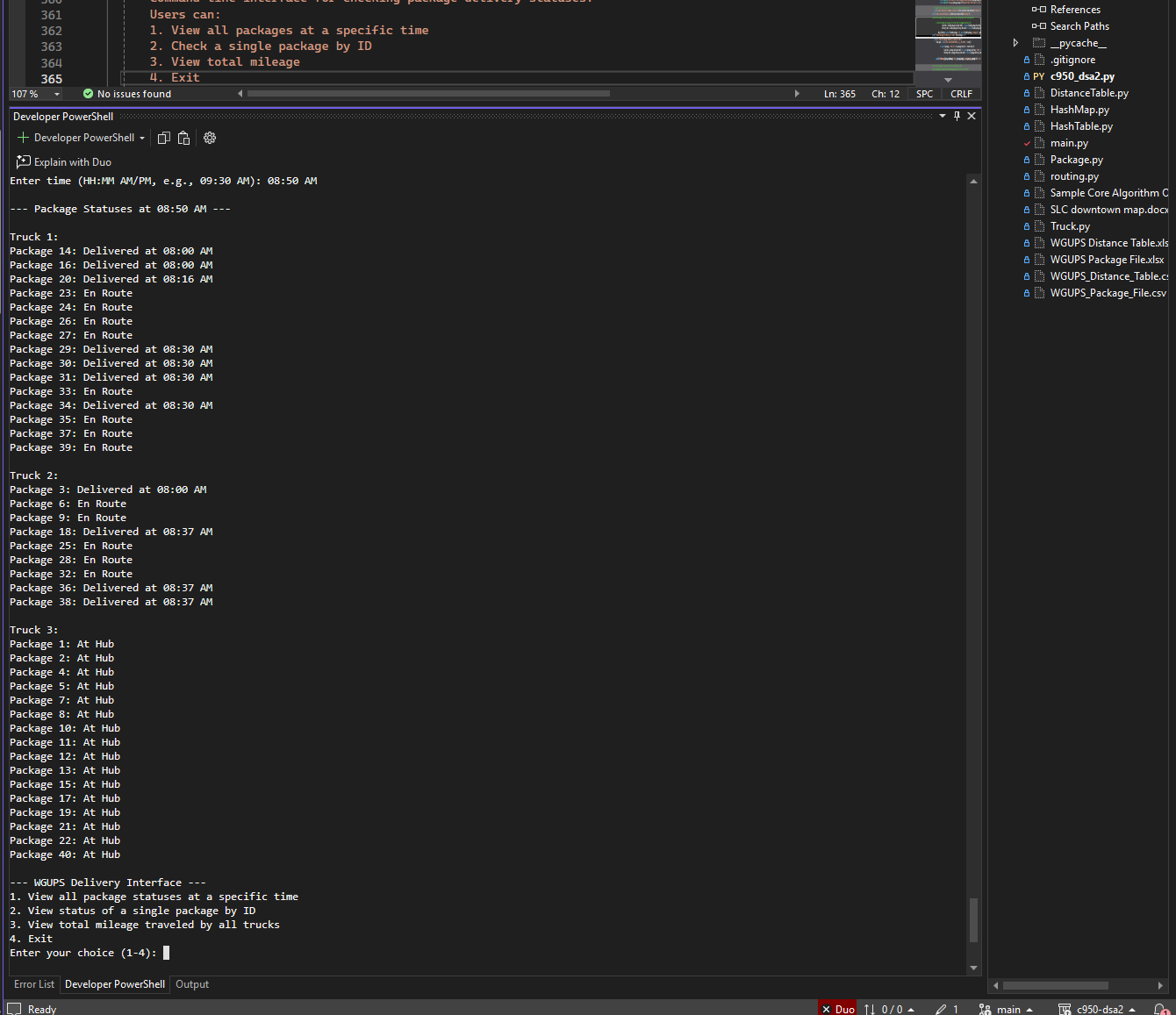
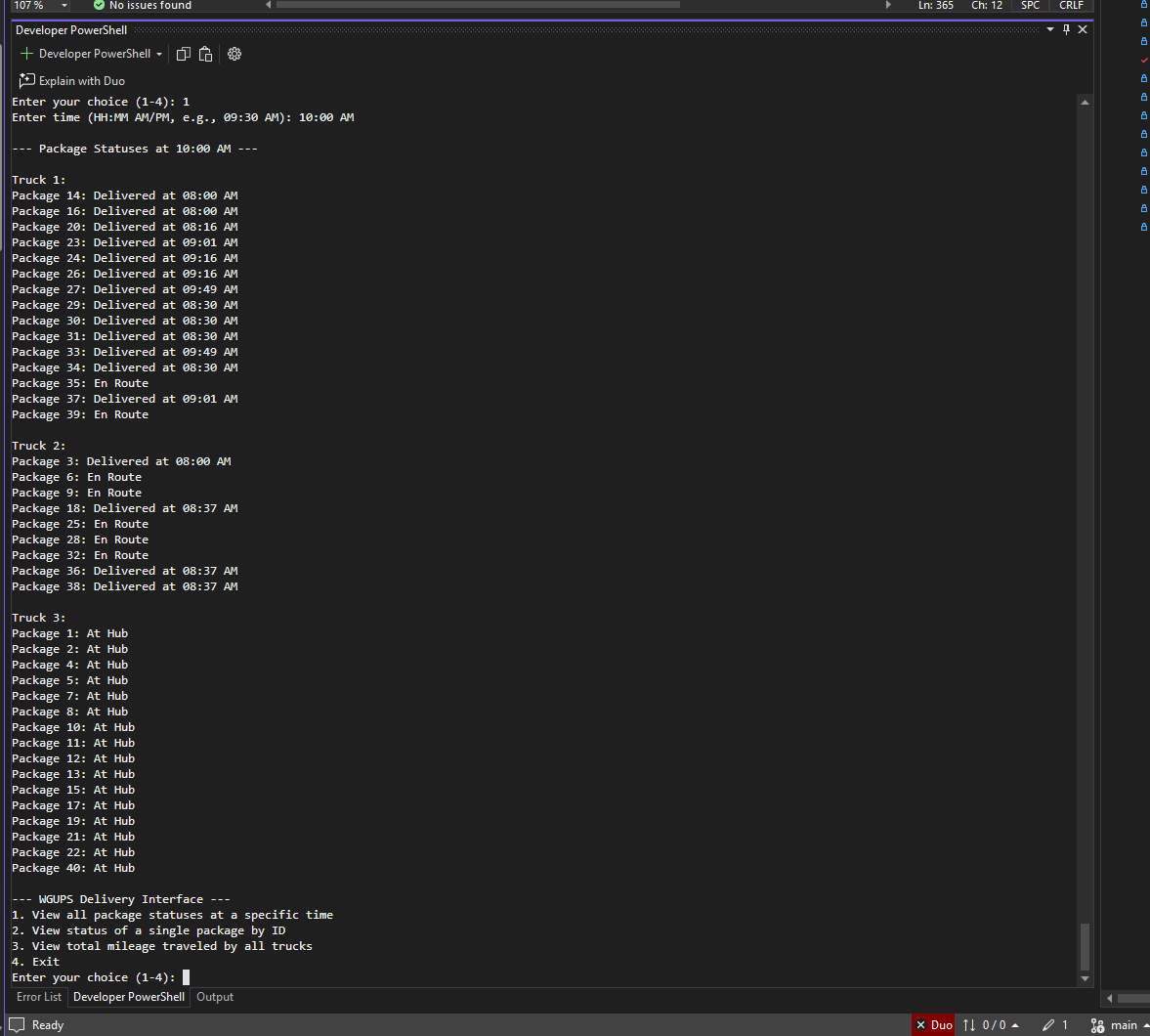
Jake Vichnis Data Structures & Algorithms:

Task 2

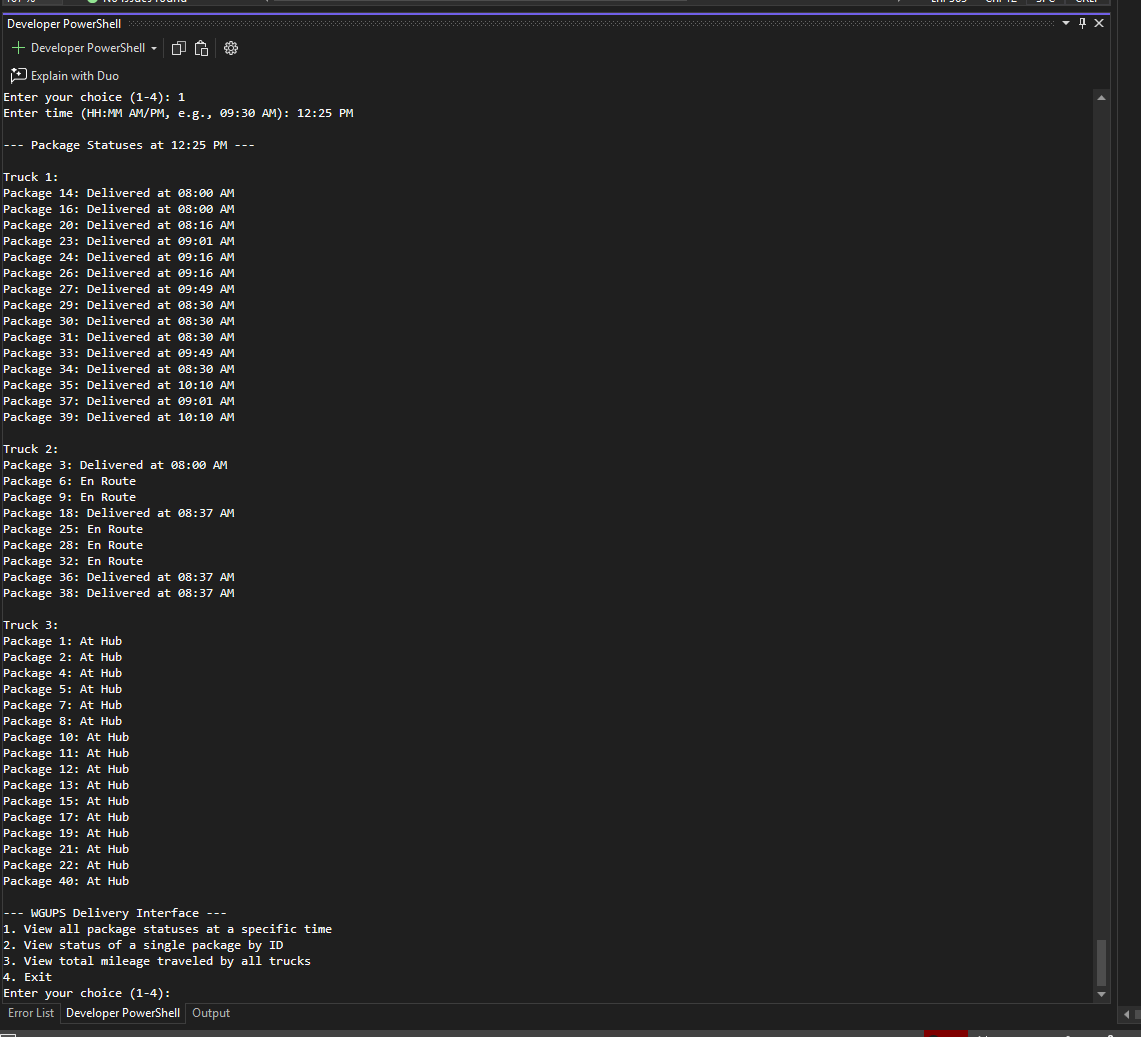
Step D:



Between 8:35-9:25

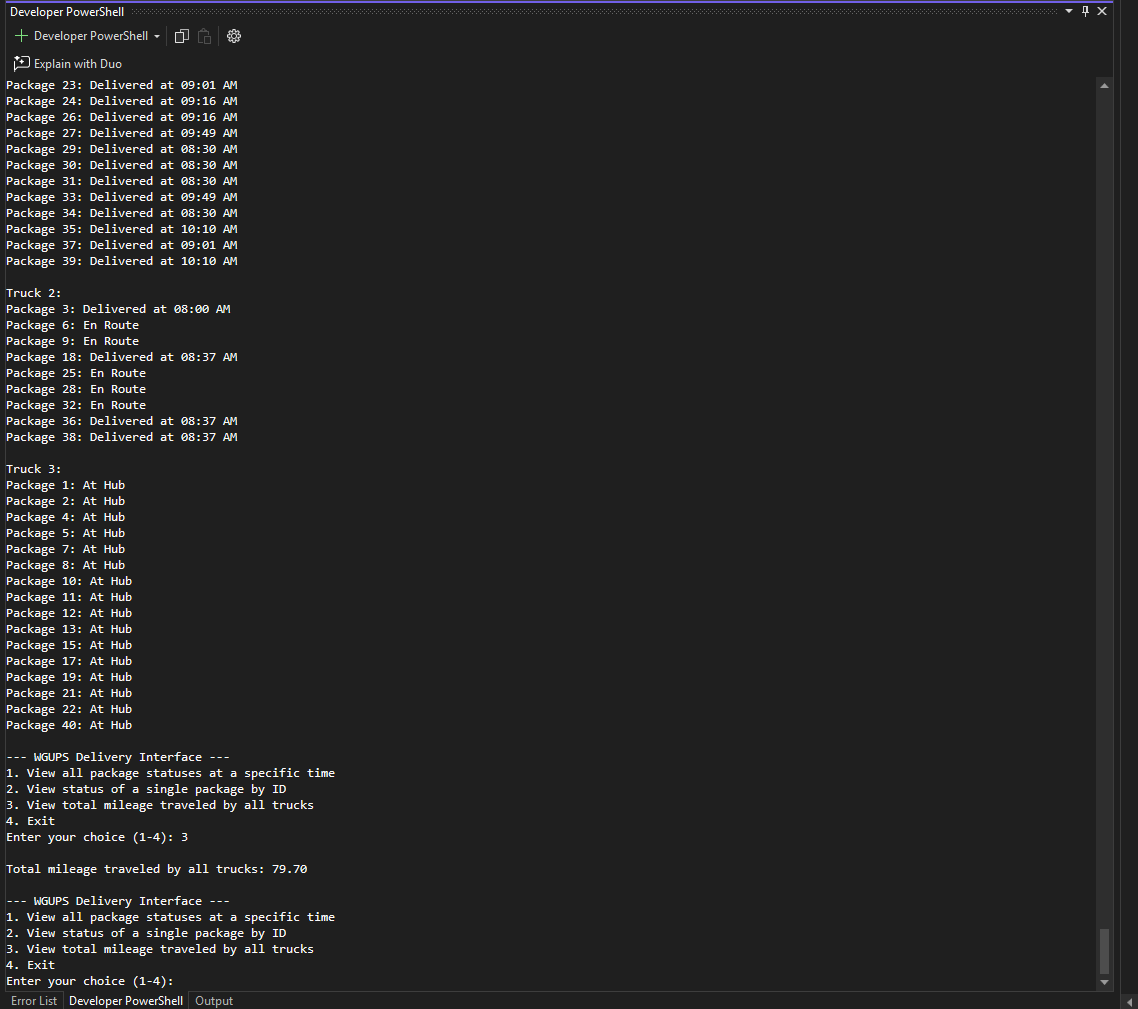


Between 9:35-10:25am



Between 12:03-1:12pm

Step E:



Successful Completion with Total Miles from all trucks

Step F:

1. Two major strengths of this algorithm are 1- it's ability to truly scale based on a wider location an additional third truck driver, provided the storing mechanism for the packages is fed into and through a database. And 2- due to the amount of possible checking, with also the inherent nature of its immediate greedy-overall-nearest neighbor, it handles odd clusters of packages outside the greedy range. This algorithm also filters for the constraints without sacrificing time or ability to further handle other packages in combination.
2. This algorithm meets all requirements of this assignment, juggling two drivers between three cars, delivering all 40 packages and keeping everything under the 140 mile limit. Biggest hurdle of writing this was filtering for the constraints of the wrong address, and grouped packages.
3. Two other named algorithms that could be used in place would be (had to research if some of these would \*actually\* be able to solve it in the same way):

Christofides- an approximation algorithm to find the minimum-weight matching for odd-degree vertices, creating a new circuit and then shortcutting it. Instead of making greedy local choices, looks to optimize for the entire route regardless of starting distance to the hub.

Genetic algorithms: using a package-based heuristic that evolves a full entire route solution using the multi trucking schedule and more complex constraints. Would need more parameter tuning and slower to implement possibly but the route has a better overall mileage than the greedy algorithm for difficult package address clusters.

Step G:

G. Other than the two algorithms, I mostly likely choose the Christofides. It’s a new algorithm for me and would be fun to implement, and looks to be related in its approximation nature to some projects I have done in previous classes. I would definitely also use my scripting & applications project parser to integrate better the packages with the csv, or even better, just reshape the entire packaging data and throw the whole thing into a database. The ultimate goal of having a seamless feed for packages in a database to be pulled directly onto a cart and shipped off on a truck handling effectively a neighborhood cluster of packages could really make this project even more seamless. Now I understand better why the post office dissects delivery routes the way they do!

Step H:

H. My data structure (As I probably have iterated in other ways in other steps) fulfills all the requirements of assignment. My hashtable provides a fast insert and lookup by package\_id which is what is needed. Frequently looking up during routing, low overhead, and easy insertion during CSV load. It gives O(1) average access which is incredibly fast and keeps the routing loop fast and simple. It stores packages by ID (as directed in the requirements) so the assignment step can keep only IDs (not package objects) on trucks.

1. The two other obvious other ways we could have structured this is definitely with a python Dictionary (dict)
2. And a balanced BST Binary Search tree split across the trucks.

a. The differences between a dict and our hashtable are largely the same but the code would be different and more maintainable on the python dict. More simplicity. The balanced BST would give a O(log n) guaranteed worse-case lookup/insert and supporting the ordered traversals through the trucks could be beneficial. However, I would imagine if the packages become large enough a trie could be more efficient than a BST at that point (just occurred to me as I was writing this).

Sources: Many, many, many google searches. Youtubes on hash tables, and Professor Bob Branch for the guidance along with the zybooks.