C950 Task-2 WGUPS Write-Up

(Task-2: The implementation phase of the WGUPS Routing Program).

(Zip your source code and upload it with this file)

Logan Tackett

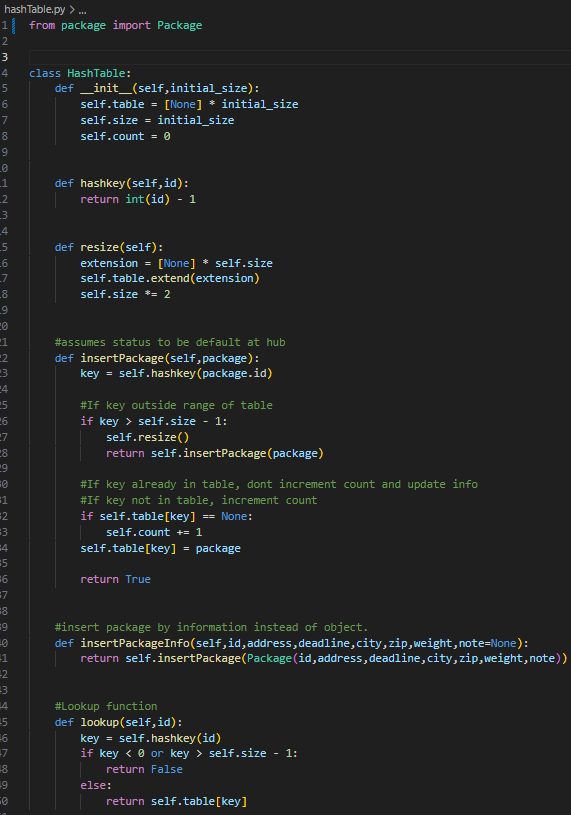
ID #012308366

WGU Email: ltacke12@wgu.edu

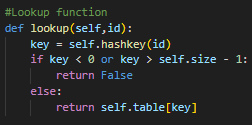
11/12/2024

C950 Data Structures and Algorithms II

# A. Hash Table



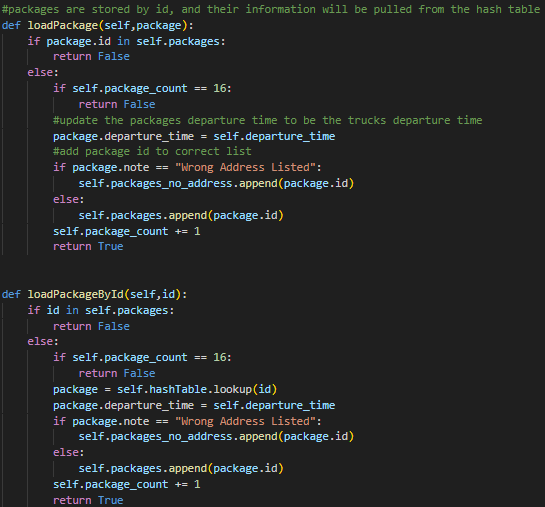
# B. Look-Up Functions



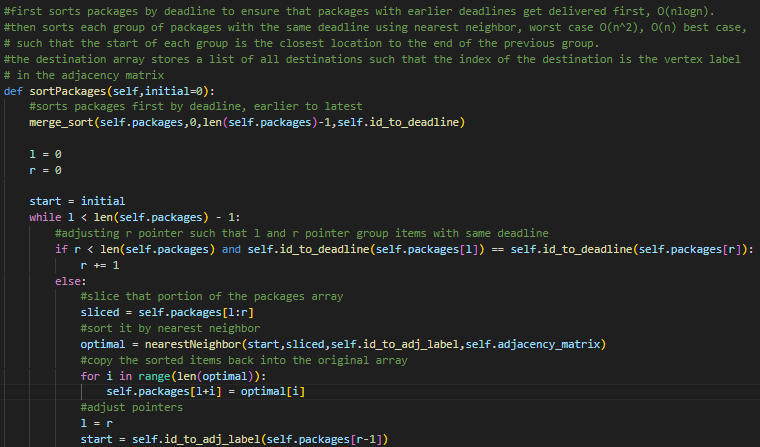
# C. Original Code

Truck Class:

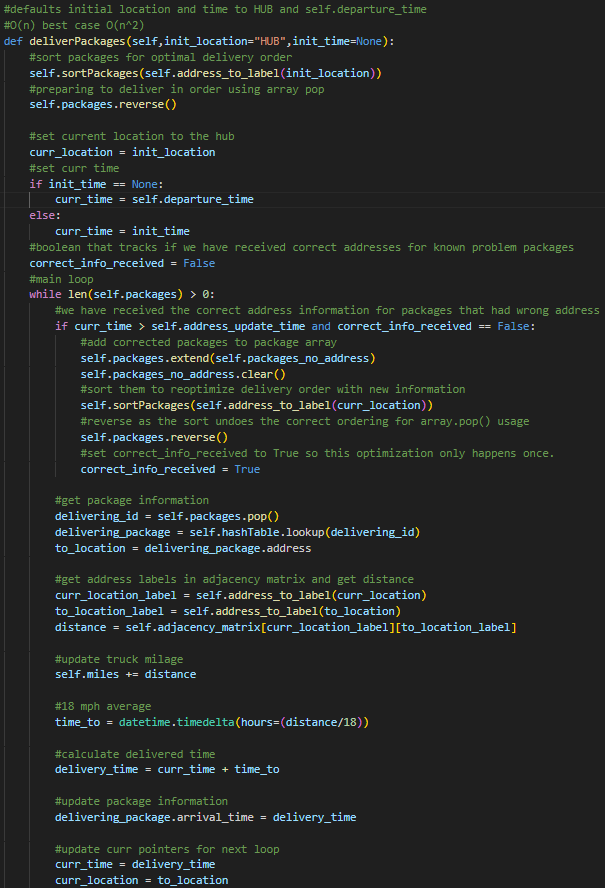
Package loading functions



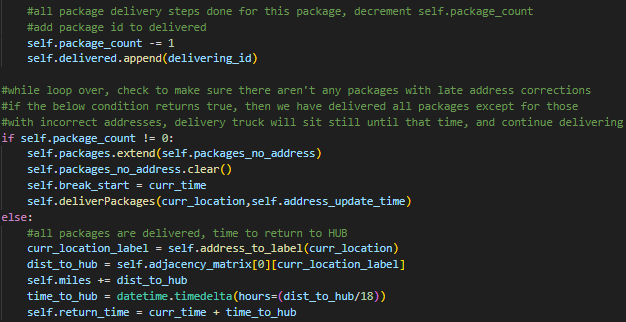
Sort packages into optimal delivery order:



Deliver Packages:



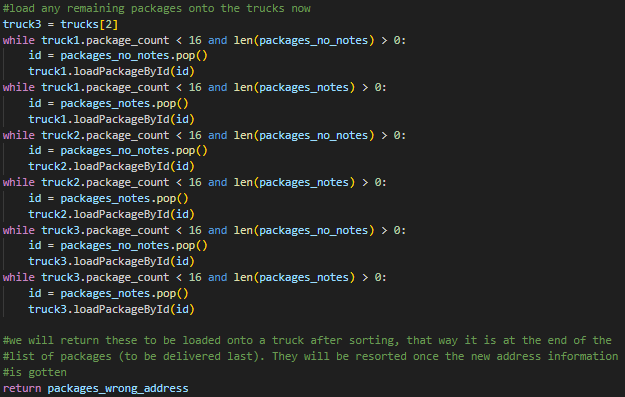
Deliver packages continued:



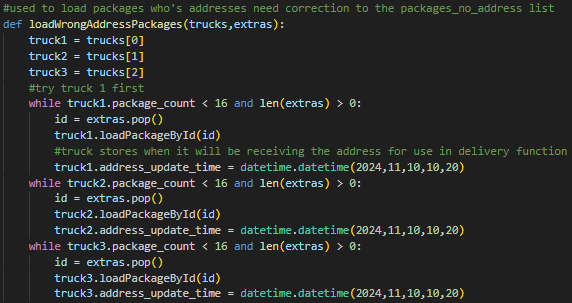
Function that loads trucks both manually and heuristically (by deadline):



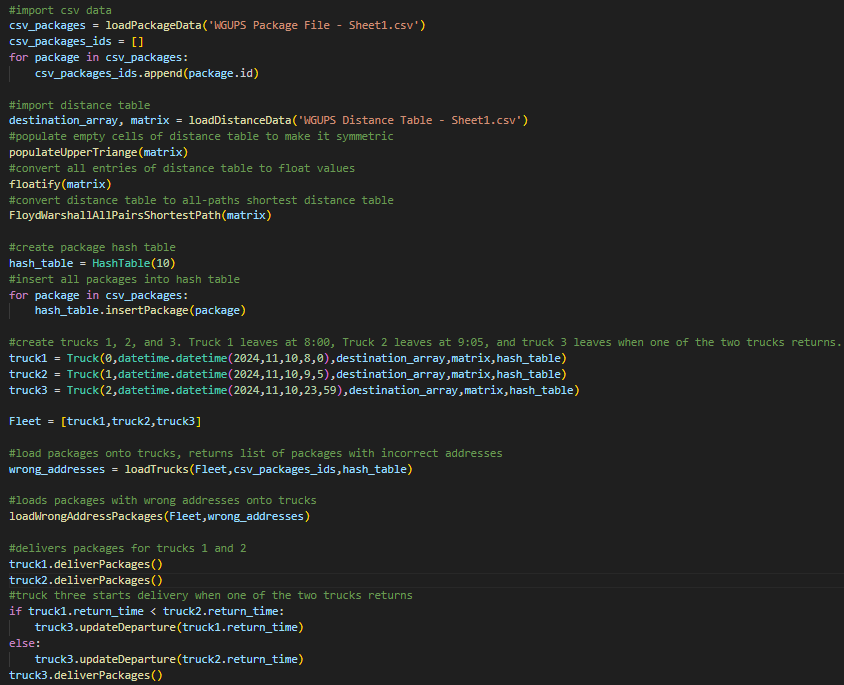
Load trucks continued:



Load incorrect address packages:

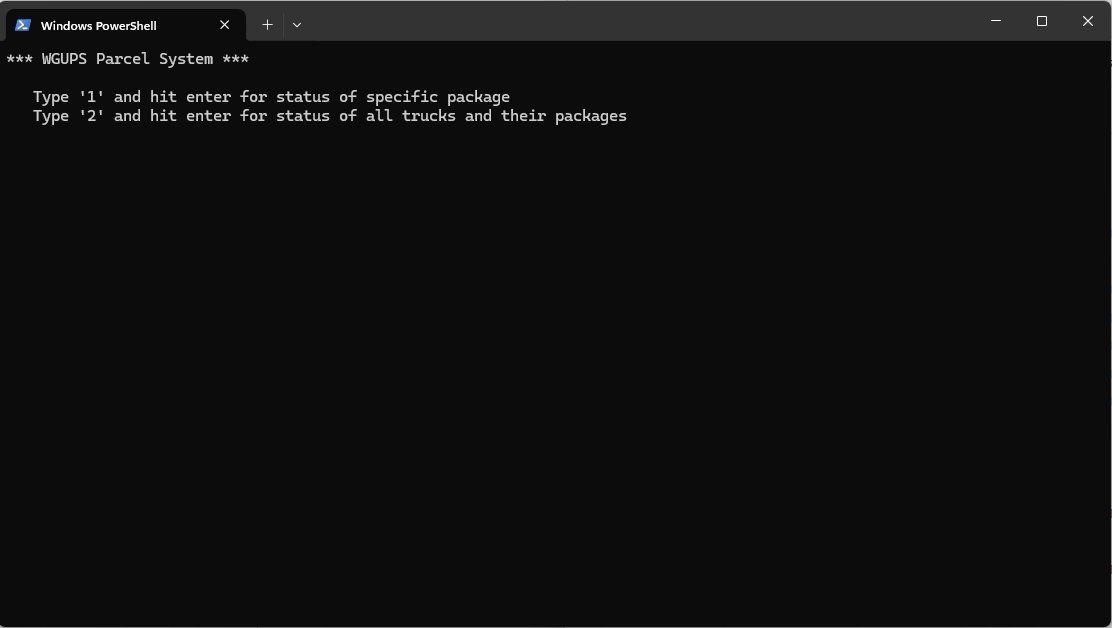


# C1. Identification Information

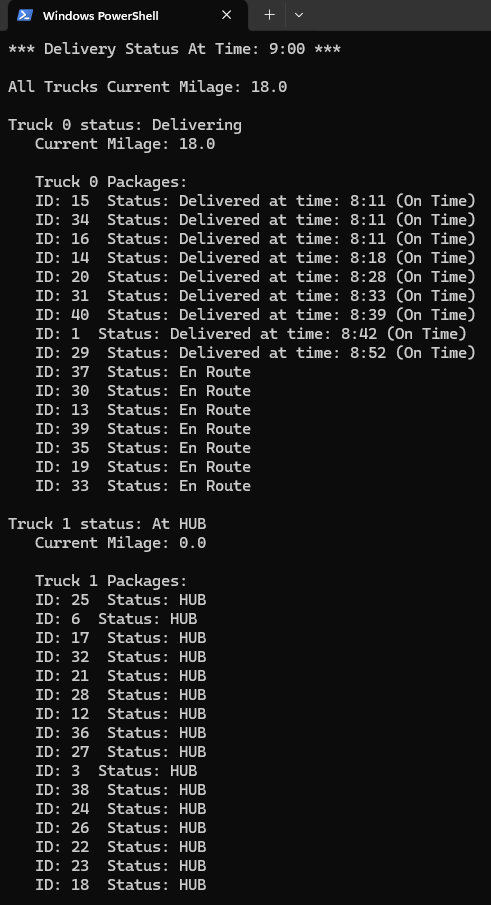


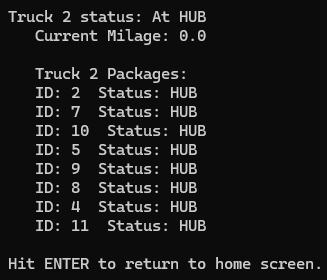
# C2. Process and Flow Comments

# D. Interface

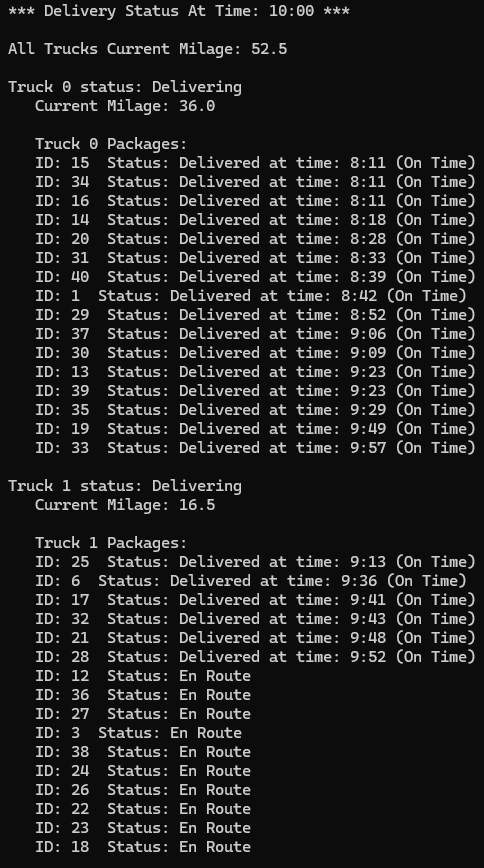


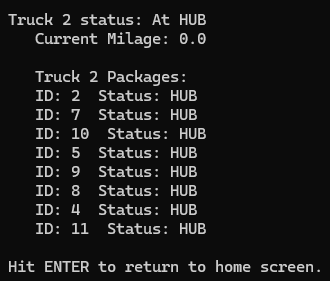
# D1. First Status Check





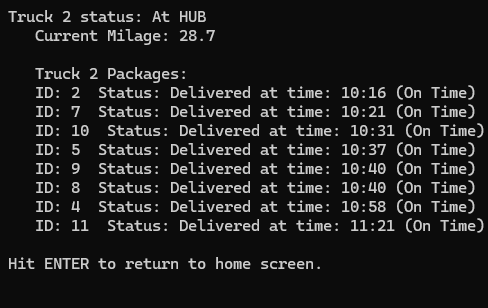
# D2. Second Status Check





# D3. Third Status Check





# E. Screenshot of Code Execution

# F1. Strengths of the Chosen Algorithm

I used a greedy nearest neighbor algorithm applied to an optimized distance table using the Floyd Warshall algorithm. The main strength of the use of nearest neighbor is the algorithm's simplicity. It runs in O(n^2) time and does a close to optimal job of delivering the packages on the truck. Compare this to a DFS algorithm, which could give a more optimal or even perfect solution at the cost of runtime. The main strength of using Floyd Warshall is the easy optimization it provides. It runs in O(n^3) time, which for this problem is efficient as it only runs once, and the outcome of running it is a more optimal distance table that can be used in all future processes of the program. I also used merge sort to sort packages by deadline. This was simply to make sure that packages got to their destination on time. The strength of sorting the packages this way is, again, simplicity and an efficient runtime of O(n log n). Using heuristics like sorting by destination time and then by nearest neighbor greatly simplifies the problem and keeps time and space complexities low when compared to more optimal methods like DFS.

# F2. Verification of Algorithm

The algorithm provided above, plus some edge case handlers, satisfies all requirements in the scenario. All packages are delivered by their deadline, and delayed packages don't leave until 9:05; the package with the missing address is loaded onto a truck as early as possible but isn't delivered until the new address is received, to which the delivery order is resorted such that the route stays optimal even with the new address; packages that must be delivered on the same truck are delivered on the same truck, and packages that must be on truck 2 are all on truck 2. Total truck mileage after all deliveries, including the drive back to the HUB, is 119.6 miles, which is less than 140 miles. Trucks leave no earlier than 8:00 AM and only two trucks are delivering at a time.

# F3. Other Possible Algorithms

Two other algorithms that could have been used to solve this problem while meeting the requirements are DFS (Depth-First-Search) and 2-Opt.

# F3a. Algorithm Differences

A DFS approach would check all possible ways to deliver the packages, excluding pruned orderings, to see which one is most optimal while satisfying all the conditions. This is very different from nearest neighbor, which is a greedy algorithm that produces a near-optimal result without checking any particular package permutation. The scale of this particular problem is probably small enough that DFS would give a more optimal answer in a tolerable amount of time with the correct optimizations, but if there were to be more locations and packages, DFS could take a very long time. With the optimizations available to us for this problem, the runtime of a DFS approach would still be O(n!). One could use a depth-limited search to calculate a close-to-optimal next package to deliver without going through all permutations of packages, but this is essentially what nearest neighbor is doing with a depth of 1. While this method could yield a more optimal result whilst having a polynomial runtime, it would still be much slower than my algorithm of choice. Specifically, the runtime would be O(n choose l), where l is the fixed depth.

2-Opt is a heuristic-based algorithm that finds paths that are close to optimal ones by making sure the path doesn't intersect itself. This could very easily replace the role of nearest neighbor with little refactoring and would have a similar runtime complexity. The question of which one will provide a more optimal result is hard to answer. The path generated by nearest neighbor may intersect itself, which is nonoptimal, but each movement in the path from node A to node B is done such that node B is the closest unvisited node to node A, which is optimal. The path generated by 2-opt would result in a similarly optimal path as the path would never intersect itself, but there is no guarantee that a connection is the shortest one that could have been made.

# G. Different Approach

If I had to redo this project, I would load packages using a more sophisticated approach. While it would be more complicated to do so, the most effective way to create a route that is close to the optimal one is to load the trucks by package proximity. Yes, greedy algorithms applied to a set of packages will produce a more optimal result, but this will never come close to loading the trucks such that packages are as close as possible. Packages with early deadlines that are ready to go on time should simply go on the first truck to leave; this doesn't require a sophisticated algorithm. The same applies to packages with an early deadline that arrive late. For packages that have no requirements, one could first find the two packages that are farthest from each other and load them onto two different trucks. Then, the trucks would take turns loading the next closest available package for each truck. If a majority of the packages had an EOD deadline, this optimization would make a large difference in mileage and delivery time at the same cost as the current implementation (O(n^2)). I wouldn't change anything about the way I deliver packages, as packages are simply delivered in the order that they are listed. This ordering in the list is optimized both before and during delivery as new information comes in.

# H. Verification of Data Structure

The data structure used to store package data is a hash table. The hash table uses no additional libraries or classes to store packages and does so using the package ID and a package object. The hash table has an insert function that takes an ID and all of the package parameters as input and stores those parameters in the hash table. The hash table also has a lookup function that allows me to pull all package data using the ID alone.

# H1. Other Data Structures

Two other data structures that could have been used to store package data are a list (array) or a BST.

# H1a. Data Structure Differences

A list implementation would simply append packages to a list to store their data. While lists and hash tables are both implemented using arrays, the key difference is that in a hash table, the index an item is stored at is dependent on a key, for example, an array containing packages such that the index of the package in the array is the package id is a hash table (this is how I implemented my hash table) while a list would simply append new packages to the end of the array assuming it is not already in the array. This means that the index of packages in a list would be independent of any of the parameters of the package, and lookups would take O(n) time, which is much less efficient than a hash table that has O(1) lookups. A BST implementation is similar to a list in that the location of the package in the tree is only partially dependent on the ID of the package, as insertion order now affects the location of packages in the tree. In the worst case, a lookup would be O(n), and in the best case, the lookup would be O(log n), which is better than a list but worse than a hash table.

# I. Sources

Lysecky, R., & Vahid, F. (2018, June). *C950: Data Structures and Algorithms II*. zyBooks.

Retrieved November 12, 2024, from <https://learn.zybooks.com/zybook/WGUC950AY20182019/>

*CSV - csv file reading and writing*. Python documentation. (n.d.). https://docs.python.org/3/library/csv.html

Retrieved November 12, 2024 from

<https://docs.python.org/3/library/csv.html>

Wikimedia Foundation. (2024, August 15). *2-opt*. Wikipedia. https://en.wikipedia.org/wiki/2-opt

Retrieved November 12, 2024 from

<https://en.wikipedia.org/wiki/2-opt>

*Nearest neighbour algorithm* (2024) *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Nearest\_neighbour\_algorithm.

Retrieved November 12, 2024 from

<https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm>

# J. Professional Communication