C950 Task-2 WGUPS Write-Up

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

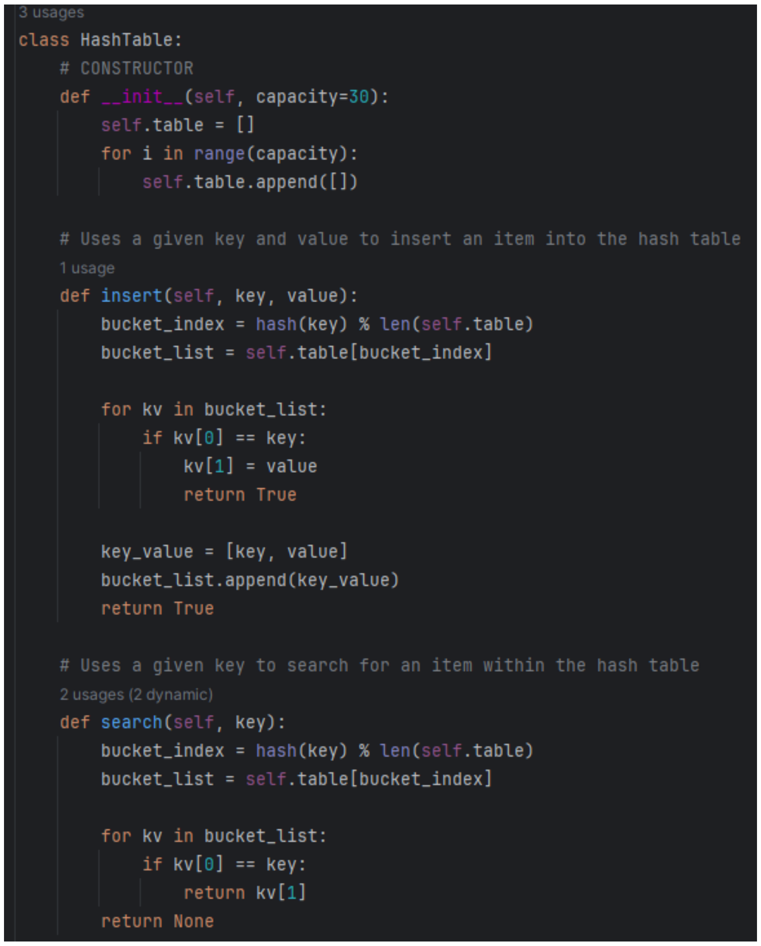
(Zip your source code and upload it with this file)

Allyson Quilter
ID #010186766
WGU Email: aquilt1@wgu.edu

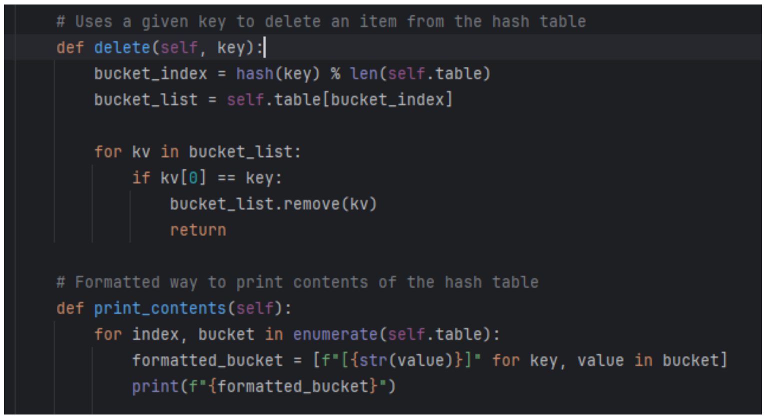
Date: 12/1/2023

C950 Data Structures and Algorithms II

A. Hash Table

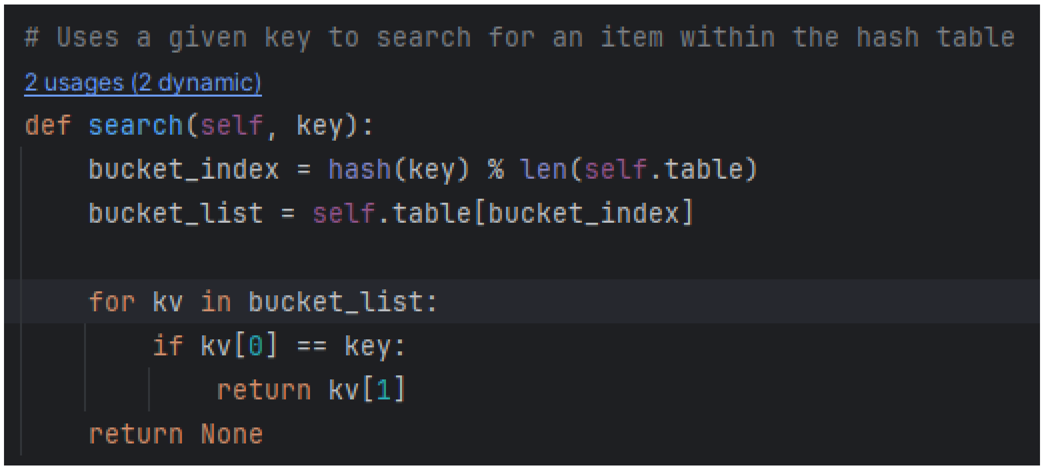


```
3 usages
class HashTable:
    # CONSTRUCTOR
    def __init__(self, capacity=30):
        self.table = []
        for i in range(capacity):
            self.table.append([])

    # Uses a given key and value to insert an item into the hash table
    1 usage
    def insert(self, key, value):
        bucket_index = hash(key) % len(self.table)
        bucket_list = self.table[bucket_index]

        for kv in bucket_list:
            if kv[0] == key:
                kv[1] = value
                return True

        key_value = [key, value]
        bucket_list.append(key_value)
        return True

    # Uses a given key to search for an item within the hash table
    2 usages (2 dynamic)
    def search(self, key):
        bucket_index = hash(key) % len(self.table)
        bucket_list = self.table[bucket_index]

        for kv in bucket_list:
            if kv[0] == key:
                return kv[1]
        return None
```



```
    # Uses a given key to delete an item from the hash table
    def delete(self, key):
        bucket_index = hash(key) % len(self.table)
        bucket_list = self.table[bucket_index]

        for kv in bucket_list:
            if kv[0] == key:
                bucket_list.remove(kv)
                return

    # Formatted way to print contents of the hash table
    def print_contents(self):
        for index, bucket in enumerate(self.table):
            formatted_bucket = [f"[{str(value)}]" for key, value in bucket]
            print(f"{formatted_bucket}")
```

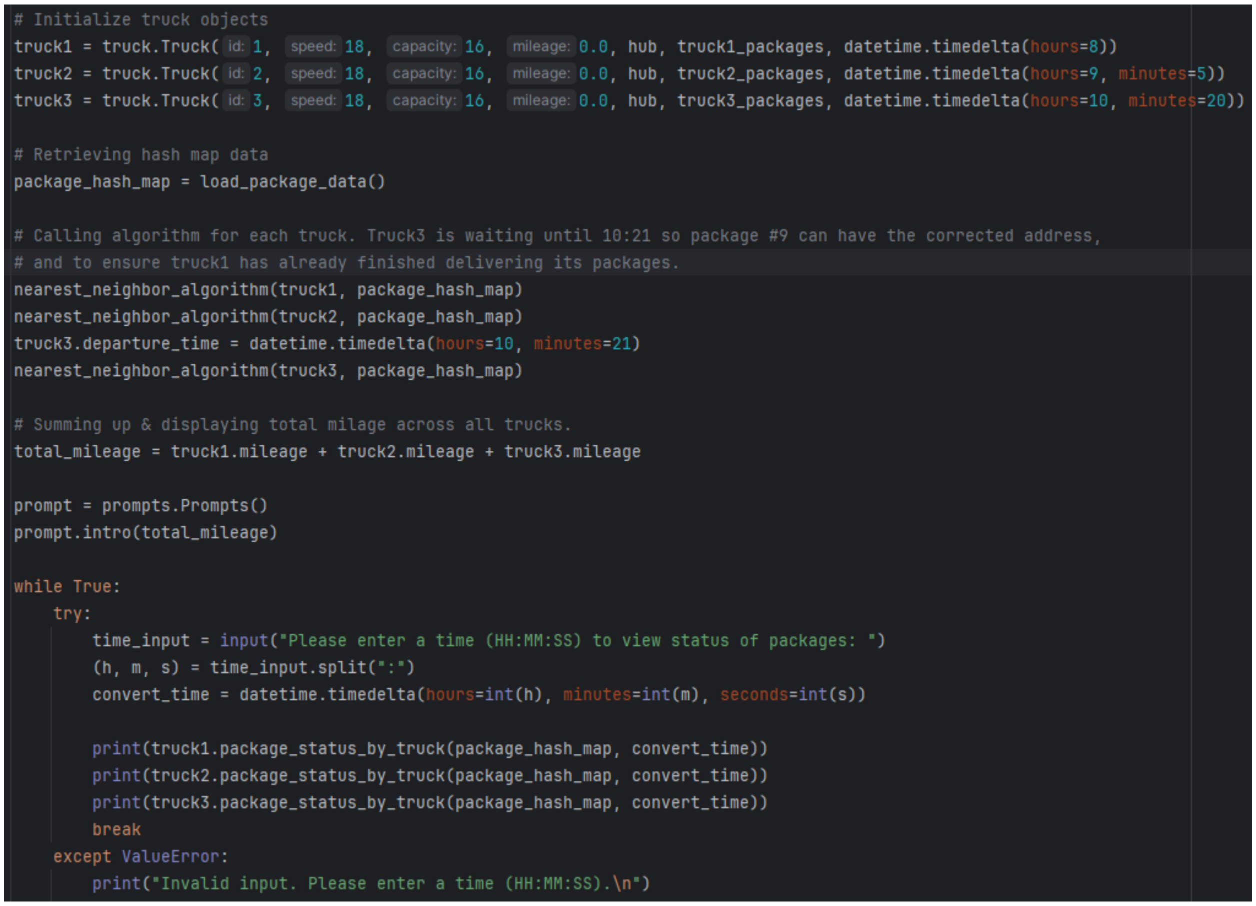
B. Look-Up Functions



```
# Uses a given key to search for an item within the hash table
2 usages (2 dynamic)
def search(self, key):
    bucket_index = hash(key) % len(self.table)
    bucket_list = self.table[bucket_index]

    for kv in bucket_list:
        if kv[0] == key:
            return kv[1]
    return None
```

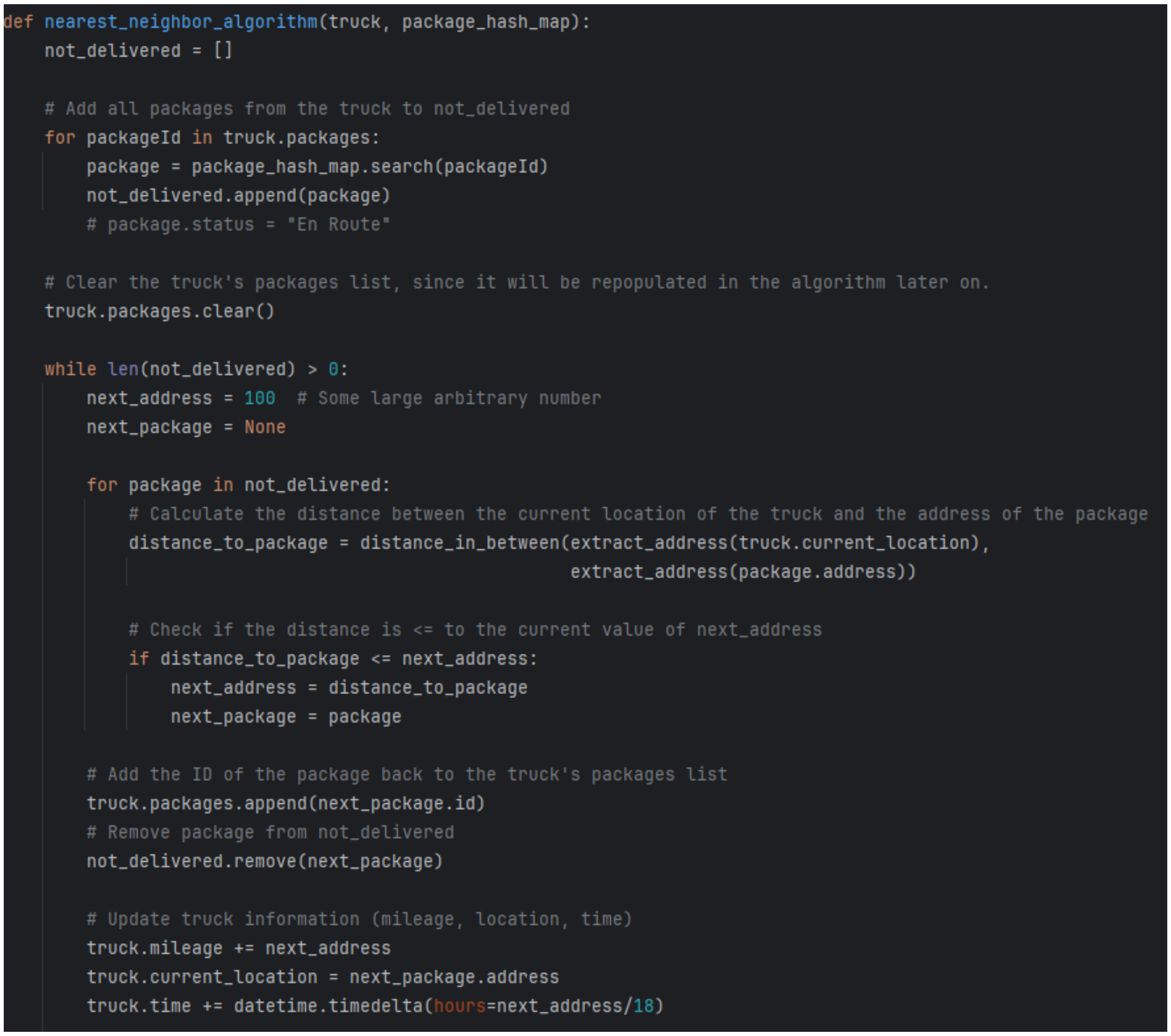
C. Original Code

Main.py:

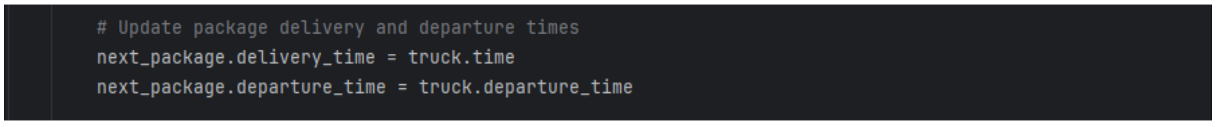


```
# Initialize truck objects
truck1 = truck.Truck( id: 1,  speed: 18,  capacity: 16,  mileage: 0.0, hub, truck1_packages, datetime.timedelta(hours=8))
truck2 = truck.Truck( id: 2,  speed: 18,  capacity: 16,  mileage: 0.0, hub, truck2_packages, datetime.timedelta(hours=9, minutes=5))
truck3 = truck.Truck( id: 3,  speed: 18,  capacity: 16,  mileage: 0.0, hub, truck3_packages, datetime.timedelta(hours=10, minutes=20))

# Retrieving hash map data
package_hash_map = load_package_data()

# Calling algorithm for each truck. Truck3 is waiting until 10:21 so package #9 can have the corrected address,
# and to ensure truck1 has already finished delivering its packages.
nearest_neighbor_algorithm(truck1, package_hash_map)
nearest_neighbor_algorithm(truck2, package_hash_map)
truck3.departure_time = datetime.timedelta(hours=10, minutes=21)
nearest_neighbor_algorithm(truck3, package_hash_map)

# Summing up & displaying total milage across all trucks.
total_mileage = truck1.mileage + truck2.mileage + truck3.mileage

prompt = prompts.Prompts()
prompt.intro(total_mileage)

while True:
    try:
        time_input = input("Please enter a time (HH:MM:SS) to view status of packages: ")
        (h, m, s) = time_input.split(":")
        convert_time = datetime.timedelta(hours=int(h), minutes=int(m), seconds=int(s))

        print(truck1.package_status_by_truck(package_hash_map, convert_time))
        print(truck2.package_status_by_truck(package_hash_map, convert_time))
        print(truck3.package_status_by_truck(package_hash_map, convert_time))
        break
    except ValueError:
        print("Invalid input. Please enter a time (HH:MM:SS).\n")
```

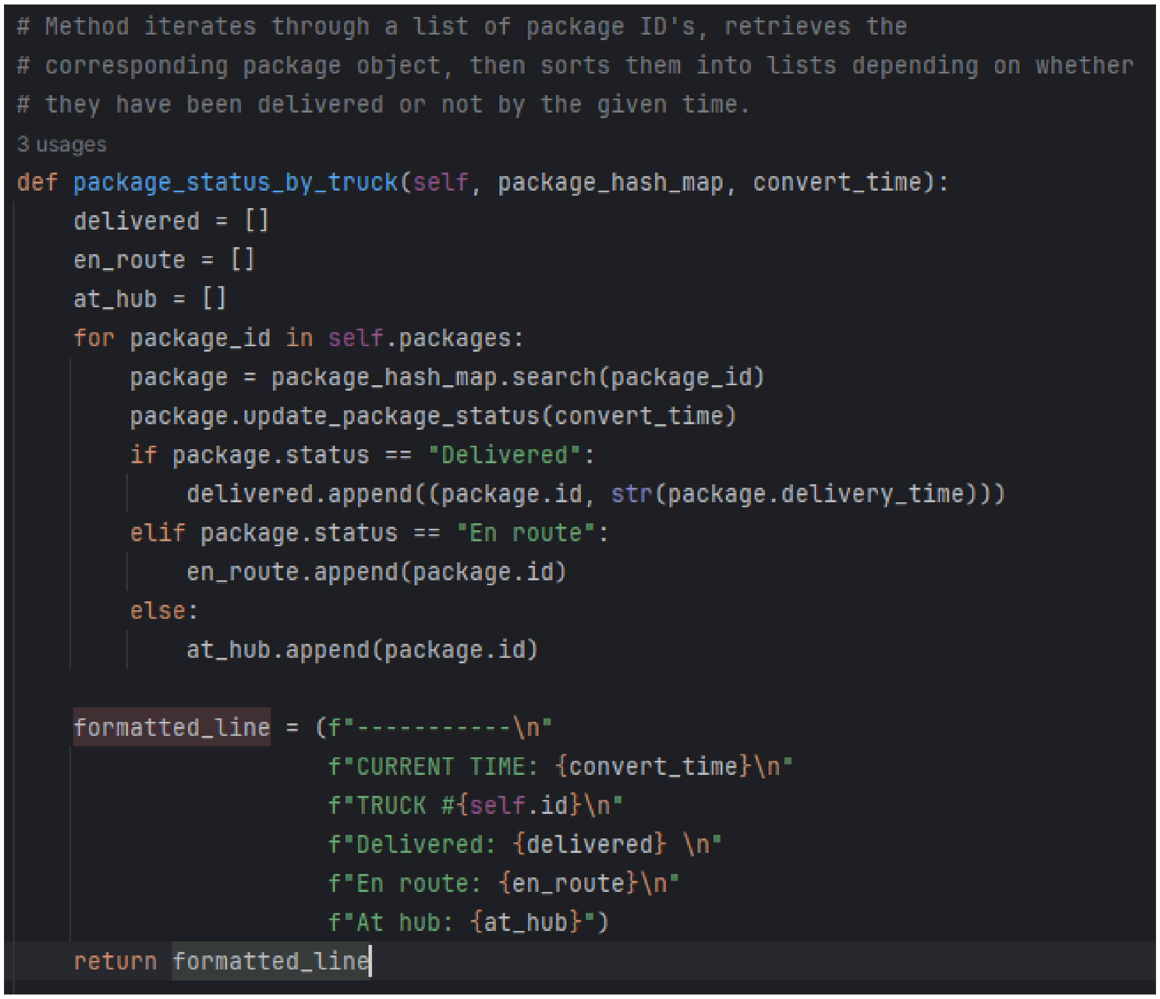
Distance.py:



```python
def nearest_neighbor_algorithm(truck, package_hash_map):
    not_delivered = []

    # Add all packages from the truck to not_delivered
    for packageId in truck.packages:
        package = package_hash_map.search(packageId)
        not_delivered.append(package)
        # package.status = "En Route"

    # Clear the truck's packages list, since it will be repopulated in the algorithm later on.
    truck.packages.clear()

    while len(not_delivered) > 0:
        next_address = 100  # Some large arbitrary number
        next_package = None

        for package in not_delivered:
            # Calculate the distance between the current location of the truck and the address of the package
            distance_to_package = distance_in_between(extract_address(truck.current_location),
                                                      extract_address(package.address))

            # Check if the distance is <= to the current value of next_address
            if distance_to_package <= next_address:
                next_address = distance_to_package
                next_package = package

        # Add the ID of the package back to the truck's packages list
        truck.packages.append(next_package.id)
        # Remove package from not_delivered
        not_delivered.remove(next_package)

        # Update truck information (mileage, location, time)
        truck.mileage += next_address
        truck.current_location = next_package.address
        truck.time += datetime.timedelta(hours=next_address/18)
```

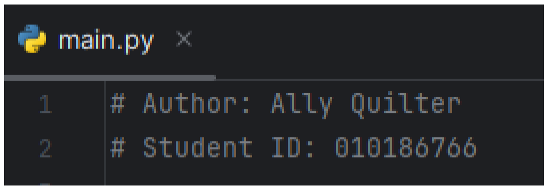


```python
        # Update package delivery and departure times
        next_package.delivery_time = truck.time
        next_package.departure_time = truck.departure_time
```



```
# Method iterates through a list of package ID's, retrieves the
# corresponding package object, then sorts them into lists depending on whether
# they have been delivered or not by the given time.
3 usages
def package_status_by_truck(self, package_hash_map, convert_time):
    delivered = []
    en_route = []
    at_hub = []
    for package_id in self.packages:
        package = package_hash_map.search(package_id)
        package.update_package_status(convert_time)
        if package.status == "Delivered":
            delivered.append((package.id, str(package.delivery_time)))
        elif package.status == "En route":
            en_route.append(package.id)
        else:
            at_hub.append(package.id)

    formatted_line = (f"-----------\n"
                      f"CURRENT TIME: {convert_time}\n"
                      f"TRUCK #{self.id}\n"
                      f"Delivered: {delivered} \n"
                      f"En route: {en_route}\n"
                      f"At hub: {at_hub}")
    return formatted_line
```

C1. Identification Information



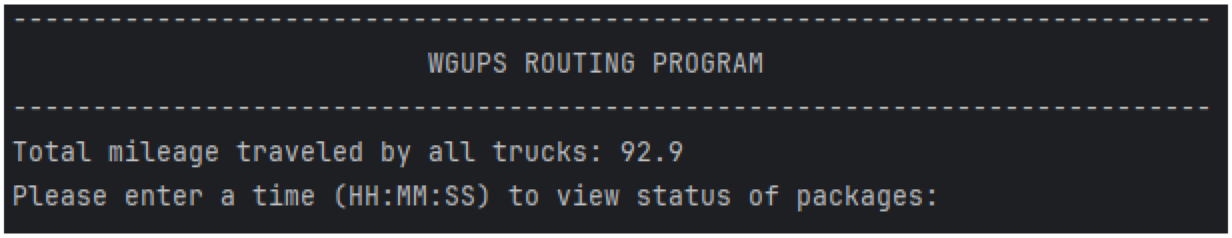
main.py

```
# Author: Ally Quilter
# Student ID: 010186766
```

C2. Process and Flow Comments

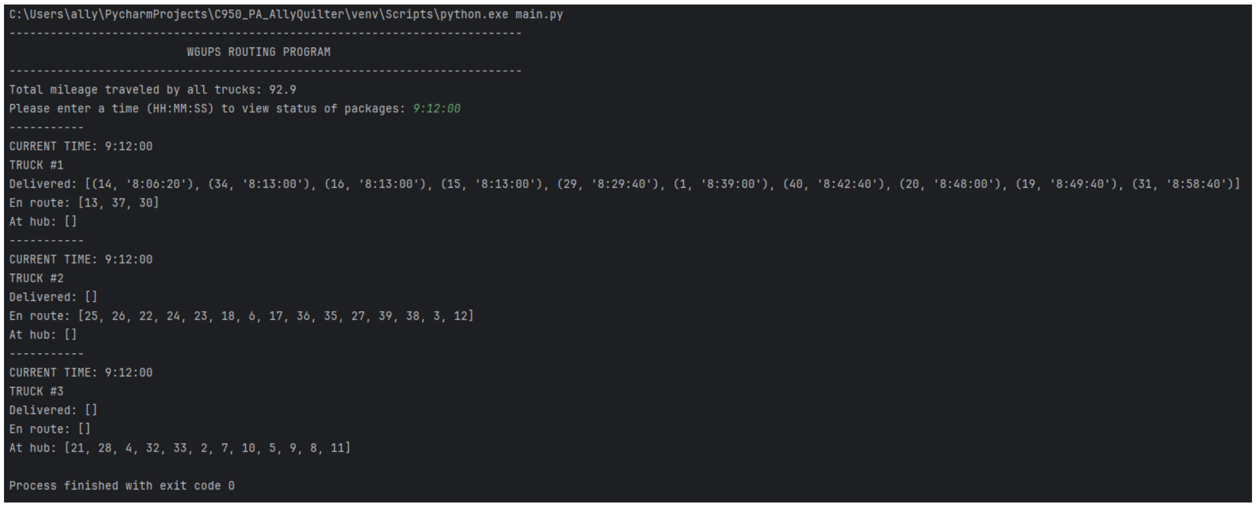
Please see screenshots for evidence of code comments.

D. Interface



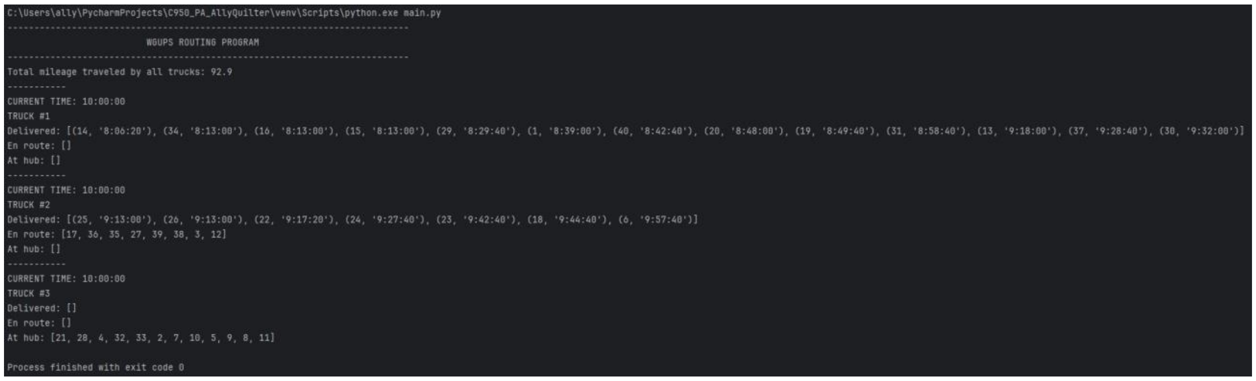
```
---------------------------------------------------------------------------
                         WGUPS ROUTING PROGRAM
---------------------------------------------------------------------------
Total mileage traveled by all trucks: 92.9
Please enter a time (HH:MM:SS) to view status of packages:
```

D1. First Status Check



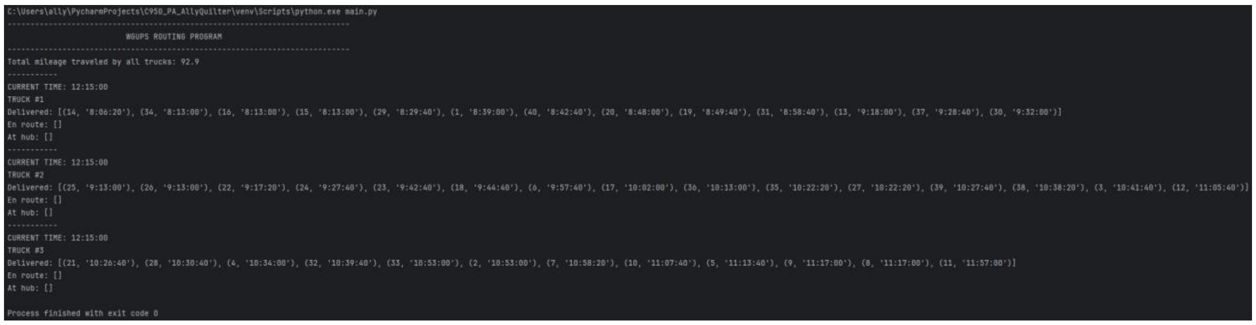
```
C:\Users\ally\PycharmProjects\C950_PA_AllyQuilter\venv\Scripts\python.exe main.py
---------------------------------------------------------------------------
                         WGUPS ROUTING PROGRAM
---------------------------------------------------------------------------
Total mileage traveled by all trucks: 92.9
Please enter a time (HH:MM:SS) to view status of packages: 9:12:00
-----------
CURRENT TIME: 9:12:00
TRUCK #1
Delivered: [(14, '8:06:20'), (34, '8:13:00'), (16, '8:13:00'), (15, '8:13:00'), (29, '8:29:40'), (1, '8:39:00'), (40, '8:42:40'), (20, '8:48:00'), (19, '8:49:40'), (31, '8:58:40')]
En route: [13, 37, 30]
At hub: []
-----------
CURRENT TIME: 9:12:00
TRUCK #2
Delivered: []
En route: [25, 26, 22, 24, 23, 18, 6, 17, 36, 35, 27, 39, 38, 3, 12]
At hub: []
-----------
CURRENT TIME: 9:12:00
TRUCK #3
Delivered: []
En route: []
At hub: [21, 28, 4, 32, 33, 2, 7, 10, 5, 9, 8, 11]

Process finished with exit code 0
```

D2. Second Status Check



```
C:\Users\ally\PycharmProjects\C950_PA_AllyQuilter\venv\Scripts\python.exe main.py
---------------------------------------------------------------------------
                         WGUPS ROUTING PROGRAM
---------------------------------------------------------------------------
Total mileage traveled by all trucks: 92.9
-----------
CURRENT TIME: 10:00:00
TRUCK #1
Delivered: [(14, '8:06:20'), (34, '8:13:00'), (16, '8:13:00'), (15, '8:13:00'), (29, '8:29:40'), (1, '8:39:00'), (40, '8:42:40'), (20, '8:48:00'), (19, '8:49:40'), (31, '8:58:40'), (13, '9:18:00'), (37, '9:28:40'), (30, '9:32:00')]
En route: []
At hub: []
-----------
CURRENT TIME: 10:00:00
TRUCK #2
Delivered: [(25, '9:13:00'), (26, '9:13:00'), (22, '9:17:20'), (24, '9:27:40'), (23, '9:42:40'), (18, '9:44:40'), (6, '9:57:40')]
En route: [17, 36, 35, 27, 39, 38, 3, 12]
At hub: []
-----------
CURRENT TIME: 10:00:00
TRUCK #3
Delivered: []
En route: []
At hub: [21, 28, 4, 32, 33, 2, 7, 10, 5, 9, 8, 11]

Process finished with exit code 0
```

D3. Third Status Check



```
C:\Users\ally\PycharmProjects\C950_PA_AllyQuilter\venv\Scripts\python.exe main.py
--------------------------------------------------------------------------
                         WGUPS ROUTING PROGRAM
--------------------------------------------------------------------------
Total mileage traveled by all trucks: 92.9
-----------
CURRENT TIME: 12:15:00
TRUCK #1
Delivered: [(14, '8:06:20'), (34, '8:13:00'), (16, '8:13:00'), (15, '8:13:00'), (29, '8:29:40'), (1, '8:39:00'), (40, '8:42:40'), (20, '8:48:00'), (19, '8:49:40'), (31, '8:58:40'), (13, '9:18:00'), (37, '9:28:40'), (30, '9:32:00')]
En route: []
At hub: []
-----------
CURRENT TIME: 12:15:00
TRUCK #2
Delivered: [(25, '9:13:00'), (26, '9:13:00'), (22, '9:17:20'), (24, '9:27:40'), (23, '9:42:40'), (18, '9:44:40'), (6, '9:57:40'), (17, '10:02:00'), (36, '10:13:00'), (35, '10:22:20'), (27, '10:22:20'), (39, '10:27:40'), (38, '10:38:20'), (3, '10:41:40'), (12, '11:05:40')]
En route: []
At hub: []
-----------
CURRENT TIME: 12:15:00
TRUCK #3
Delivered: [(21, '10:26:40'), (28, '10:30:40'), (4, '10:34:00'), (32, '10:39:40'), (33, '10:53:00'), (2, '10:53:00'), (7, '10:58:20'), (10, '11:07:40'), (5, '11:13:40'), (9, '11:17:00'), (8, '11:17:00'), (11, '11:57:00')]
En route: []
At hub: []

Process finished with exit code 0
```

E. Screenshot of Code Execution

See any of the status check screenshots for evidence that code ran successfully, as indicated by the project file location and exit code 0. (.png's are also included in the zip file)

F1. Strengths of the Chosen Algorithm

One strength of the nearest neighbor algorithm is the fact that it's greedy, which allows it to select the most ideal (aka nearest) package at each iteration. This made it relatively easy to implement compared to other algorithms, since I had a clear understanding of **how** it should behave. Another strength is its adaptability. I wouldn't have to adjust the algorithm at all if I ever needed to add more packages or trucks, since the algorithm should behave the same no matter the input size. However, it would eventually run slower at higher inputs since its space-time complexity is O(N^2).

F2. Verification of Algorithm

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

The algorithm meets all requirements, since iteratively selects the nearest undelivered package based on distance. It also calculates the total milage of the truck.

F3. Other Possible Algorithms

I believe Dijkstra's algorithm and K-means Clustering would meet the requirements.

F3a. Algorithm Differences

Dijkstra's algorithm also finds the shortest route but is different from NNA in that it considers the whole data set globally and guarantees the shortest route. NNA is similar, but it works more locally, selecting the nearest unvisited location at each step.

K-means Clustering involves grouping similar data points together and identify. This is helpful because it could potentially identify a new optimal location for the hub based on geographical information. Based on the hub location, K-means can then be applied to cluster delivery addresses so that a specific delivery truck can handle a set of deliveries within that cluster. (GeeksForGeeks, 2023)

G. Different Approach

If I had some more time to put into this project, I probably would have cleaned up my main.py file. For example, I feel like I could have put certain method calls/object initialization into their appropriate classes (such as putting the truck initialization and algorithm call inside the truck.py file). I would try to make the project more modular overall.

H. Verification of Data Structure

The hash table does meet all the requirements listed, which state that it should be able to take the package ID (key) and store all its data components (value). This is implemented with the insert() method. I also needed to make a look up function that takes the ID as input and returns all its associated data components. This function is also working and is implemented by using search().

H1. Other Data Structures

I believe a linked list or a (balanced) tree would meet the requirements.

H1a. Data Structure Differences

A linked list is a linear data structure has nodes that point to the next node. Each node can represent a package object, and a new package can be inserted or searched using a package ID. This is different from a hash table in that: it does not have key-value pairs, and insertion involves creating a new node for each package and updating the list accordingly.

Similar to linked list, you can also store package objects as nodes in a tree. You can also search and insert nodes to a tree, but you don't have direct access to a certain node like you do with a hash table. It is also more suited for hierarchical relationships, unlike hash tables.

I. Sources

"K Means Clustering - Introduction." *GeeksforGeeks*, GeeksforGeeks, 25 Aug. 2023, www.geeksforgeeks.org/k-means-clustering-introduction/.

J. Professional Communication

N/A