the solution.

* A genetic algorithm can very quickly provide a good enough answer for very difficult problems and can be parallelized easily. It was not parallelized in this application, but it can be distributed.
* What I liked most about it is how it was simple to create the fitness of the route. It tests weather the package was delivered on time and if the route was not short enough, I can just run it again with more iterations.

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

* The screen shot for G3 shows that all the packages were delivered by 12:33 pm and that the route was 93.10 miles.

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

* There is the nearest neighbor algorithm that will just go to the nearest location and brute force algorithm that would try every possible combination.

a.  Describe how each algorithm identified in part I3 is different from the algorithm used in the solution.

* The nearest neighbor is very simple where it only looks for the next stop and then once the route has been completed can it be judged. It works fast but will not be a near optimal solutions.
* The brute force method will provide an optimal solution but will not scale with additional packages. Even the 40 packages would take days or weeks to run (I tried it!)

J.  Describe what you would do differently, other than the two algorithms identified in I3, if you did this project again.

* If I were to do this again I would try to do a dynamic programed algorithm and use and adjacency list instead of an adjacency matrix. The dynamic algorithm would be fast and get a new optimal solution and the adjacency list has a much lower space complexity compared to a matrix.

## K.  Justify the data structure you identified in part D by doing the following:

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

a.  Explain how the time needed to complete the look-up function is affected by changes in the number of packages to be delivered.

* Using a hash table the look up complexity, assuming collisions would be O(number of packages / hash table size). Since I would use a prime number for the size of the hash table this will more evenly distribute the packages and the most it would be is around O(5) operations, 1 to find the hash, then 4 to find the package in the inner list. Overall it can be considered an O(1) operations for hash table lookups.

b.  Explain how the data structure space usage is affected by changes in the number of packages to be delivered.

* The space complexity of a hash table is O(n), as you add more packages, the data structure only adds the one package. There is a fixed space cost of the table, but those do not grow, so it would be O(n + number of hashes).

c.  Describe how changes to the number of trucks or the number of cities would affect the look-up time and the space usage of the data structure.

* The number of trucks would not add to the time complexity since it would be smaller routes to compute if the number of packages stays the same. The space complexity would not increase with the number of trucks since you would not add more trucks than packages, this would be a poor business decision.
* If the number of cities increase, then the data matrix space complexity will be O(n^2) since the table is square and the distance from every city to the new one will need be added along with the distance to every city would also be added.

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

a.  Describe how each data structure identified in part K2 is different from the data structure used in the solution.

* Instead of a using a distance matrix, an adjacency list could have been used. While the adjacency list does have a slower loop up O(n), but the space complexity is O(n) on average. However, if it contains edges to every other vertex then the space complexity is the same as the matrix O(n^2). So, in the case of a Hamiltonian graph a adjacency matrix is preferred for the quicker look up.
* A linked list could have been used to store all the packages. This would have been simpler, but instead of a near O(1) lookup it would be O(n), unless they were sorted by id then a binary search could have been used to with O(log(n)) time complexity.

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

Wikipedia contributors. (2023, July 2). Travelling salesman problem. In *Wikipedia, The Free Encyclopedia*. Retrieved 01:36, July 12, 2023, from <https://en.wikipedia.org/w/index.php?title=Travelling_salesman_problem&oldid=1163005535>

Wikipedia contributors. (2023, July 1). Genetic algorithm. In *Wikipedia, The Free Encyclopedia*. Retrieved 01:36, July 12, 2023, from <https://en.wikipedia.org/w/index.php?title=Genetic_algorithm&oldid=1162821022>

## M.  Demonstrate professional communication in the content and presentation of your submission.