WGU C950 Data Structures and Algorithms 2- Task 2

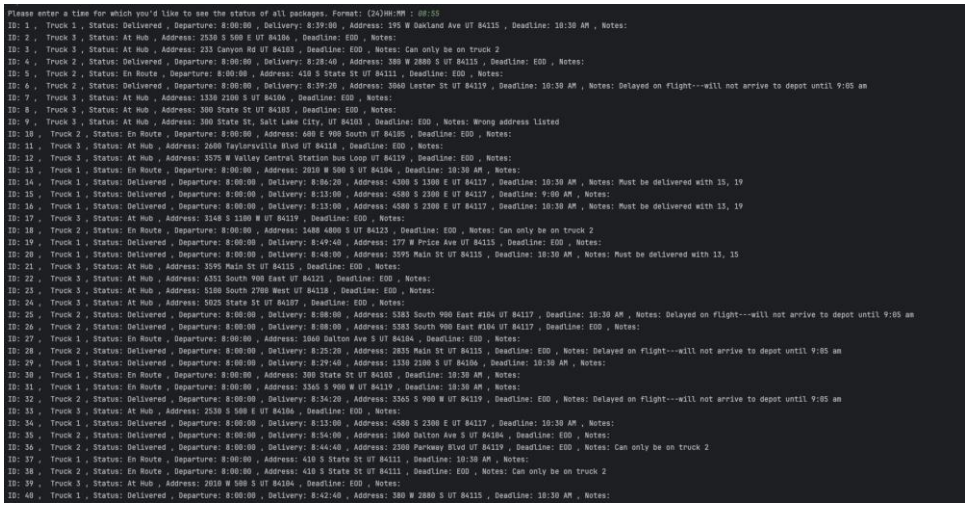
By: Andrew Vigil

Date: 3/10/24

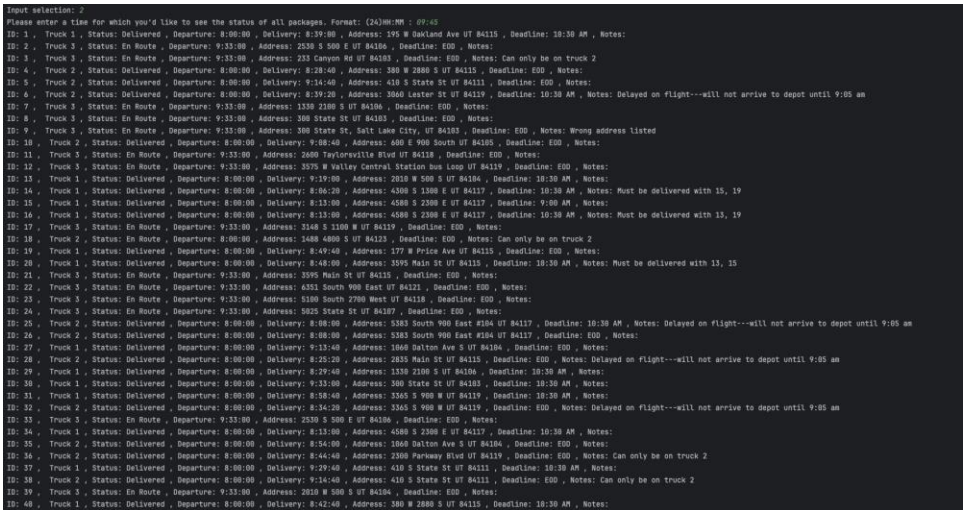
D. Provide an intuitive interface for the user to view the delivery status (including delivery time) of any package at any time and the total miles traveled by all trucks.

```
/Users/andrewstephens/PycharmProjects/c950/.venv/bin/python /Users/andrewstephens/PycharmProjects/c950/main.py
Western Governors University Parcel Service (WGUPS)
Select from the 4 options below:
1. Package Statuses and Total Miles for All 3 Trucks
2. Package Statuses at a specific time
3. Get single package status for a specific time
4. Exit the program
Input selection:
```

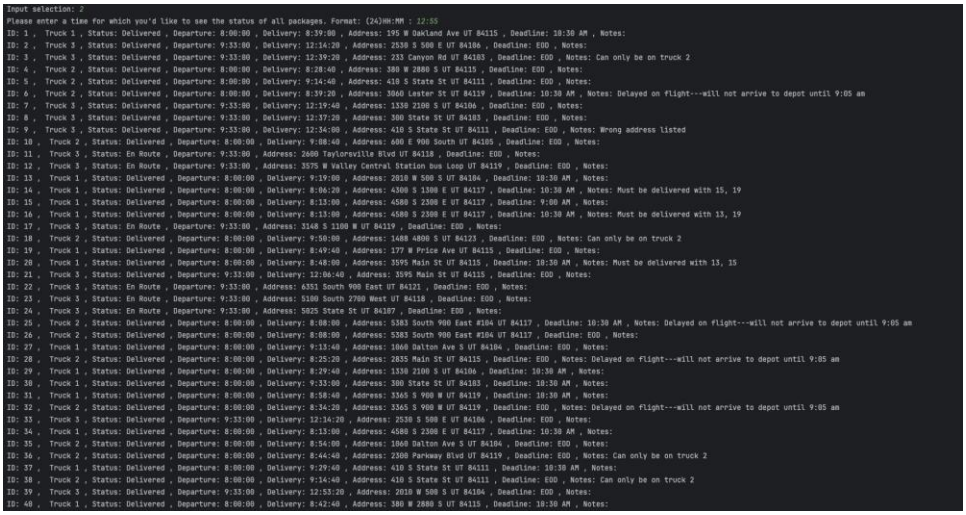
1. Provide screenshots to show the status of all packages loaded onto each truck at a time between 8 :35 am and 9:25 am.



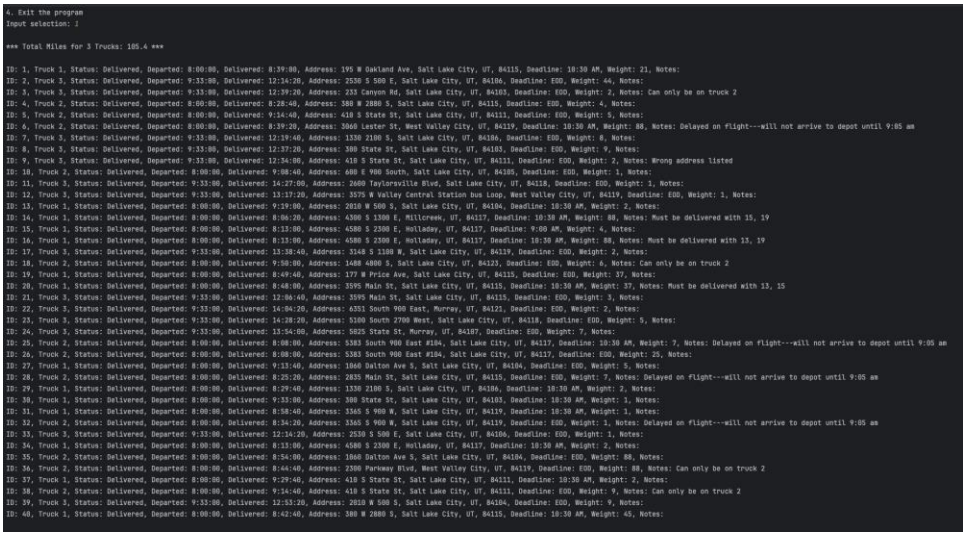
2. Provide screenshots to show the status of all packages loaded onto each truck at a time between 9:35 am and 10:25 am.



3. Provide screenshots to show the status of all packages loaded on each truck at a time between 12:03 am and 1:12 pm.



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



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

1. Describe 2 or more strengths of the algorithm used in the solution.

The first strength of the Nearest Neighbor algorithm would be its simplicity, its digestible to understand and works simply fine for a small sample size like this project or a bit bigger where you can still manually load vehicles. The second, and playing-off of the first, would be ease of implementation. I was able to create the bulk of the algorithm in 25 or so lines, which I was surprised by.

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

The algorithm I selected meets all the requirements of the given scenario including 1) keeping the mileage under 140 total miles, 2) delivering packages to their address by/after a specific time, and 3) accommodating to packages with special instructions.

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

The two algorithms that I was contemplating using were Prim's algorithm and Kruskal's algorithm

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

From what I understand, Kruskal's algorithm works by sorting the graph's edges by weight to find the minimum spanning tree. It seems simple and should produce a good result, but I am just less familiar with the heap data structure that is recommended for it.

Prim's algorithm is very similar to Kruskal's except it prefers a list data structure and it starts the minimum spanning tree from any vertex in the graph, as opposed to the vertex carrying the minimum weight like in Kruskal's.

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 did this project again, I would spend more time optimizing for scalability. For example, for this project I loaded the trucks manually, but something I would do different in the future is make it so it can programmatically load the trucks.

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

The hash data structure meets all the requirements for the scenario provided. It accurately stores all package info, with the ID acting as the unique key. The values it stores are the address, street, city, zip, weight, status, deadline, and special notes. The hash allows for packages to be inserted, searched for, and deleted.

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

I believe that a queue or a doubly linked list both could have worked as an alternative data structure option.

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

A FIFO queue data structure takes in data and puts it at the end and takes from the front of the queue. It is less efficient because you must go through the queue to retrieve what you are looking for versus a hash where you can go directly to what you are lookingf or with a key value pair. The same is the case for the doubly linked list, ,which is an often useful structure and would work in this situation as well, but would also warrant the traversal of an entire data set to find the desired value.