**Problem Statement:**

               This project was to create the best route and delivery of several packages for Western Governors University Parcel Service (WGUPS) using Python 3.7. It started with 40 packages to be delivered across the Salt Lake City area utilizing two trucks and three drivers throughout a typical workday. The information for the packages contained certain constraints, delays, or errors that needed to be remedied before shipment. There was also to be a user-friendly interface to display this information quickly and correctly.

**Algorithm Identification: Explain and point out the algorithms used**

This program utilized a greedy algorithm to optimize the delivery route of each truck. It takes the shorted path possible for each package from where it is currently located and repeats that process until there are no more packages to get delivered for that truck.

**Logic Comments: Explain logic using pseudocode**

To properly accomplish this greedy algorithm, specific parameters were created

truck\_packages - The list of packages given for a particular truck

truck\_number - The number of the truck that is out for delivery (Frist, Second, Third)

truck\_location – Where the truck is located at any given time

Next, the list needs to be checked if is empty, or there are no more packages to get delivered, which will require a for loop.

               If the length of truck\_packages is equal to 0 then return an empty list

Otherwise, the for loop will continue and the following parameters are set

               next\_closest\_location set equal to 50.0 – The next closest location

next\_location set equal to 0.0 – Where the truck will go next

Current\_location - Where the truck is currently going

If truck\_packages is not equal to 0

               For index in truck\_packages s:

Checks all locations and if a route with a shorter distance is found then it will update next\_location

next\_location is then set equal to Current Path

**Development Environment: Software and hardware explanation used to create Python program**

This program is written in Python 3.7 using the Visual Studio Code IDE. All of the files are located on a local machine therefore, the program is limited to run on a machine that has access to all the appropriate files.

**Space-Time and Big – O: Explain big O for each segment and the program as a whole**

This is the breakdown of the Space-Time Complexity broken down by each file, including the totals.

Distance.py

|  |  |  |
| --- | --- | --- |
| **Function** | **Space Complexity** | **Time Complexity** |
| check\_address | O(1) | O(1) |
| check\_distances | O(1) | O(1) |
| Check\_current\_distances | O(1) | O(1) |
| check\_first\_truck\_time | O(N) | O(N) |
| check\_second\_truck\_time | O(N) | O(N) |
| check\_third\_truck\_time | O(N) | O(N) |
| find\_shortest\_path | O(N^2) | O(N^2) |
| first\_truck\_final\_index | O(1) | O(1) |
| first\_truck\_final\_list | O(1) | O(1) |
| second\_truck\_final\_index | O(1) | O(1) |
| second\_truck\_final\_list | O(1) | O(1) |
| third\_truck\_final\_index | O(1) | O(1) |
| third\_truck\_final\_list | O(1) | O(1) |
| **Total** | O(N^2) | O(N^2) |

HashTables.py

|  |  |  |
| --- | --- | --- |
| **Function** | **Space Complexity** | **Time Complexity** |
| \_init\_ | O(1) | O(1) |
| create\_hash\_Key | O(1) | O(1) |
| Insert | O(N) | O(N) |
| Lookup | O(N) | O(N) |
| Delete | O(N) | O(N) |
| **Total** | O(N) | O(N) |

Main.py

|  |  |  |
| --- | --- | --- |
| **Function** | **Space Complexity** | **Time Complexity** |
| While loop | O(1) | O(N) |
| If -else | O(N) | O(N) |
| For loop | O(N) | O(N^2) |
| **Totals** | O(N^2) | O(N^2) |

Packages.py

|  |  |  |
| --- | --- | --- |
| **Function** | **Space Complexity** | **Time Complexity** |
| Check\_first\_truck *for* | O(N) | O(N) |
| First\_truck\_count *for* | O(N^2) | O(N^2) |
| Distance for distances.py (1) | O(N) | O(N) |
| Update Status *for (1)* | O(N) | O(N) |
| Second\_truck\_count *for* | O(N^2) | O(N^2) |
| Distance for distances.py (2) | O(N) | O(N) |
| Update Status *for (2)* | O(N) | O(N) |
| Third\_Truck\_count *for* | O(N^2) | O(N^2) |
| Distance for distances.py (3) | O(N) | O(N) |
| Total Distance | O(1) | O(1) |
| **Totals** | O(N^2) | O(N^2) |

ReadCSV.py

|  |  |  |
| --- | --- | --- |
| **Function** | **Space Complexity** | **Time Complexity** |
| readCSV *for* | O(N) | O(N) |
| get\_hash\_map | O(1) | O(1) |
| check\_first\_truck | O(1) | O(1) |
| check\_second\_truck | O(1) | O(1) |
| check\_third\_truck | O(1) | O(1) |
| **Totals** | O(N) | O(N) |

**Scalability and Adaptability: Explain how the solution can adapt to different size packages**

The program can adapt to different size packages due to the fact that it is mainly linear in space and time complexity. This means that the package size can size up or down without much of an effect in space–time complexity.

**Software Efficiency and Maintainability: Why the software is efficient and easy to maintain**

This software is efficient and easy to maintain since it is mainly linear in space-time complexity. The only maintenance that would need to be done, is if different packages need to get delivered. This program manually selected the packages for each truck, therefore only those particular packages can go through the program.

**Self-Adjusting Data Structures: Discuss the strengths and weakness of the self-adjusting data structures and hash tables**

Self-Adjusting Data Structures and Hash tables are great in the fact that they will always go with the path of the shortest distance. Also, the time complexity of a hash table is consistent no matter how large the data set is, which works well when there are applications that require different data sets that are both large and small. Collisions, however, can be a disadvantage to using hash tables as they can decrease performance.

**Explanation of Data Structure: How the DS accounts for the relationship between the data points to be stored**

This program implements a list of the list. This type of data structure accounts for the relationship between data points as it allows for chaining in hash tables which makes lookup, updating, and deleting much quicker.

**Strengths of the Chosen Algorithm: At least 2 strengths**

This program was written with a greedy algorithm that meets all the given requirements of the project. One of the great advantages of using Greedy Algorithms is that it always finds the optimal path for each truck. Also, with using a Greedy Algorithm, the time complexities are typically less than if a different type of algorithm was implemented.

**Other Possible Algorithms: ID 2 other possible algorithms that could have been used**

**Algorithm Difference What would have made those two algorithms different?**

There are a couple of different ways that this program could have been approached in terms of which algorithm to choose. There is a self-adjusting heuristic algorithm, which would have not been within the constraints of the project as according to Zybooks “ A heuristic may willingly sacrifice accuracy to improve execution speed” (zybooks, 3.1) It is similar to the algorithm chosen in the fact that the shortest path is always chosen, however, a heuristic algorithm would have determined which packages need to be loaded in each truck, from that list of packages it would have chosen the closest location from the hub. After those packages are delivered, the next closest location would be chosen until there are no more packages in the truck to deliver.

Another type of Algorithm that could have been approached is that of dynamic programming, which splits a problem into smaller pieces. If this algorithm was implemented, the space-time complexity would have been greater as the algorithm would have

**Different Approach: ID 1 aspect that would be done differently if the project attempted again**

If this program were to be attempted another time, there would be a couple of things to do differently, the main of which is a non-manual entry of packages into trucks, this would be beneficially in the program’s ability to be more adaptable if there were different or additional packages that need to be delivered.

**Verification of Data Structure: includes the total miles added to all trucks, the total combined delivery distance is less than 140 total miles, all packages are delivered on time, the hash table with look-up function is present, and the reporting needed is accurate and efficient.**

Below are screenshots verifying this information

**Efficiency: Explain how if the # of packages changed, how changes in the number of packages directly affect the time needed to complete the lookup function.**

If the number of packages was to change, the time it would take to complete the lookup function would change, the time complexity for the lookup function of the hash table is O(N), which is means that the number of packages would affect the time by a factor of N.

**Overhead: how changes in the number of packages directly affect the data structure space usage.**

Unlike, how the time it would take to use the lookup function would increase with the changing number of packages, the data structure space usage would not increase if the number of packages was to change. The space complexity for this is O(1) meaning that for N number of packages, the space needed to sort those packages would change by a factor of. In other words, it would remain consistent no matter how the number of packages fluctuates.

**Implications: how changes to the number of trucks or the number of cities would affect look-up time and space usage.**

Similarly, to how the time it would take to use the lookup function if the number of packages increases, the space-time would need to increase as well if there were an additional number of trucks or cities. The space-time complexity for this is O(N) meaning that for N number of packages, trucks, or cities, space and time needed to sort those packages, into the different trucks or cities would change by a factor of N.

**Other Data Structures: Two data structures other than the one used in part D that could meet the requirements in the scenario.**

**Data Structure Differences: describes attributes of each data structure identified in part K2, and it accurately compares these attributes to the attributes of the data structure used in the solution.**

There are a couple of different data structures that could have been used with this program that would have worked within the constraints provided. One of these is a graph, this would have been a good data structure to implement if there was a need for larger scalability as with graphs, multiple data points could be grouped in adjacent vertices.

Another data structure that could have been used is a Binary Search tree. This would have made it easier for getting a specific data point as it would have been pre-sorted.

**Sources**

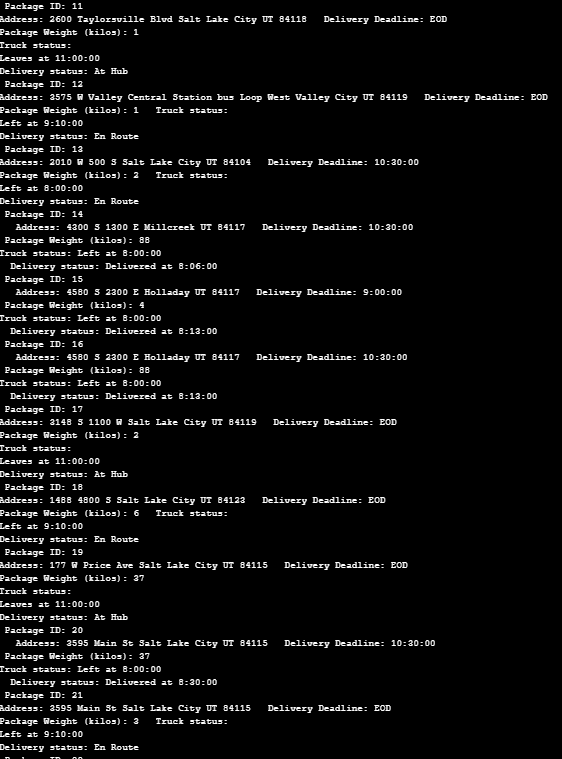
Learn.zybook.com. (n.d) *zybooks.* [online] Available at:

<https://learn.zybooks.com/zybook/WGUC950AY20182019/chapter/3/section/1> [Accessed 17 March 2021]

**Screenshots**

First Status Check at time between 8:35 a.m. and 9:25 a.m (09:15:00)

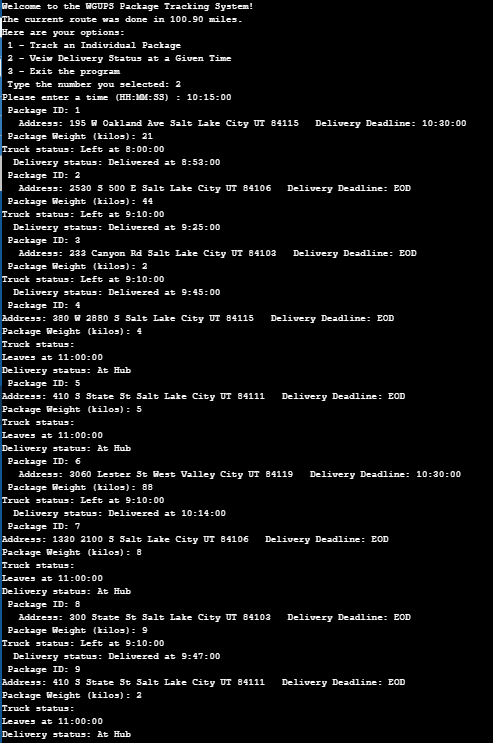


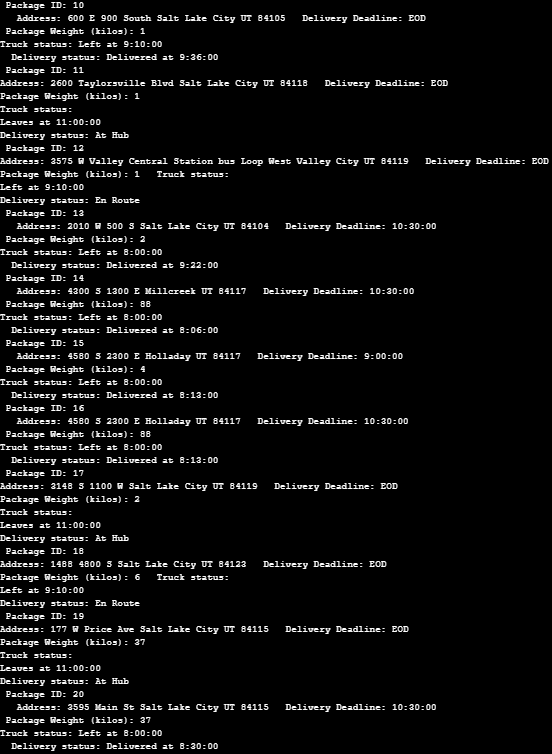


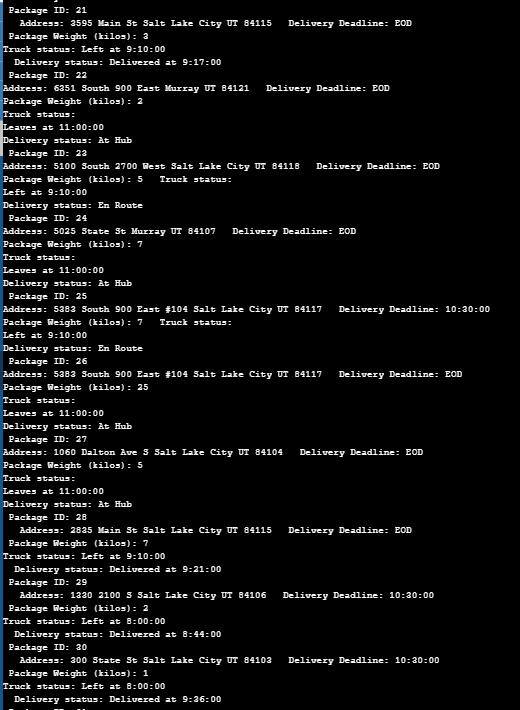




Second Status Check at a time between 9:35 a.m. and 10:25 a.m. (10:15:00)



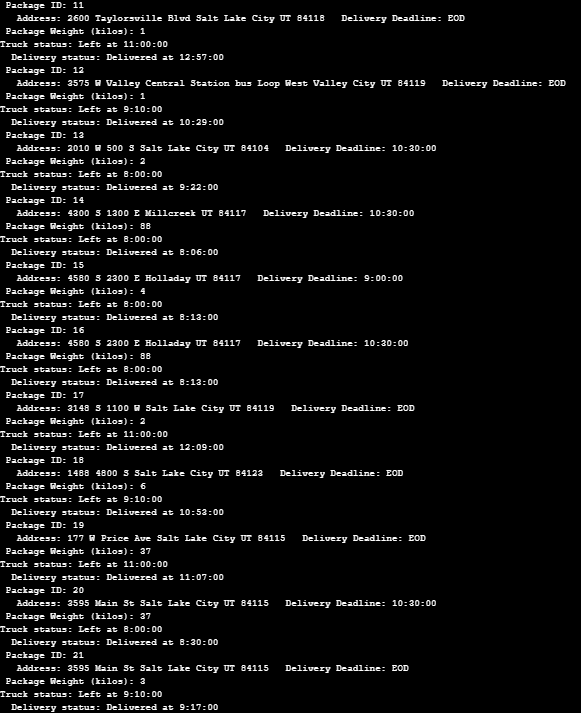


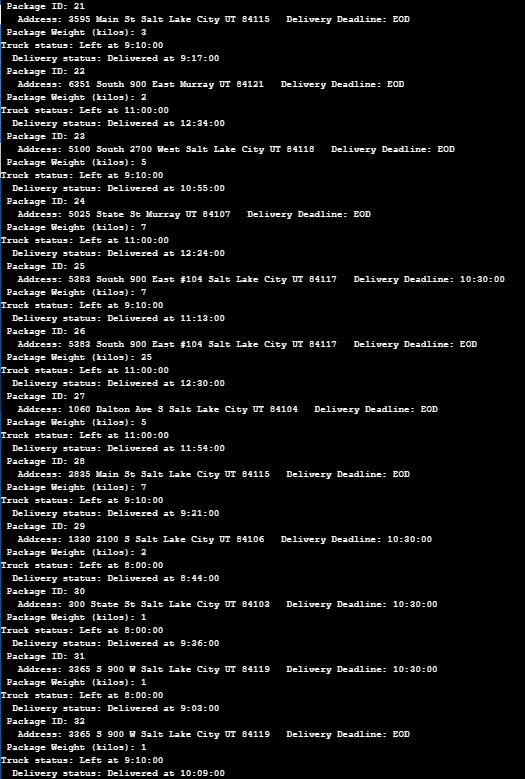


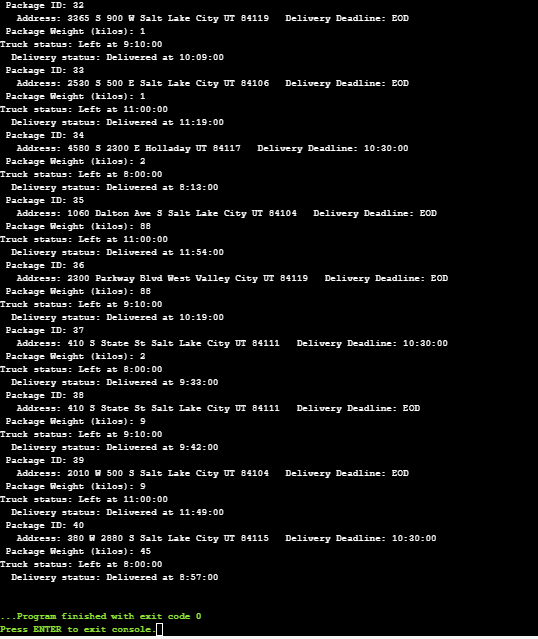


Third Status Check  at a time between 12:03 p.m. and 1:12 p.m. (13:00:00)



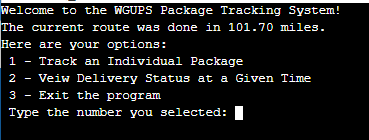




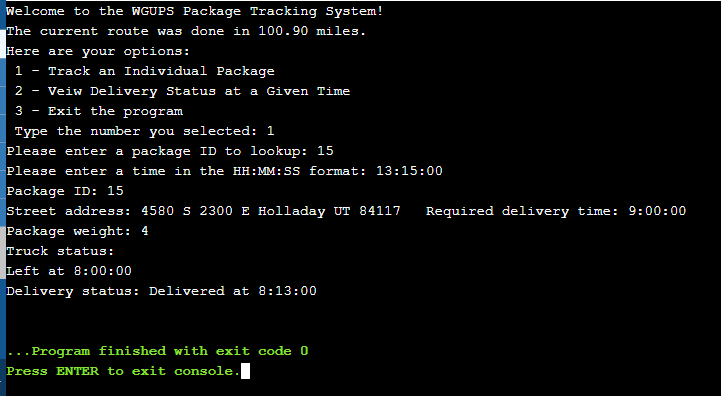


**Screenshot of Code Execution**

User Interface



Options 1



Option 3

