**`Core Algorithm Overview**

**Stated Problem:**

The purpose of this project is to create an algorithm using Python to route delivery trucks that will allow you to meet all delivery constraints while traveling under 140 miles. You will then describe and justify the decisions you made while creating this program.

**A. Self-adjusting algorithm used to create the program**

Nearest Neighbor Algorithm

**B. Algorithm Overview**

The routing program is created in the following manner:

1. Create a graph to hold vertices representing each delivery stop.
2. Create a package object for each package provided in the sample file
3. Create a hub and vertex object for each delivery address provided in the sample file.
4. Add the vertices to the graph and connect each vertex using weighted undirected edges.
5. Load each truck using the nearest neighbor algorithm while sorting packages with special notes once they have been discovered. Once a package is loaded, the associated vertex is added to a truck’s available travel path.
6. Once each truck is active, it will go to the closest vertex in its available travel path and deliver the package and remove that vertex from its list of available paths.

**B1. Nearest Neighbor Algorithm Using Pseudocode**

Nearest Neighbor Algorithm

Initialize empty visited location list

Get the current location of the truck

While there are still unvisited locations and the truck has less than 16 packages

For all adjacent locations

If the adjacent location is not the visited location and it’s distance is less than the current location with the nearest distance

Remember the location as the closest location and it’s distance

Add the closest location to the truck’s available locations

Load the truck with the packages for the closest location

Add the closest location to the list of visited locations

Change the truck’s current location to the closest location

**B2. Programming Environment**

IDE – PyCharm Community Edition 2020.3.3

Python Interpreter – Python 3.9

OS – Windows 10

RAM: 8GB

Environment Variables: C:\Python39\Scripts, C:Python39\, %PyCharm Community Edition%, D:\Program Files\JetBrains\PyCharm Community Edition 2020.3.3\bin

**Please note: The program is designed to run in the Windows command line on a 1080p monitor. The GUI may look/behave strange if you run the program in the IDE terminal or on a Mac/Linux.**

**B3. Major Segments**

* **Startup**
  + Initializes the Graphical User Interface (GUI) in the form of a command-line interface (CLI).
    - **Space-Time Complexity - O(1)**
  + Creates a Graph data structure to represent our delivery locations. Loads the information located in the package and distance files as package, truck, hub, and vertex objects.
    - **Space-Time Complexity – O(n^3)**
  + Load the trucks with packages utilizing the nearest neighbor algorithm.
    - **Space-Time Complexity – O(n)**
  + Begin a loop that will cause a truck that is active to deliver a package at the current vertex and then travel to the closest vertex in its available vertex path.
    - **Space-Time Complexity – O(n)**
  + After each loop, update the GUI to reflect new events, package status, and truck details.
    - **Space-time Complexity: O(n)**

Operation time would depend mainly on the number of vertices to be created. The program indicates that there are no directional limitations on the distance, and as a result, each vertex is linked with all other vertices using weighted edges. During creation, we must use two for loops to obtain the distance for each entry in the distance file resulting in the program’s worst-case space-time complexity of O(n^3).

**B4. Ability to Scale:**

The program’s ability to scale would be limited up to 48 packages. Due to the requirements of the program, we only have three trucks with the capability of storing 16 packages each. The loading of the trucks only loads one truck at a time until they have 16 packages or until there are no more vertices left to discover. If the package count is greater than 48, truck count is greater than 3, or driver count changes. The loading and special\_event functions would have to be recreated to accommodate the changes. However, searching packages will continue to scale with additional packages due to the search function being implement with a HashMap which on average gives a space-time complexity of O(logN) and a worst case scenario of O(n).

**B5. Maintainability**

The program is efficient due to it utilizing a nearest neighbor algorithm to determine the next package to load and the next destination to deliver a package to.

The program is easy to maintain due to it being very modular with comments at each major action. Each major function reads like a set of instructions with each action listed as a separate function/method. This allows the major function to be adjusted and understood easily.

**B6. Strengths and Weaknesses of the Self-Adjusting Data Structures:**

(**K1.A Explain how the time needed to complete the look-up function is affected by changes in the number of packages to be delivered)**

The current strengths of the hash table include an efficient search of key, value pairs with an average space-complexity of O(logN) and a worst-case space-time complexity of O(n). However, the worst-case space-time complexity would be very unlikely unless the number of entries is extremely small

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

The weakness of the current table is the inability to append values to the key to create a list to show a key, [values] pair. This would have provided increased efficiency in loading packages by searching for a list of packages based on value instead of iterating through the entire package file during certain functions.

**(I1. Described two strengths of the algorithm used in the solution)**.

The current strength of the Nearest neighbor algorithm is that it gives an efficient way of choosing the cheapest delivery location and results in a low mileage and delivery time However, a major weakness is that it is difficult to account for special instructions or delivery deadlines. There are various ways that the algorithm can be improved by only accepting priority packages up to a certain expected mileage. The fact that the program delivers all packages before the delivery deadline and below the mile limit is a side effect of the algorithm, but not a hard-coded intention.

(**I2. Verify that the algorithm used in the solution meets all requirements in the scenario)**

The algorithm sorts packages into the truck using python’s built in substring recognition. Package 9 is delivered after the event to change its address to the correct address. This is hardcoded into the program’s loading and path generation.

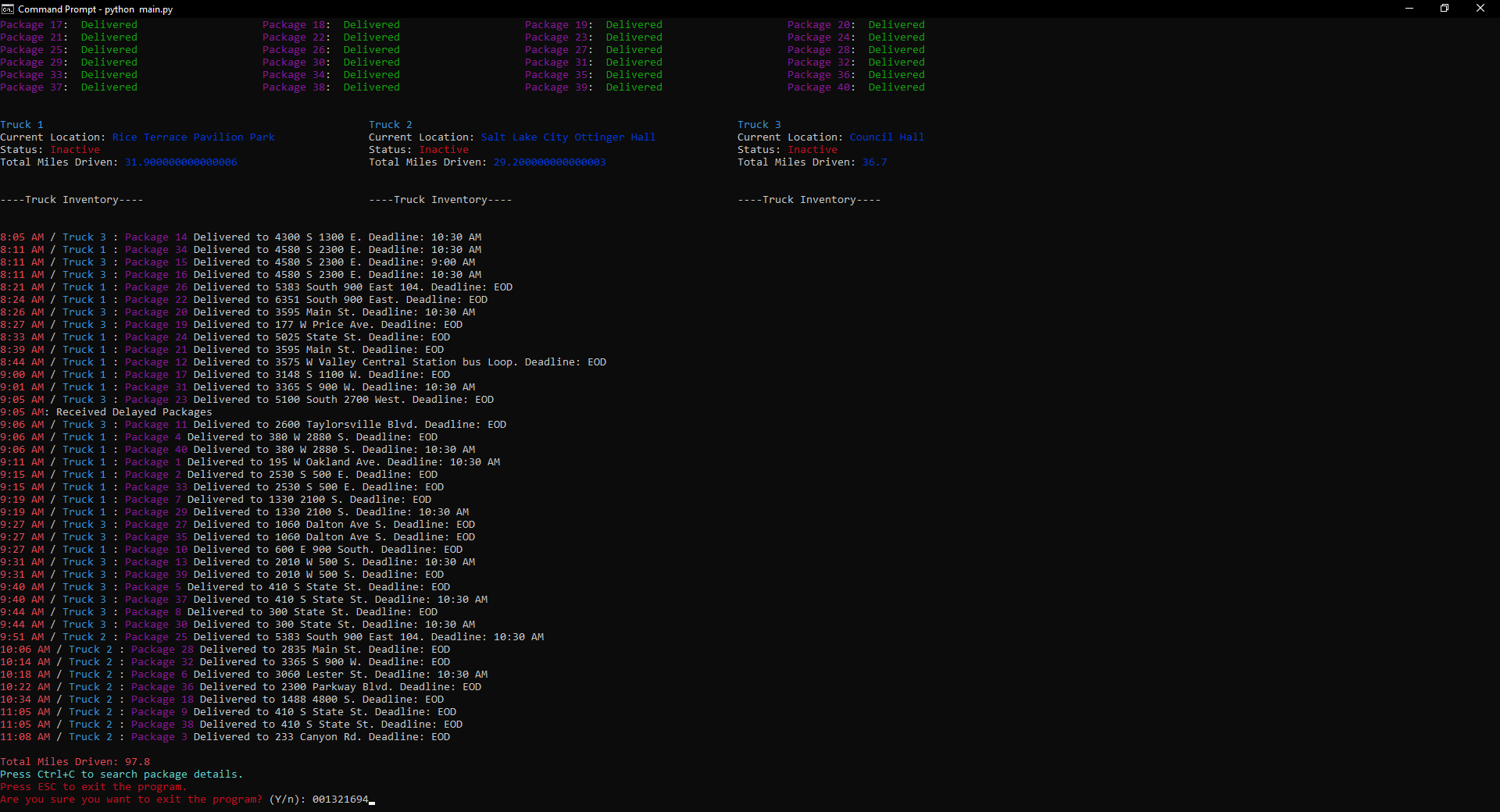
**D.  Identify a self-adjusting data structure, such as a hash table, that can be used with the algorithm identified in part A to store the package data.**

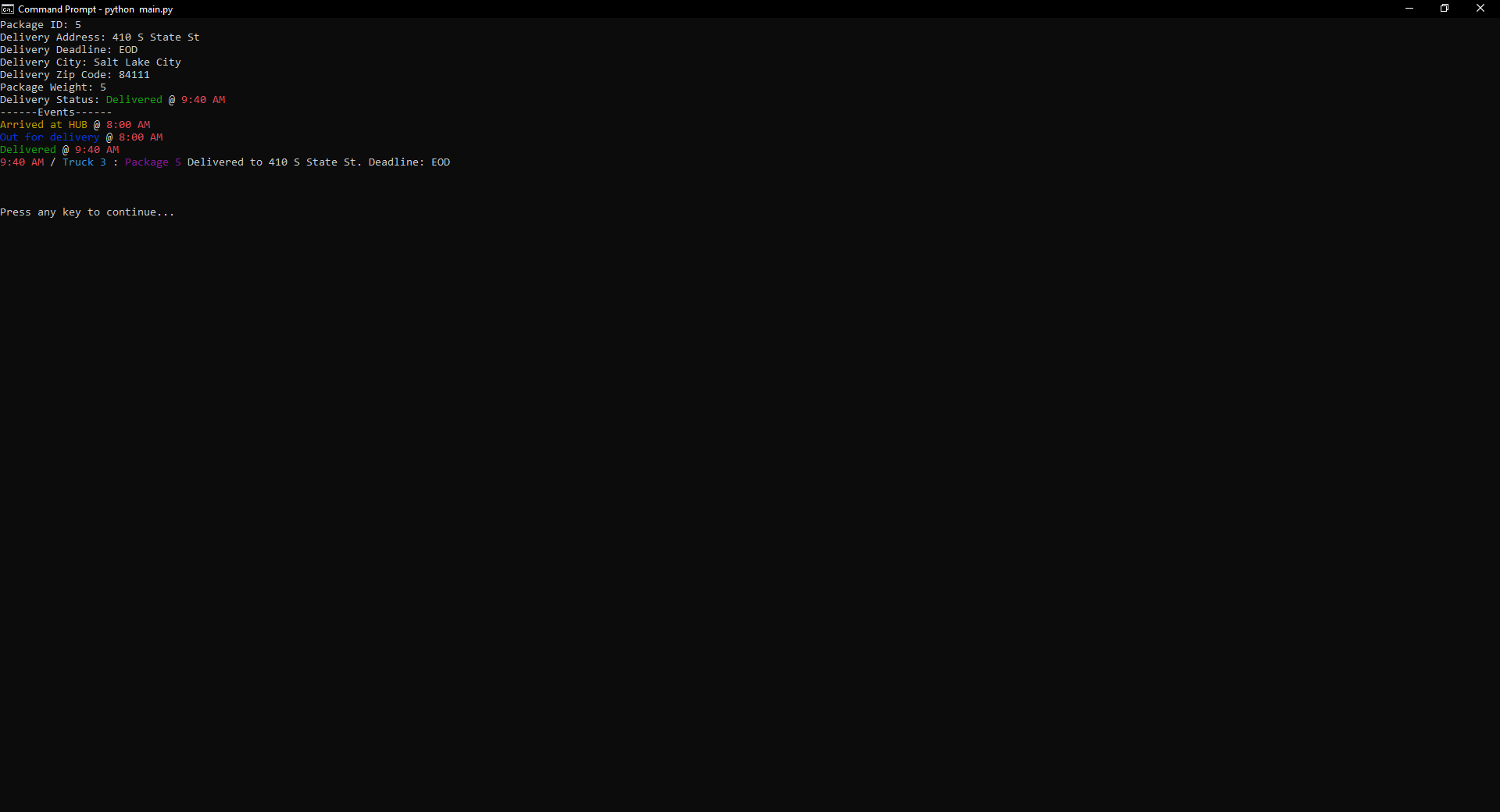
**(D1. Explain how the data structure accounts for the relationship between the data points you are storing)**

The Package, Truck, Hub, and Vertex objects were all stored in a hashmap using the associated ID (truck, package), address (truck, package, hub), or label/nae (hub, label). When searching for a specific object, the hashmap is used to use information from one object to find another. The hashmaps also act as a way to provide object persistence since the object is being stored as a value for future retrieval.

**G 1 – 3. Provide an interface for the user to view the status and info of any package at any time and the total mileage traveled by all trucks.**

**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.**





**I3. Identify two other named algorithms, different from the algorithm implemented in the solution that would meet the requirements in the scenario.**

1. Dijkstra’s Shortest Path
2. Floyd Warshall Algorithm
   1. **Describe how each algorithm identified in part I3 is different from the algorithm used in the solution**

The Dijkstra’s Shortest Path algorithm provides the shortest possible path from one destination to another, instead of provided the closest location that has not been visited. You must know both your current destination and the desired destination

The Floyd Warshall Algorithm provides the closest route from the start destination to all destinations available. If the graph required directed edges, this may have been the algorithm I used to combine the nearest neighbor and Dijkstra’s Shortest Path to handle directed edges better.

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

**1b. Explain how the data structure space usage is affected by changes in the number of packages to be delivered**

The hashmap space-complexity increases linearly O(n) with the number of packages to be delivered.

**1c. 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 space usage of the data structure would increase linearly O(n). The lookup time would increase at a rate of O(logN) due to the key,value pairs being separated into buckets. Look-ups only search a set of keys in the bucket

1. **Identify two other data structures that could meet the same requirements in the scenario.**
   1. **Linked List**
      1. This data structure differs from the Hashmap as a list of connected nodes. The insertion, search, and removal would all be space-time complexity O(n)
   2. **Binary Search Tree**
      1. This data structure differs from the Hashmap as root-leaf relationship. She Insertion and search of information would be O(logN). However, implementing a BST may not be the best option for this type of program.