WGUPS Write-Up

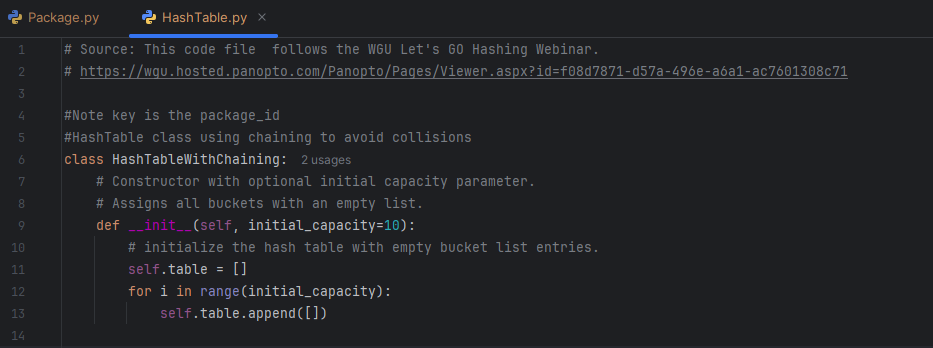
The implementation phase of the WGUPS Routing Program

Jacob Vierstra

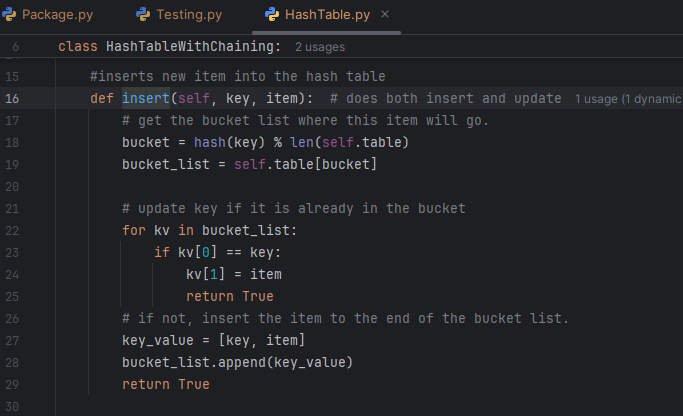
08/31/24

Data Structures and Algorithms II

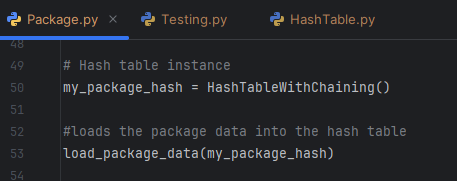
# A. Hash Table



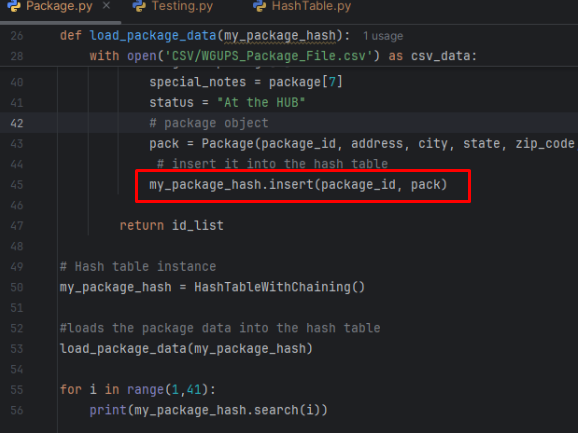
This is the Hash Table with the Chaining class initialization code.



This is the function to insert or update data into the hash table.

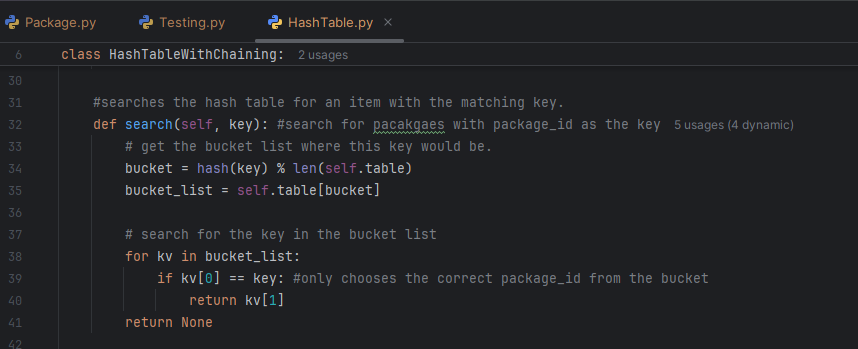


Creates the instance of the hash table and inserts data in the load\_package\_data function

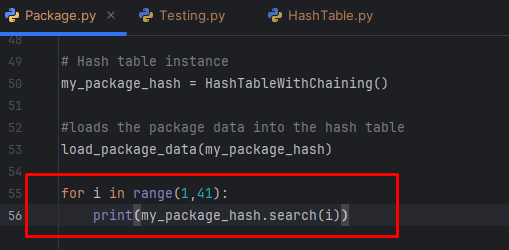


The package data is inserted into the hash table.

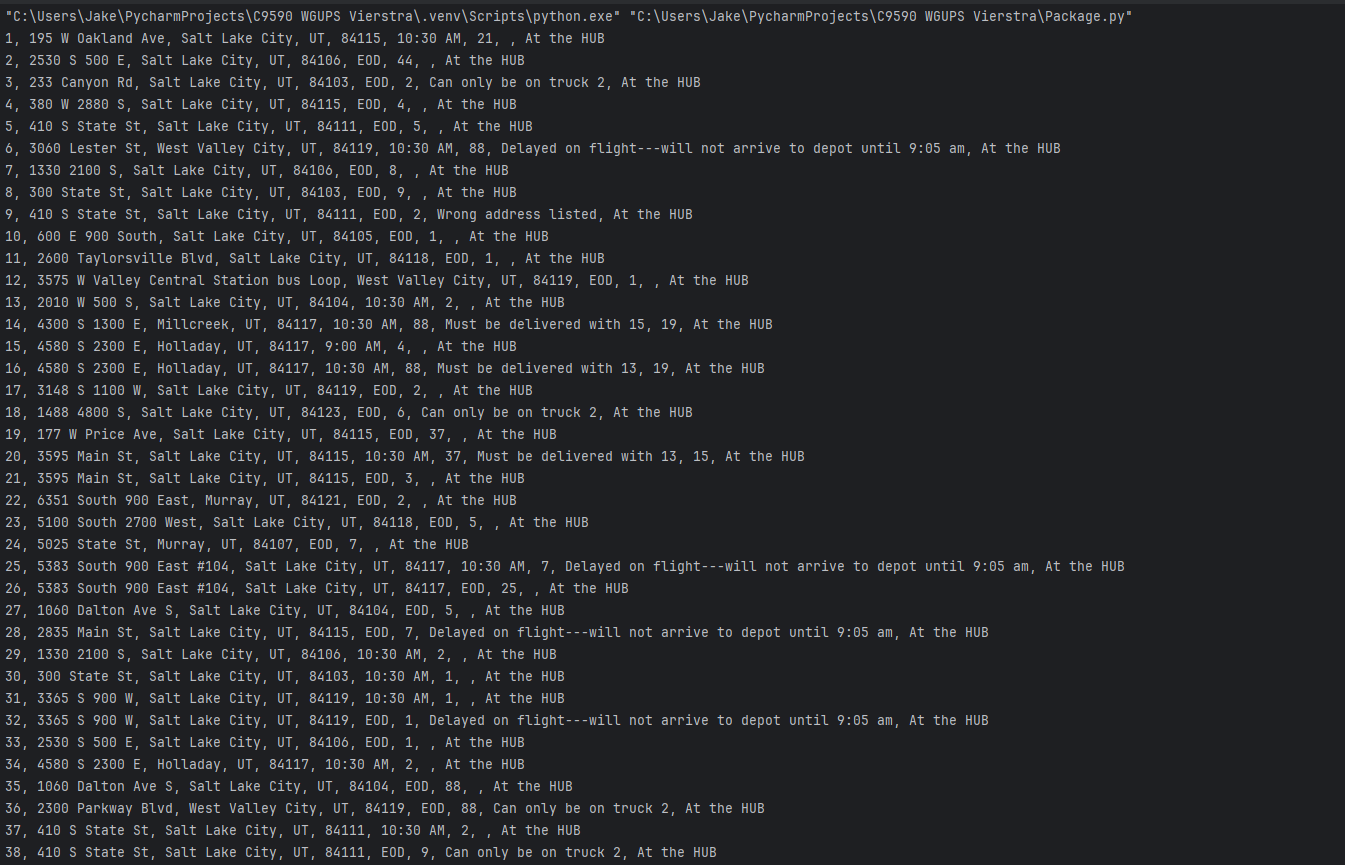
# B. Look-Up Functions

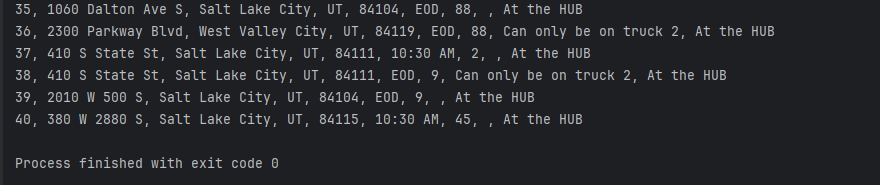


This is the code for the look-up function, which is titled search.



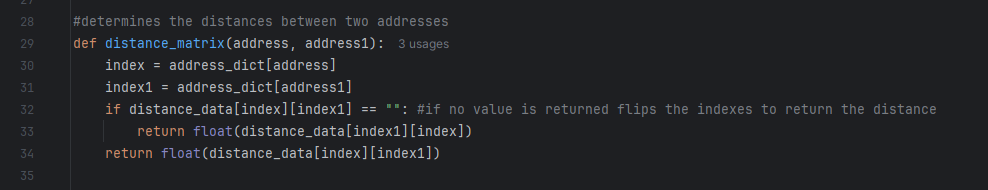
The look-up function or search function is used to display the info for all 40 packages in the hash table.



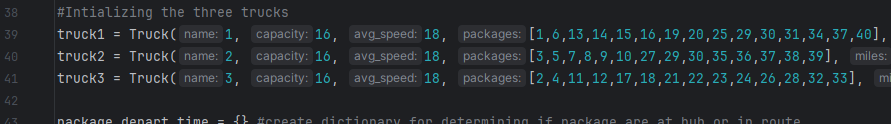


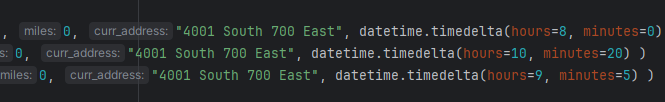
Here is the result of searching the hash table for all 40 packages.

# C. Original Code



This is the function distance\_matrix, which is used to determine the distance between addresses.



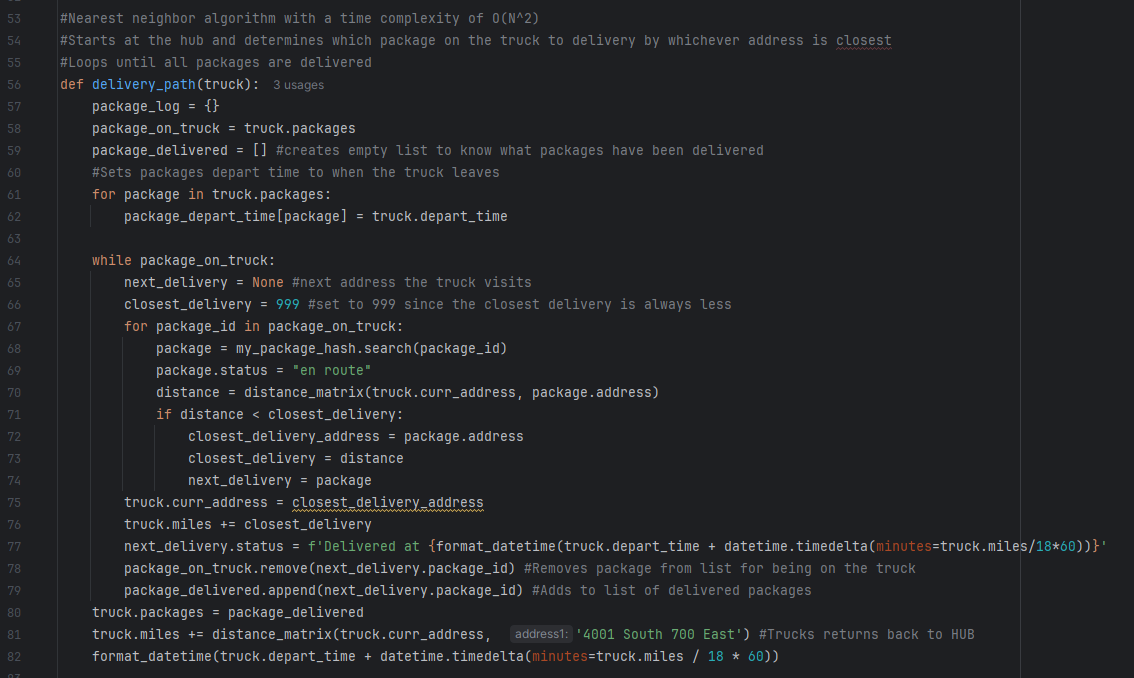
(same three lines of code in two screenshots for clarity)

This is what manually loads the trucks with the various packages.

A screenshot of a computer

Description automatically generated

This function is to determine what package a truck is on.



This is the nearest neighbor algorithm that gets the delivery path.

A screen shot of a computer program

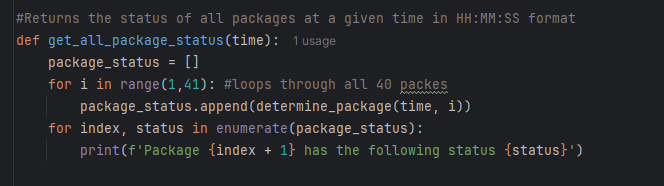
Description automatically generated

This determines the package status and stores the delivery time of the package with the status. Also, it updates the package 9 address at 10:20:00.

A screen shot of a computer code

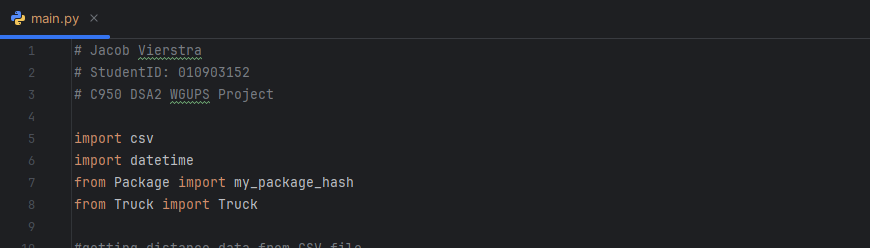
Description automatically generated

This is the function to print the package status of a particular package at any given time.



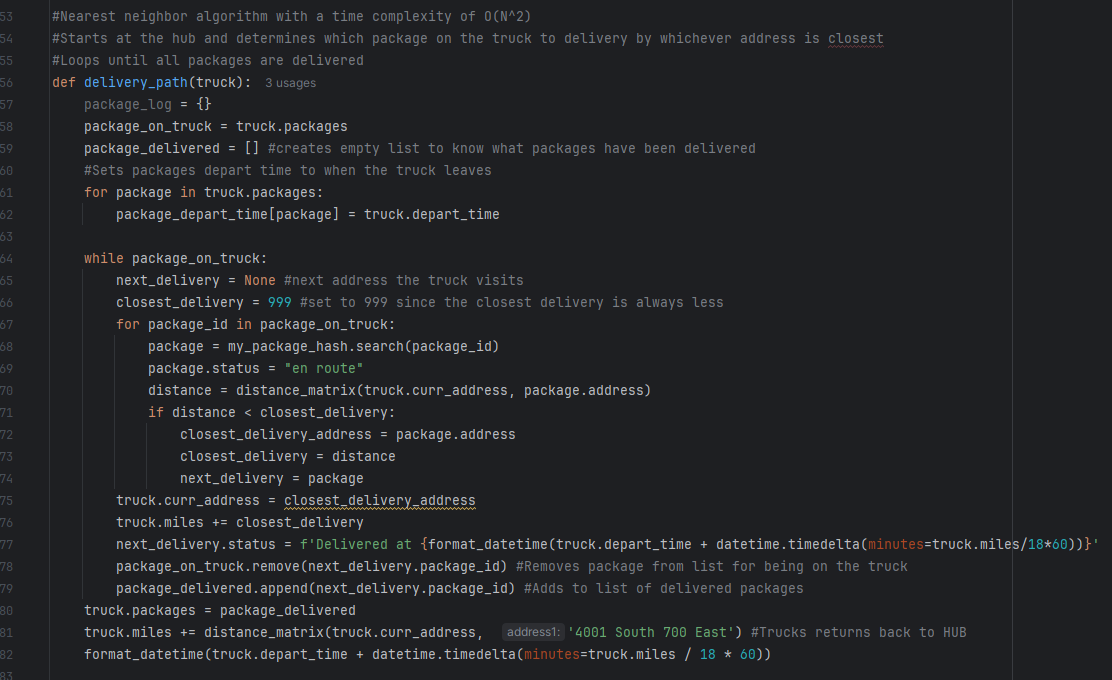
This function prints all the package statuses at any given time.

# C1. Identification Information

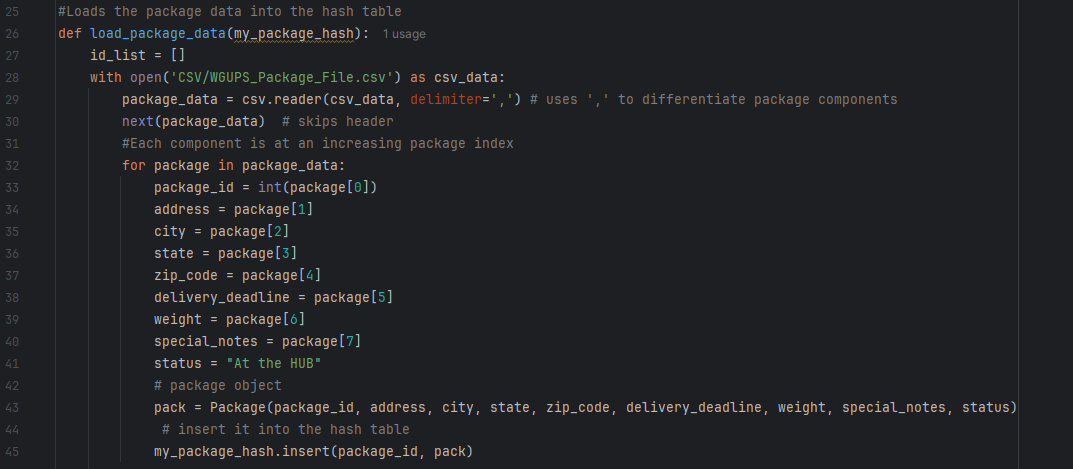


Main.py screenshot with student ID.

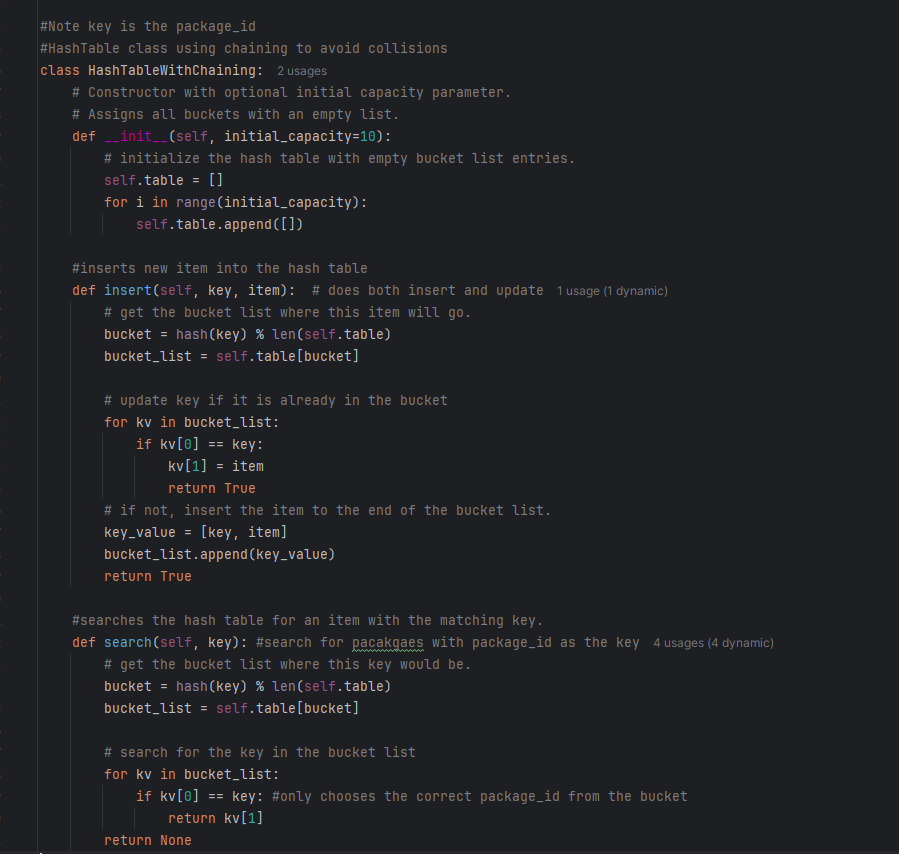
# C2. Process and Flow Comments



Code block of Nearest Neighbor algorithm with comments



Code block of reading data from CSV files. Comments help give an understanding of what is going on. For example, what does the delimiter do, and what are the package indices for? The comments aid in that.



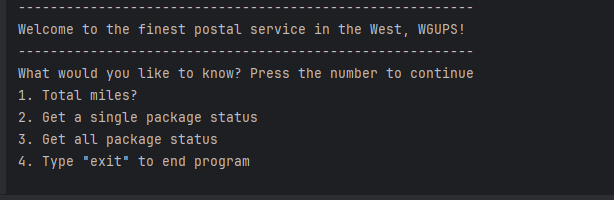
Code block of Hash Table functions. They are commented on to give understanding and help. For instance, stating that the key value is the package ID so the reader knows explicitly or stating that the insert function does insertion and updating.

# D. Interface

A computer screen shot of a program code

Description automatically generated

Here is the code for the command line interface. The program keeps running until the user types 'exit', which terminates the program.



This is what the command line interface prompts the user to get information.

# D1. First Status Check

A screenshot of a computer program

Description automatically generated

A screen shot of a computer screen

Description automatically generated

Here is the first status check of all the packages at 09:00:00. Notice that package 9 still has the wrong address.

# D2. Second Status Check

A screenshot of a computer

Description automatically generated

A screen shot of a computer screen

Description automatically generated

Here is the status check for all packages at 10:20:00. The first truck has returned, and the driver leaves in truck two at 10:20:00. The third truck left at 9:05. Notice that package 9 has been updated to the correct address at 10:20:00.

# D3. Third Status Check

A screenshot of a computer

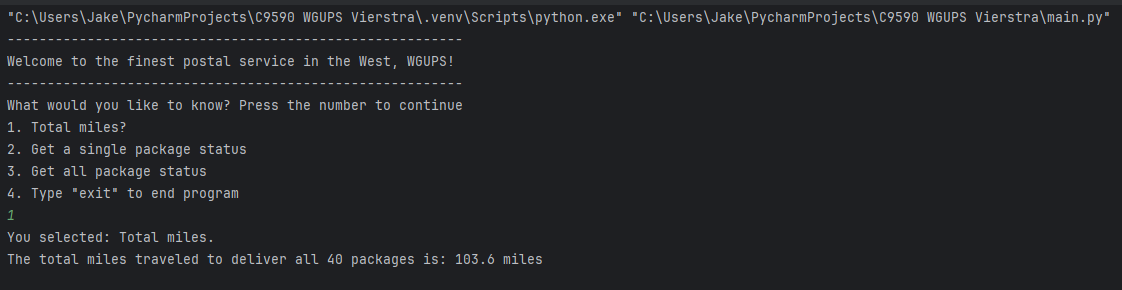
Description automatically generated

A screen shot of a computer screen

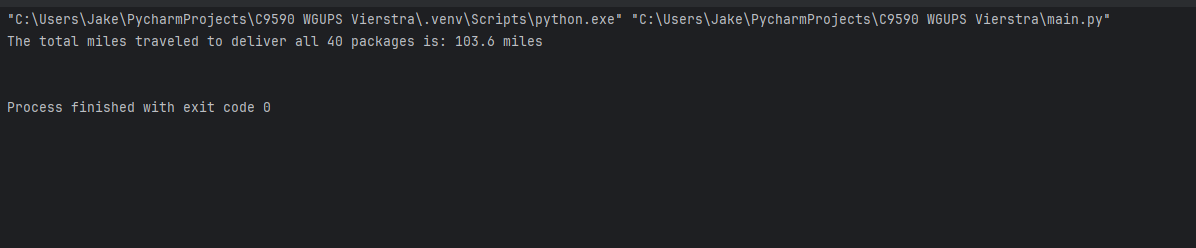
Description automatically generated

Here is the third status check, which is at 1:00 pm or 13:00:00. All packages have been delivered by this time, and all trucks are back at the hub.

# E. Screenshot of Code Execution



Here is the code being executed, and the total miles are displayed through the CLI.



Additionally, here is the code being executed without the CLI, which displays the total truck miles of all three trucks.

# F1. Strengths of the Chosen Algorithm

There are many strengths of using the greedy nearest neighbor algorithm. First is the simplicity and ease of implementation. It was implemented as a For Loop nested in a While Loop and is very intuitive. Anyone with a basic understanding of data structures and algorithms will recognize the structure of it. Another advantage is it is efficient. With a time complexity of O(N^2), it doesn't take long to determine the path of a truck. Lastly, it doesn't have a low space complexity of O(N).

# F2. Verification of Algorithm

There are requirements for the algorithm to deliver all 40 packages on time while still meeting the package requirements and have a combined total distance travel of all three trucks under 140 miles. The nearest neighbor algorithm meets all the requirements for this. All the packages meet the requirements. Anything needed to be delivered by 10:30 is on truck one, which has its last package delivered at 9:48. Additionally, all packages that are grouped together, including packages 13,14,15,16,19 and 20, are all together on truck 1. Truck 3 waits for delayed packages and departs at 9:05. Truck 2 has all packages required on the truck, waits till a driver returns, and waits till package nine's wrong address is updated to depart at 10:20. Lastly, the total miles of all three trucks is 103.6 miles which is well under the 140-mile limit.

# F3. Other Possible Algorithms

There are many other possible algorithms that could have been used. The first that could have been used is a depth-first search or brute force algorithm. It would have certainly been within the limits as it would have the most optimal solution, meaning the lowest miles and, in turn, the fastest delivery. It would have taken a much longer time to determine the path being a time complexity of O(N!), but the speed of the algorithm is not a requirement. Another difference is the data structure used for each, as the nearest neighbor uses a hash table, and depth-first search would use a stack, array/set/map, and graph representation. Another possible algorithm that could have been used is Dijkstra's algorithm. It's ideal for shortest paths in graphs, which makes it a great choice for this instance. As a result, Dijkstra's algorithm will certainly meet the requirements of being 140 miles and package delivery deadlines.

# F3a. Algorithm Differences

There are many differences between the nearest neighbor and a depth-first search algorithm. First is the time complexity; the nearest neighbor is O(N^2) in our case, where the depth-first search is O(N!). Another difference between these is the

There are many differences between the nearest neighbor and Dijkstra's. First, how does the algorithm navigate to the next address? Nearest neighbor checks all other addresses and picks the shortest one, whereas Dijkstra's the shortest path for vertices, which are addressed in this case, and the nodes or values in our distance table. As a result, Dijkstra's algorithm guaranteed that the shortest path and nearest neighbor were good enough for a solution but not the best.

# G. Different Approach

In doing this project again, I would do many things differently. First, I would make the interface a GUI instead of a CLI. The CLI gets the job, but having a visual representation of package status would be very informative. Furthermore, I would also add more functions to the interface regardless of if I used a GUI or CLI. It is very primitive, and I am unable to tell which package was delivered by truck in the interface. Another thing I would approach differently is how I loaded the trucks. Due to ease of implementation and time constraints, I opted to manually load the trucks. By loading the trucks with an algorithm, I could have likely achieved a much shorter total travel distance.

# H. Verification of Data Structure

The data structure of a hash table with chaining meets the requirements of being a self-adjusting data structure, has no collision, and develops with any additional libraries or classes. While Python has built-in functions for the hash table, I manually created the hash table without it. I also defined all the functions to manipulate the hash table with insert, update, search, and remove functionality. Also, in order to handle collisions, I implemented chaining so that the keys that point to the same bucket still have distinct data. Lastly, it is self-adjusting as it is able to handle any number of inputs. For example, it is a hash table with ten buckets, but there are 40 packages in it.

# H1. Other Data Structures

There are many other data structures that meet the requirements. The first that could have been used is a min-heap. It is already self-adjusting due to the heap properties, and the heap properties also ensure there are no collisions. It also goes great with Dijkstra's algorithms, which would have been easier to implement than other algorithms and data structures. The second is a linked list. Due to the nature of linked lists, there is no worry about collisions, and self-adjusting isn't an issue as well.

# H1a. Data Structure Differences

There are many differences between a min heap and a hash table. First is the dealing with collisions. Min heap automatically prevents collisions due to their properties, whereas hash tables must add something, like chaining, in order to handle collisions properly. Another difference is data manipulation times. Hash tables are much faster at an average O(1) for search, insertion, and deletion, whereas min heap is an average O(1) for search but O(log N) for insertion and deletion.

There are many differences between a hashtable and a linked list. First is the data manipulation times of each. Once again, hash tables are extremely fast at search, insertion, and deletion with an average time of O(1), whereas a linked list is an average of O(1) for insertion and deletion but much slower at search and traversal, which have an average time of O(N). Another difference is that a linked list maintains the order of elements as they were entered, whereas a table places elements somewhat randomly based on key values.

# I. Sources

No outside sources were used.