Task D:

Snapshot at 8:50AM:

A screen shot of a computer screen

Description automatically generated

Snapshot at 10:00AM:

A screen shot of a computer screen

Description automatically generated

Snapshot at 12:30PM:

A screen shot of a computer screen

Description automatically generated

Task E:

A screen shot of a computer screen

Description automatically generated

Total mileage for all trucks was around 111.2.

Task F:

One of the strengths for choosing the nearest neighbor algorithm for my project was because it is an easy algorithm to implement and understand. It was also the first algorithm that came to mind when I was planning the project. The drawback to the nearest neighbor algorithm is that it is a greedy algorithm and might not result in the most optimal solution. However, per the webinars, it is better to keep the project simple and not to over complicate things, so I decided to stick with the nearest neighbor algorithm. Another strength of the nearest neighbor algorithm is that it is efficient. Even though it won’t always provide the most optimal solution, an efficient solution is good enough for the scope of this project.

Task F2:

The algorithm used in the solution met all the requirements in the scenario. The program utilizes three trucks with two drivers. No more than two trucks are active on their respective routes due to limitation of the number of drivers. All 40 packages are delivered on the three trucks by their deadlines and have their specific requirements listed on the notes section fulfilled. Each truck deliveries no more than 16 packages as once. Their average speed is set to 18 mph and no trucks leave the hub earlier than 8AM. Package 9 address was updated and the program was able to deliver package 9 to the correct address based on the result screen. The algorithm was able to keep the mileage for not just two but all three trucks to under 140 mph as well.

Two other named algorithms that would also meet the requirements in the scenario would be the Djikstra’s shortest path algorithm and Bellman-Ford’s shortest path algorithm (*ZyBooks*, n.d.). Both Djikstra and Bellman-Ford algorithms use graph traversal techniques from the starting source node. Every node has a field for storing the smallest distance to said node observed thus far and a field designating the predecessor node. For both algorithms, every node is first initialized with the fields set to positive infinity and null. The starting node is set to 0 and values for adjacent nodes are updated if the sum of the edge weight and predecessor node value is less than the adjacent node current value. This is repeated for all unvisited nodes in a queue. Unlike the Djikstra algorithm, the Bellman-Ford algorithm however only propagate to only one node each iteration and can handle negative edge weights.

Task G:

If I had to do this project again, I probably will stick to the nearest neighbor algorithm again, but this time I will try to find the most optimal routes. In my project so far, I only ran one set of data. Truck 1 had 13 packages, truck 2 had 16 packages and truck 3 had 11 packages. I can use different package size for each truck or use a different starting point instead of the first package of each package set when filling out the package set, then run the program multiple times to find the most optimal truck/package combinations.

Task H:

In my opinion, the best data structure for this problem is one that has a fast insert, search & delete time complexity and expandability. A hash table has an average constant time complexity for insert, search & delete functions with a worse case of O(n), which is fast compared to other data structures. For expandability, a hash table can be dynamically resized if needed (*ZyBooks*, n.d.).

The modified chaining hash table based on key value pairings enforces that keys are unique based on package IDs. The user can search and lookup the information of any packages for efficient access. Hash tables allows for fast look ups at constant time. Thus, in this case, package 9 that had an incorrect address was easily first searched in the hash table by its unique key identifier and then updated with the correct address. The hash table is big enough to store all 40 packages and since a hash table can by dynamically resized, this allows for future scalability as well. The data structure used in the solution therefore meets all requirements in the scenario.

Two other data structures that can meet the requirements in the scenario would be a binary search tree or a linked list. A binary search tree runs in log of n time complexity in a best case scenario, O(n) in the worst case scenario and does not have to be resized (*Binary Search Trees: BST Explained with Examples*, 2019). A linked list offers similar time complexity O(n) for insert, search and delete functions, with potentially a best-case scenario of constant time if the item to be inserted, searched or deleted is at the head of the linked list (*Coding Ninjas Studio*, n.d.). Linked lists have O(n) space complexity with additional space required/wasted for pointers.

Task I:

*zyBooks*. (n.d.). Learn.zybooks.com. Retrieved October 28, 2023, from <https://learn.zybooks.com/zybook/WGUC950Template2023/chapter/5/section/10>

*zyBooks*. (n.d.). Learn.zybooks.com. Retrieved October 28, 2023, from <https://learn.zybooks.com/zybook/WGUC950Template2023/chapter/5/section/9>

*zyBooks*. (n.d.). Learn.zybooks.com. Retrieved October 28, 2023, from https://learn.zybooks.com/zybook/WGUC950Template2023/chapter/6/section/6

*Binary Search Trees: BST Explained with Examples*. (2019, November 16). FreeCodeCamp.org. <https://www.freecodecamp.org/news/binary-search-trees-bst-explained-with-examples/>

*Coding Ninjas Studio*. (n.d.). Www.codingninjas.com. https://www.codingninjas.com/studio/library/time-and-space-complexity-of-linear-data-structures