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)

Aaron Tandem

ID #010169998

WGU Email: atandem@wgu.edu

1/20/24

C950 Data Structures and Algorithms II

A. Hash Table

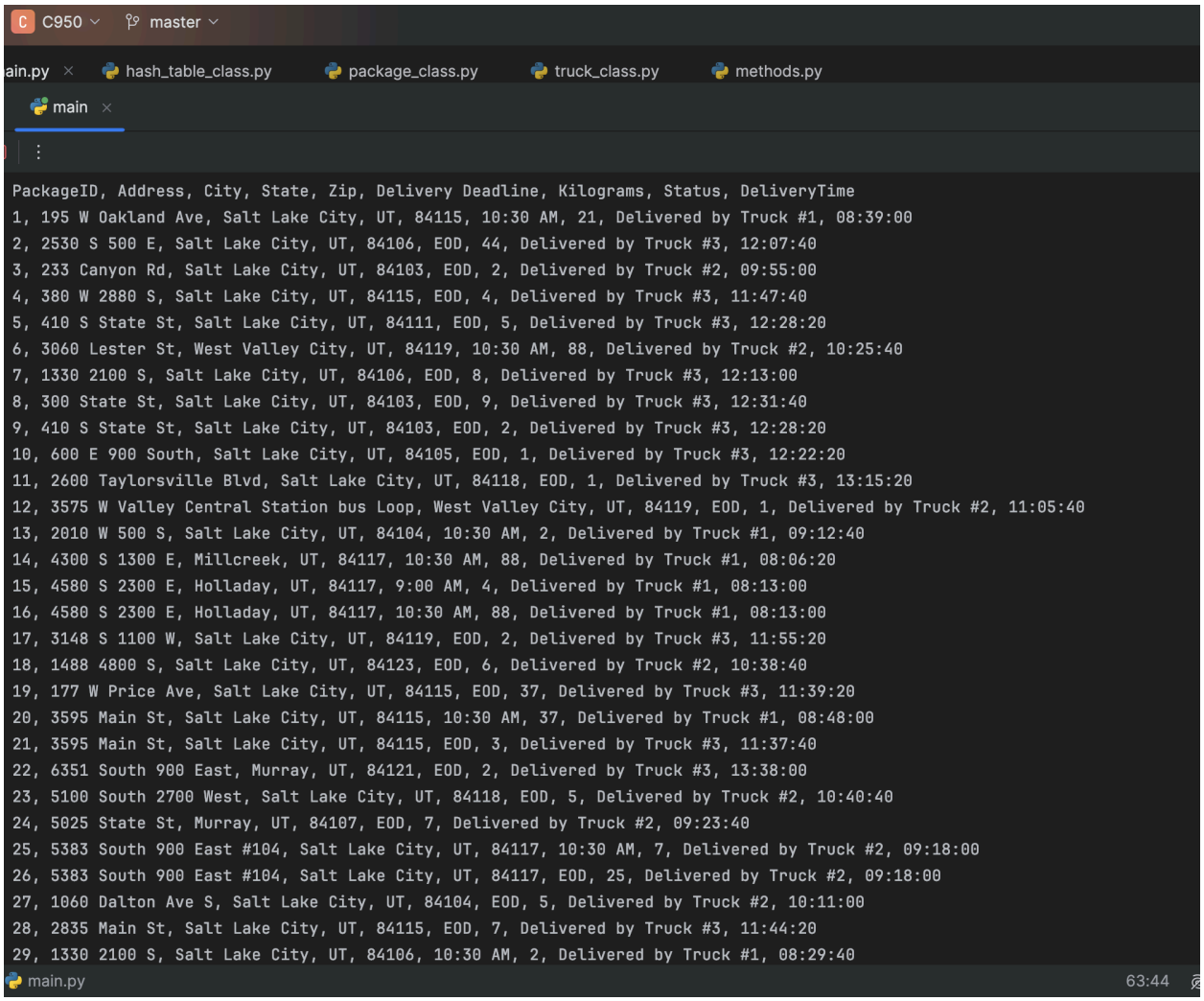
Below is a screenshot of the hash table's code: (full code can be viewed in the provided zip file)

```
102
103
104  # This is the instance of the self-adjusting data structure (hash table), followed by a method call to load it with package data
105  packageHashTable = hash_table_class.ChainingHashTable()
106  methods.loadPackageData( fileName: "WGUPS_package_file.csv", packageHashTable)
```

main.py | hash_table_class.py | package_class.py | truck_class.py | methods.py

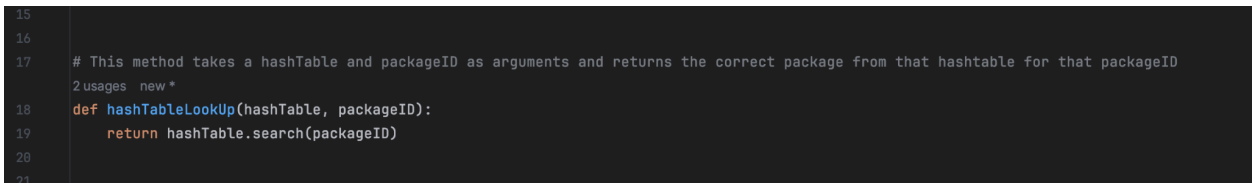
```
1   # C950 - Webinar-1 - Let's Go Hashing
2   # W-1_ChainingHashTable_zyBooks_Key-Value.py
3   # Ref: zyBooks: Figure 7.8.2: Hash table using chaining.
4   # Modified for Key:Value
5
6   # HashTable class using chaining.
    1 usage  Aaron Tandem
7   class ChainingHashTable:
8       # Constructor with optional initial capacity parameter.
9       # Assigns all buckets with an empty list.
        Aaron Tandem
10      def __init__(self, initial_capacity=40):
11          # initialize the hash table with empty bucket list entries.
12          self.table = []
13          for i in range(initial_capacity):
14              self.table.append([])
15
16      # Inserts a new item into the hash table.
        2 usages (2 dynamic)  Aaron Tandem
17      def insert(self, key, item):  # does both insert and update
18          # get the bucket list where this item will go.
19          bucket = hash(key) % len(self.table)
20          bucket_list = self.table[bucket]
21
22          # update key if it is already in the bucket
23          for kv in bucket_list:
24              # print (key_value)
25              if kv[0] == key:
26                  kv[1] = item
27                  return True
28
29          # if not, insert the item to the end of the bucket list.
```

Proof the hash table contains correct data, as seen in Option #1 of the user interface:

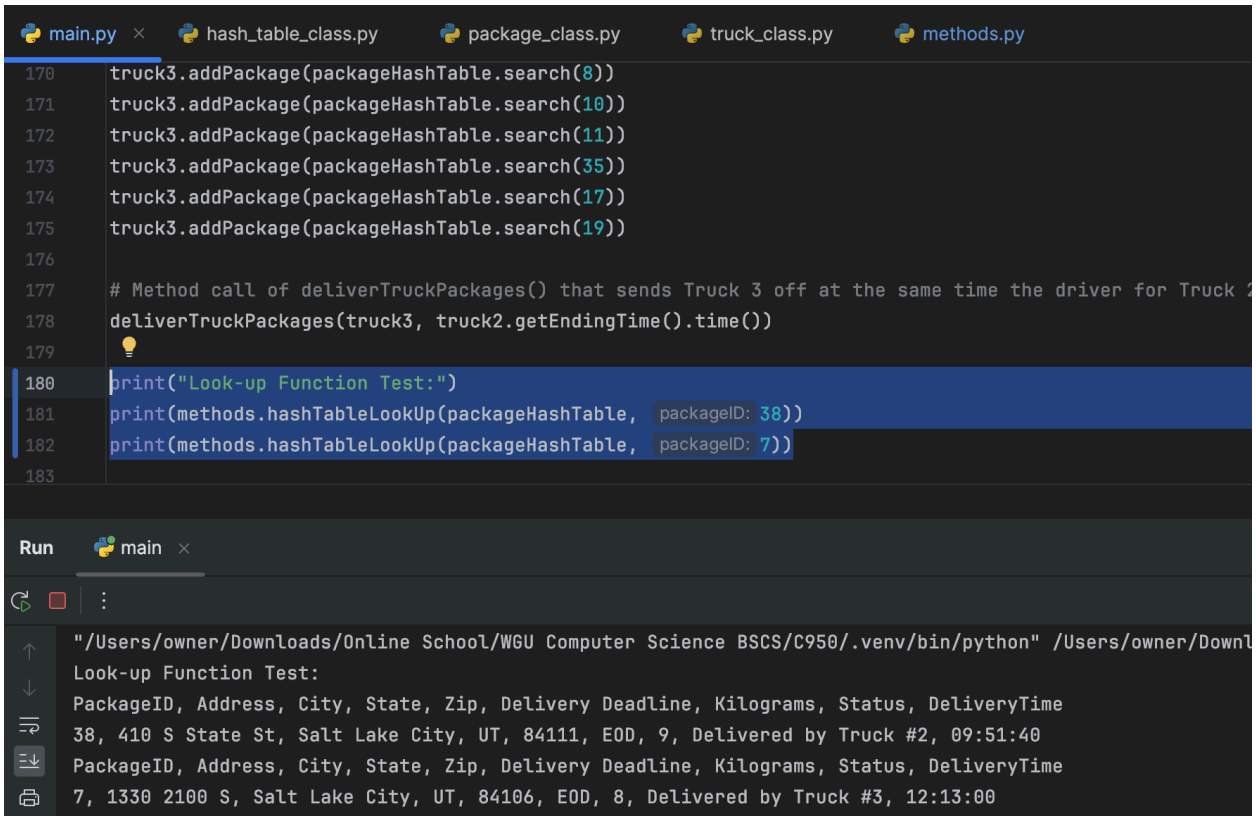


B. Look-Up Functions

Screenshot of Look-up function, as seen in **methods.py**:



Proof of functionality:



C. Original Code

Below are a few screenshots showing major blocks of code demonstrating implementation of this project: (full code can be viewed in the provided zip file for greater detail)

main.py:

main.py × hash_table_class.py package_class.py truck_class.py methods.py

```
# This method finds and returns how many miles there are between two provided addresses held within a list storing address data.
# It is subsequently used in the minDistanceFrom() method below to sequentially find which two addresses are closest to one another.
4 usages  Aaron Tandem
def distanceBetween(address1, address2):
    indexAddress1 = addressDataList.index(address1)
    indexAddress2 = addressDataList.index(address2)

    # The 2D list is a series of arrays of i++ size, so 'coordinates' will only work uni-directionally
    # e.x. distanceDataList[3][22] will return an integer representing distance, but distanceDataList[22][3] will throw an exception
    # However, because coordinates should be 'reversible', logic should be provided for cases when a coordinate is 'out-of-bounds'
    # So, this 'flips' the indexes when necessary so that the index reference never exceeds what the Address Data List allows for
    if indexAddress2 > indexAddress1:
        tempIndex = indexAddress1
        flippedIndexAddress1 = indexAddress2
        flippedIndexAddress2 = tempIndex
        return distanceDataList[flippedIndexAddress1][flippedIndexAddress2]

    else:
        return distanceDataList[indexAddress1][indexAddress2]


# This method makes use of the distanceBetween() function above to sequentially find the address of a package in a truck's list that is
# closest to the provided 'fromAddress' (i.e. 'where the truck was last located'). This 'fromAddress' generally always starts at the WGUPS
# HUB located at "4001 South 700 East, Salt Lake City, UT 84107". However, this address is automatically updated to the address of the last
# delivered package in the recursive call of minDistanceFrom() in the deliverTruckPackages() method below. This is a critical part of the
# Nearest Neighbor Greedy Algorithm used by this program to solve WGUPS's package delivery efficiency issues.
1 usage  Aaron Tandem
def minDistanceFrom(fromAddress, truckPackages):...
```

main.py × hash_table_class.py package_class.py truck_class.py methods.py

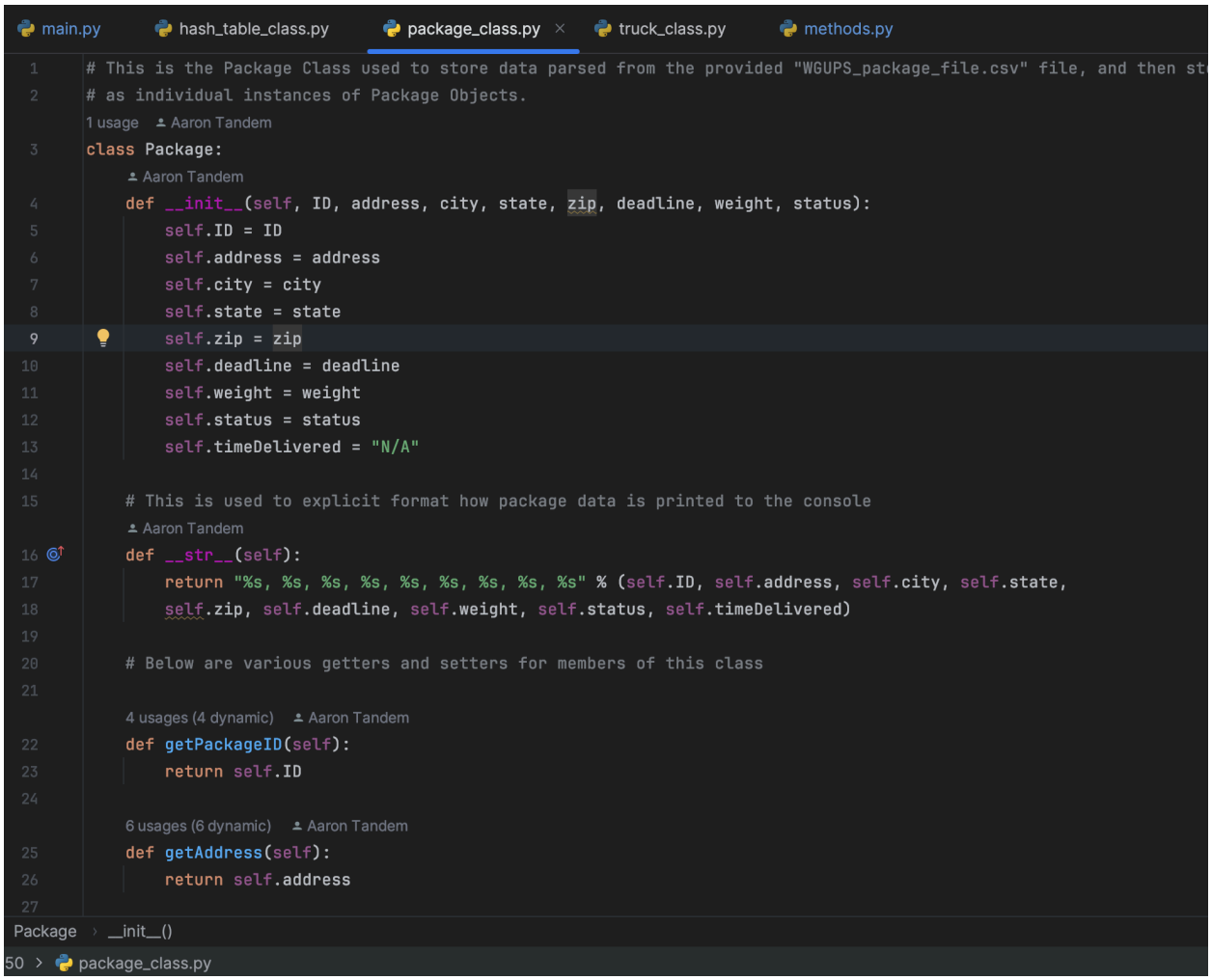
```
def deliverTruckPackages(truck, startTime):...


# This is the instance of the self-adjusting data structure (hash table), followed by a method call to load it with package data
packageHashTable = hash_table_class.ChainingHashTable()
methods.loadPackageData( fileName: "WGUPS_package_file.csv", packageHashTable)

# This is an empty list data structure that'll hold distances, followed by a method call to load the list with distance data
distanceDataList = []
methods.loadDistanceData( fileName: "WGUPS_distance_table.csv", distanceDataList)

# This is an empty list data structure that'll hold addresses, followed by a method call to load the list with address data
addressDataList = []
methods.loadAddressData( fileName: "WGUPS_distance_table.csv", addressDataList)

# Truck 1 will be the first truck to leave the HUB, prioritizing packages that have early deadlines
truck1 = truck_class.Truck(1)  # The constructor of the Truck Class is called to create a new truck object
# Packages are loaded manually by calling the truck's addPackage() along with the hash table's search functionality
truck1.addPackage(packageHashTable.search(1))
# packages 13, 14, 15, 16, 19, 20 must all be delivered together
truck1.addPackage(packageHashTable.search(13))
truck1.addPackage(packageHashTable.search(14))
truck1.addPackage(packageHashTable.search(15))
truck1.addPackage(packageHashTable.search(16))
truck1.addPackage(packageHashTable.search(20))
# other packages
truck1.addPackage(packageHashTable.search(29))
truck1.addPackage(packageHashTable.search(30))
truck1.addPackage(packageHashTable.search(31))
truck1.addPackage(packageHashTable.search(34))
truck1.addPackage(packageHashTable.search(37))
truck1.addPackage(packageHashTable.search(40))
```

hash_table_class.py:

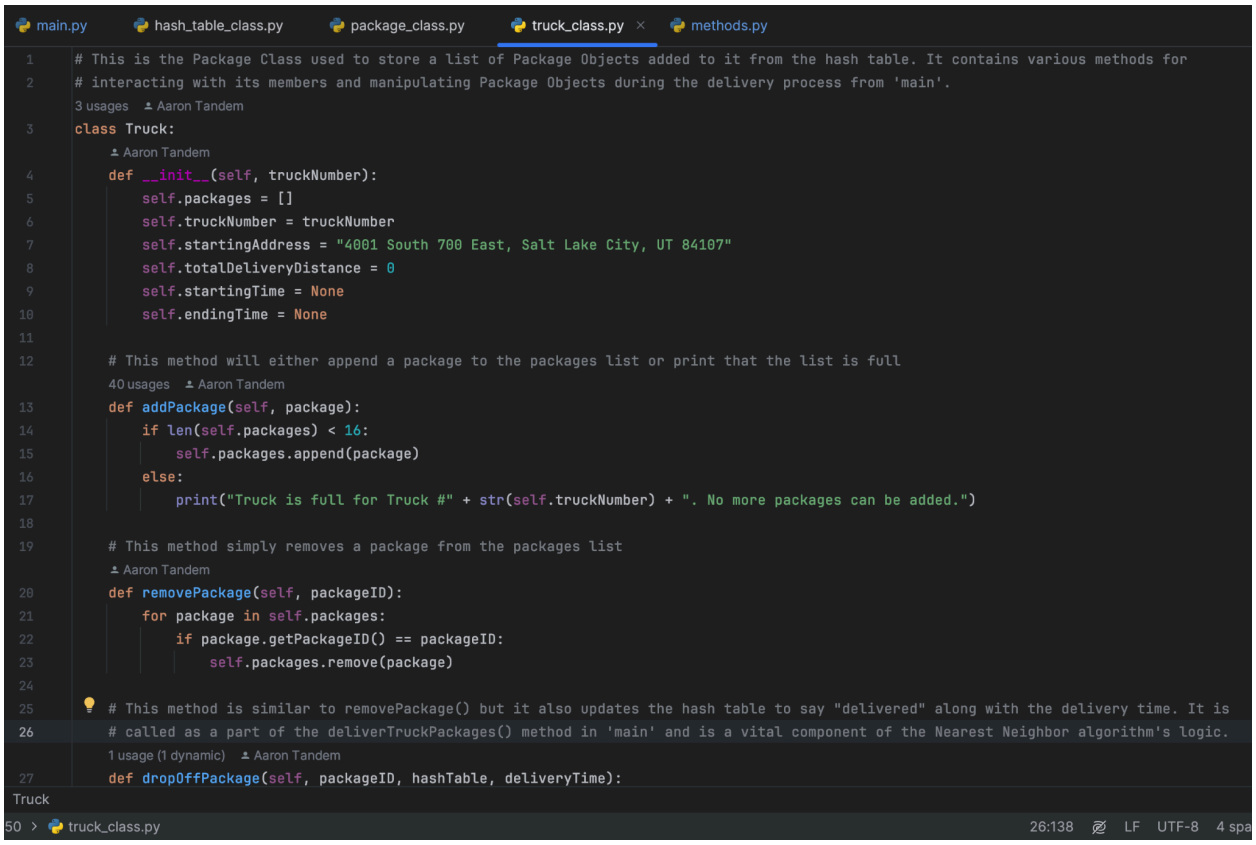
main.py | hash_table_class.py | package_class.py | truck_class.py | methods.py

```python
# C950 - Webinar-1 - Let's Go Hashing
# W-1_ChainingHashTable_zyBooks_Key-Value.py
# Ref: zyBooks: Figure 7.8.2: Hash table using chaining.
# Modified for Key:Value

# HashTable class using chaining.
1 usage  Aaron Tandem
class ChainingHashTable:
    # Constructor with optional initial capacity parameter.
    # Assigns all buckets with an empty list.
    Aaron Tandem
    def __init__(self, initial_capacity=40):
        # initialize the hash table with empty bucket list entries.
        self.table = []
        for i in range(initial_capacity):
            self.table.append([])

    # Inserts a new item into the hash table.
    2 usages (2 dynamic)  Aaron Tandem
    def insert(self, key, item):  # does both insert and update
        # get the bucket list where this item will go.
        bucket = hash(key) % len(self.table)
        bucket_list = self.table[bucket]

        # update key if it is already in the bucket
        for kv in bucket_list:
            # print (key_value)
            if kv[0] == key:
                kv[1] = item
                return True

        # if not, insert the item to the end of the bucket list.
```

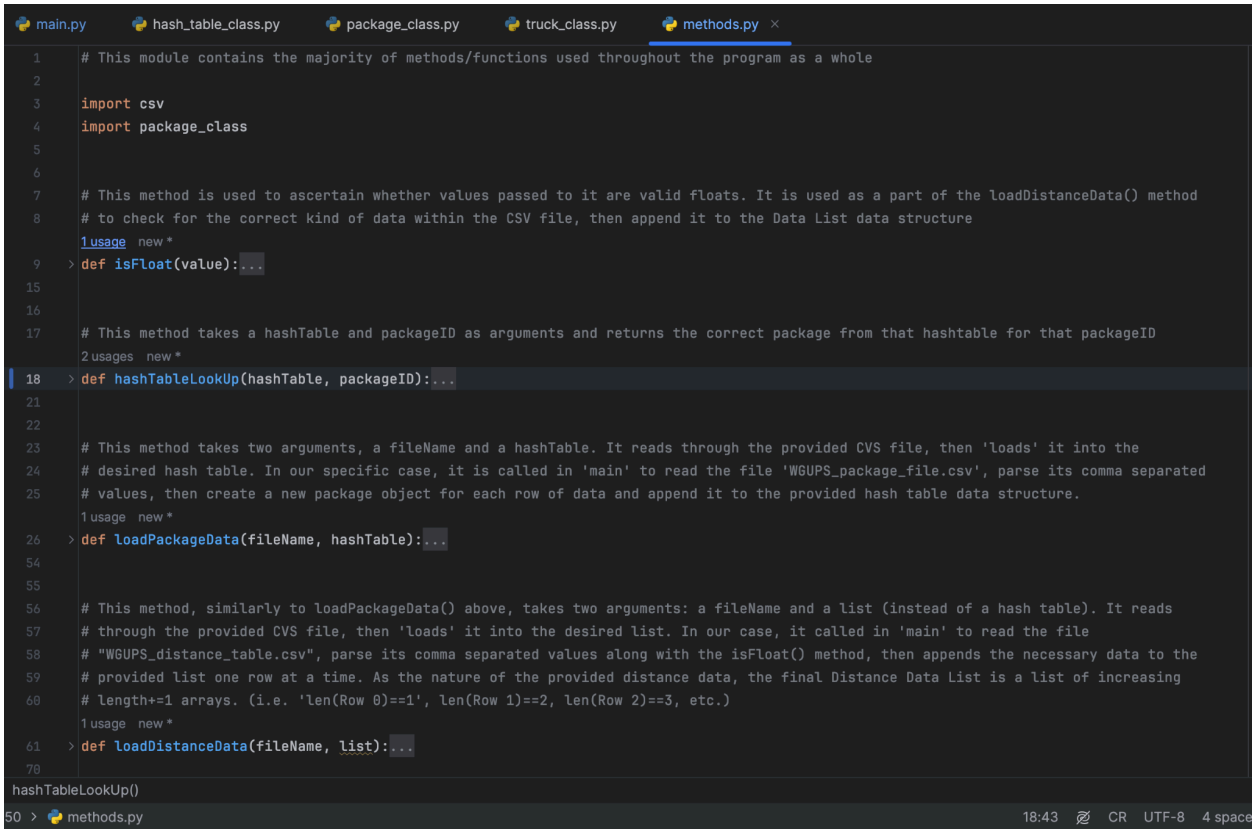
package_class.py:



truck_class.py:



methods.py:

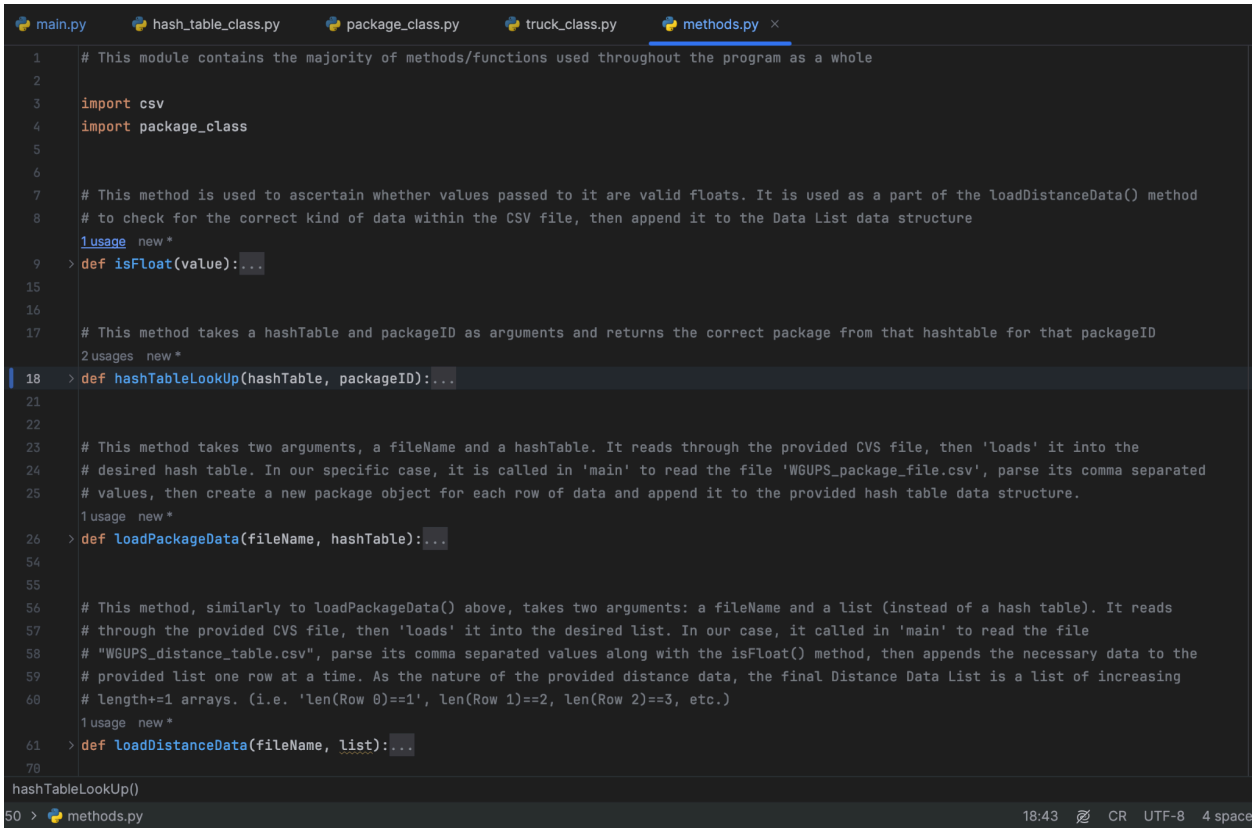


C1. Identification Information

Main.py screenshot showing StudentID:

C2. Process and Flow Comments

Below are some screenshots showing detailed comments explaining the processes and flow of the program for WGUPS:




```
# Truck 2 will prioritize LATE on arrival packages that still have an early deadline
# NOTE: IT WILL LEAVE @ 9:10am!
truck2 = truck_class.Truck(2)
# packages 3, 18, 36, 38 must all be on Truck #2
truck2.addPackage(packageHashTable.search(3))
truck2.addPackage(packageHashTable.search(18))
truck2.addPackage(packageHashTable.search(36))
truck2.addPackage(packageHashTable.search(38))
# Truck 2 PRIORITY packages:
truck2.addPackage(packageHashTable.search(6))
truck2.addPackage(packageHashTable.search(25))
truck2.addPackage(packageHashTable.search(33))
truck2.addPackage(packageHashTable.search(12))
truck2.addPackage(packageHashTable.search(23))
truck2.addPackage(packageHashTable.search(24))
truck2.addPackage(packageHashTable.search(26))
truck2.addPackage(packageHashTable.search(27))
```

```
# This method finds and returns how many miles there are between two provided addresses held within a list storing address data.
# It is subsequently used in the minDistanceFrom() method below to sequentially find which two addresses are closest to one another.
4 usages  Aaron Tandem
def distanceBetween(address1, address2):...


# This method makes use of the distanceBetween() function above to sequentially find the address of a package in a truck's list that is
# closest to the provided 'fromAddress' (i.e. 'where the truck was last located'). This 'fromAddress' generally always starts at the WGUPS
# HUB located at "4001 South 700 East, Salt Lake City, UT 84107". However, this address is automatically updated to the address of the last
# delivered package in the recursive call of minDistanceFrom() in the deliverTruckPackages() method below. This is a critical part of the
# Nearest Neighbor Greedy Algorithm used by this program to solve WGUPS's package delivery efficiency issues.
1 usage  Aaron Tandem
def minDistanceFrom(fromAddress, truckPackages):...


# This method is the final part of the Nearest Neighbor algorithm chosen for this project, and the part of the algorithm that
# simultaneously removes (i.e. "delivers") a package from a truck's package list and update's that truck's "startingAddress" as the same
# address of the package that was just removed/delivered.
# The second argument "startTime" must be of format 'time(x, x, x)', including when calling getStartingTime() or getEndingTime() from the
# provided 'truck' argument.
3 usages  Aaron Tandem
def deliverTruckPackages(truck, startTime):...
```

D. Interface

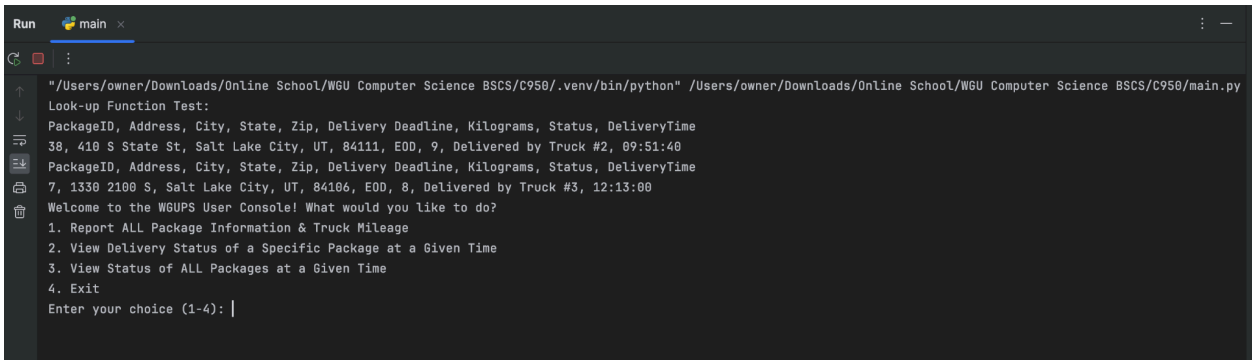
Below are screenshots showing the code and functionality of the user interface:

Code: (can be viewed in greater detail in the provided zip file)

main.py | hash_table_class.py | package_class.py | truck_class.py | methods.py

```
# This is the User Interface
# This script checks to make sure it is being run as the main program, then calls the displayMenu() method to show the user
# the options available to view certain package related information.
if __name__ == "__main__":
    while True:
        methods.displayMenu()

        choice = input("Enter your choice (1-4): ")

        # Displays the hash table data to the console by calling the hash table's search() functionality in a loop, then prints total
        # distances traveled for each truck, followed by the cumulative total distance of all three trucks combined.
        if choice == "1":

            print("PackageID, Address, City, State, Zip, Delivery Deadline, Kilograms, Status, DeliveryTime")
            # Fetch data from Hash Table and then print it
            for i in range(len(packageHashTable.table)):
                print("{}".format(packageHashTable.search(i + 1)))
            print("Distance Traveled for Truck #1: " + str(round(truck1.getTotalDeliveryDistance(), 1)) + " miles.")
            print("Distance Traveled for Truck #2: " + str(round(truck2.getTotalDeliveryDistance(), 1)) + " miles.")
            print("Distance Traveled for Truck #3: " + str(round(truck3.getTotalDeliveryDistance(), 1)) + " miles.")
            print("Cumulative Distance for All Trucks: " +
                  str(round(truck1.getTotalDeliveryDistance(), 1) +
                      round(truck2.getTotalDeliveryDistance(), 1) +
                      round(truck3.getTotalDeliveryDistance(), 1)
                      ) + " miles."
                  )

        # Asks the user for a PackageID first, followed by a time, then returns the delivery status of a package that matches the provided
        # PackageID (if it exists), as it relates to the time provided.
        elif choice == "2":
            inputID = int(input("Enter packageID (1-40): "))
```

CLI interface:



D1. First Status Check

Screenshot of the first status check at 9:00am:

```
Delivery Status for All Packages as of 09:00:00:
Package 1 was Delivered by Truck #1 at 08:39:00 to 195 W Oakland Ave.
Package 2 is en route as of 09:00:00.
Package 3 is en route as of 09:00:00.
Package 4 is en route as of 09:00:00.
Package 5 is en route as of 09:00:00.
Package 6 is en route as of 09:00:00.
Package 7 is en route as of 09:00:00.
Package 8 is en route as of 09:00:00.
Package 9 is en route as of 09:00:00.
Package 10 is en route as of 09:00:00.
Package 11 is en route as of 09:00:00.
Package 12 is en route as of 09:00:00.
Package 13 is en route as of 09:00:00.
Package 14 was Delivered by Truck #1 at 08:06:20 to 4300 S 1300 E.
Package 15 was Delivered by Truck #1 at 08:13:00 to 4580 S 2300 E.
Package 16 was Delivered by Truck #1 at 08:13:00 to 4580 S 2300 E.
Package 17 is en route as of 09:00:00.
Package 18 is en route as of 09:00:00.
Package 19 is en route as of 09:00:00.
Package 20 was Delivered by Truck #1 at 08:48:00 to 3595 Main St.
Package 21 is en route as of 09:00:00.
Package 22 is en route as of 09:00:00.
Package 23 is en route as of 09:00:00.
Package 24 is en route as of 09:00:00.
Package 25 is en route as of 09:00:00.
Package 26 is en route as of 09:00:00.
Package 27 is en route as of 09:00:00.
Package 28 is en route as of 09:00:00.
Package 29 was Delivered by Truck #1 at 08:29:40 to 1330 2100 S.
```

```
Package 30 is en route as of 09:00:00.
Package 31 was Delivered by Truck #1 at 08:53:20 to 3365 S 900 W.
Package 32 is en route as of 09:00:00.
Package 33 is en route as of 09:00:00.
Package 34 was Delivered by Truck #1 at 08:13:00 to 4580 S 2300 E.
Package 35 is en route as of 09:00:00.
Package 36 is en route as of 09:00:00.
Package 37 is en route as of 09:00:00.
Package 38 is en route as of 09:00:00.
Package 39 is en route as of 09:00:00.
Package 40 was Delivered by Truck #1 at 08:42:40 to 380 W 2880 S.
```

D2. Second Status Check

Screenshot of the second status check at 10:05am:

```
Delivery Status for All Packages as of 10:05:00:
Package 1 was Delivered by Truck #1 at 08:39:00 to 195 W Oakland Ave.
Package 2 is en route as of 10:05:00.
Package 3 was Delivered by Truck #2 at 09:55:00 to 233 Canyon Rd.
Package 4 is en route as of 10:05:00.
Package 5 is en route as of 10:05:00.
Package 6 is en route as of 10:05:00.
Package 7 is en route as of 10:05:00.
Package 8 is en route as of 10:05:00.
Package 9 is en route as of 10:05:00.
Package 10 is en route as of 10:05:00.
Package 11 is en route as of 10:05:00.
Package 12 is en route as of 10:05:00.
Package 13 was Delivered by Truck #1 at 09:12:40 to 2010 W 500 S.
Package 14 was Delivered by Truck #1 at 08:06:20 to 4300 S 1300 E.
Package 15 was Delivered by Truck #1 at 08:13:00 to 4580 S 2300 E.
Package 16 was Delivered by Truck #1 at 08:13:00 to 4580 S 2300 E.
Package 17 is en route as of 10:05:00.
Package 18 is en route as of 10:05:00.
Package 19 is en route as of 10:05:00.
Package 20 was Delivered by Truck #1 at 08:48:00 to 3595 Main St.
Package 21 is en route as of 10:05:00.
Package 22 is en route as of 10:05:00.
Package 23 is en route as of 10:05:00.
Package 24 was Delivered by Truck #2 at 09:23:40 to 5025 State St.
Package 25 was Delivered by Truck #2 at 09:18:00 to 5383 South 900 East #104.
Package 26 was Delivered by Truck #2 at 09:18:00 to 5383 South 900 East #104.
Package 27 is en route as of 10:05:00.
Package 28 is en route as of 10:05:00.
Package 29 was Delivered by Truck #1 at 08:29:40 to 1330 2100 S.
main.py
```

D3. Third Status Check

Screenshot of the third status check at 1:00pm:

```
Delivery Status for All Packages as of 13:00:00:
Package 1 was Delivered by Truck #1 at 08:39:00 to 195 W Oakland Ave.
Package 2 was Delivered by Truck #3 at 12:07:40 to 2530 S 500 E.
Package 3 was Delivered by Truck #2 at 09:55:00 to 233 Canyon Rd.
Package 4 was Delivered by Truck #3 at 11:47:40 to 380 W 2880 S.
Package 5 was Delivered by Truck #3 at 12:28:20 to 410 S State St.
Package 6 was Delivered by Truck #2 at 10:25:40 to 3060 Lester St.
Package 7 was Delivered by Truck #3 at 12:13:00 to 1330 2100 S.
Package 8 was Delivered by Truck #3 at 12:31:40 to 300 State St.
Package 9 was Delivered by Truck #3 at 12:28:20 to 410 S State St.
Package 10 was Delivered by Truck #3 at 12:22:20 to 600 E 900 South.
Package 11 is en route as of 13:00:00.
Package 12 was Delivered by Truck #2 at 11:05:40 to 3575 W Valley Central Station bus Loop.
Package 13 was Delivered by Truck #1 at 09:12:40 to 2010 W 500 S.
Package 14 was Delivered by Truck #1 at 08:06:20 to 4300 S 1300 E.
Package 15 was Delivered by Truck #1 at 08:13:00 to 4580 S 2300 E.
Package 16 was Delivered by Truck #1 at 08:13:00 to 4580 S 2300 E.
Package 17 was Delivered by Truck #3 at 11:55:20 to 3148 S 1100 W.
Package 18 was Delivered by Truck #2 at 10:38:40 to 1488 4800 S.
Package 19 was Delivered by Truck #3 at 11:39:20 to 177 W Price Ave.
Package 20 was Delivered by Truck #1 at 08:48:00 to 3595 Main St.
Package 21 was Delivered by Truck #3 at 11:37:40 to 3595 Main St.
Package 22 is en route as of 13:00:00.
Package 23 was Delivered by Truck #2 at 10:40:40 to 5100 South 2700 West.
Package 24 was Delivered by Truck #2 at 09:23:40 to 5025 State St.
Package 25 was Delivered by Truck #2 at 09:18:00 to 5383 South 900 East #104.
Package 26 was Delivered by Truck #2 at 09:18:00 to 5383 South 900 East #104.
Package 27 was Delivered by Truck #2 at 10:11:00 to 1060 Dalton Ave S.
Package 28 was Delivered by Truck #3 at 11:44:20 to 2835 Main St.
Package 29 was Delivered by Truck #1 at 08:29:40 to 1330 2100 S.
main.py
```

```
Package 30 was Delivered by Truck #1 at 09:26:40 to 300 State St.
Package 31 was Delivered by Truck #1 at 08:53:20 to 3365 S 900 W.
Package 32 was Delivered by Truck #3 at 11:53:20 to 3365 S 900 W.
Package 33 was Delivered by Truck #2 at 09:38:00 to 2530 S 500 E.
Package 34 was Delivered by Truck #1 at 08:13:00 to 4580 S 2300 E.
Package 35 was Delivered by Truck #3 at 12:51:00 to 1060 Dalton Ave S.
Package 36 was Delivered by Truck #2 at 10:20:20 to 2300 Parkway Blvd.
Package 37 was Delivered by Truck #1 at 09:23:20 to 410 S State St.
Package 38 was Delivered by Truck #2 at 09:51:40 to 410 S State St.
Package 39 was Delivered by Truck #3 at 12:45:40 to 2010 W 500 S.
Package 40 was Delivered by Truck #1 at 08:42:40 to 380 W 2880 S.
```

E. Screenshot of Code Execution

Below is a screenshot showing the successful completion of the code, including the total mileage traveled by all trucks:

```
PackageID, Address, City, State, Zip, Delivery Deadline, Kilograms, Status, DeliveryTime
1, 195 W Oakland Ave, Salt Lake City, UT, 84115, 10:30 AM, 21, Delivered by Truck #1, 08:39:00
2, 2530 S 500 E, Salt Lake City, UT, 84106, EOD, 44, Delivered by Truck #3, 12:07:40
3, 233 Canyon Rd, Salt Lake City, UT, 84103, EOD, 2, Delivered by Truck #2, 09:55:00
4, 380 W 2880 S, Salt Lake City, UT, 84115, EOD, 4, Delivered by Truck #3, 11:47:40
5, 410 S State St, Salt Lake City, UT, 84111, EOD, 5, Delivered by Truck #3, 12:28:20
6, 3060 Lester St, West Valley City, UT, 84119, 10:30 AM, 88, Delivered by Truck #2, 10:25:40
7, 1330 2100 S, Salt Lake City, UT, 84106, EOD, 8, Delivered by Truck #3, 12:13:00
8, 300 State St, Salt Lake City, UT, 84103, EOD, 9, Delivered by Truck #3, 12:31:40
9, 410 S State St, Salt Lake City, UT, 84103, EOD, 2, Delivered by Truck #3, 12:28:20
10, 600 E 900 South, Salt Lake City, UT, 84105, EOD, 1, Delivered by Truck #3, 12:22:20
11, 2600 Taylorsville Blvd, Salt Lake City, UT, 84118, EOD, 1, Delivered by Truck #3, 13:15:20
12, 3575 W Valley Central Station bus Loop, West Valley City, UT, 84119, EOD, 1, Delivered by Truck #2, 11:05:40
13, 2010 W 500 S, Salt Lake City, UT, 84104, 10:30 AM, 2, Delivered by Truck #1, 09:12:40
14, 4300 S 1300 E, Millcreek, UT, 84117, 10:30 AM, 88, Delivered by Truck #1, 08:06:20
15, 4580 S 2300 E, Holladay, UT, 84117, 9:00 AM, 4, Delivered by Truck #1, 08:13:00
16, 4580 S 2300 E, Holladay, UT, 84117, 10:30 AM, 88, Delivered by Truck #1, 08:13:00
17, 3148 S 1100 W, Salt Lake City, UT, 84119, EOD, 2, Delivered by Truck #3, 11:55:20
18, 1488 4800 S, Salt Lake City, UT, 84123, EOD, 6, Delivered by Truck #2, 10:38:40
19, 177 W Price Ave, Salt Lake City, UT, 84115, EOD, 37, Delivered by Truck #3, 11:39:20
20, 3595 Main St, Salt Lake City, UT, 84115, 10:30 AM, 37, Delivered by Truck #1, 08:48:00
21, 3595 Main St, Salt Lake City, UT, 84115, EOD, 3, Delivered by Truck #3, 11:37:40
22, 6351 South 900 East, Murray, UT, 84121, EOD, 2, Delivered by Truck #3, 13:38:00
23, 5100 South 2700 West, Salt Lake City, UT, 84118, EOD, 5, Delivered by Truck #2, 10:40:40
24, 5025 State St, Murray, UT, 84107, EOD, 7, Delivered by Truck #2, 09:23:40
25, 5383 South 900 East #104, Salt Lake City, UT, 84117, 10:30 AM, 7, Delivered by Truck #2, 09:18:00
26, 5383 South 900 East #104, Salt Lake City, UT, 84117, EOD, 25, Delivered by Truck #2, 09:18:00
27, 1060 Dalton Ave S, Salt Lake City, UT, 84104, EOD, 5, Delivered by Truck #2, 10:11:00
28, 2835 Main St, Salt Lake City, UT, 84115, EOD, 7, Delivered by Truck #3, 11:44:20
29, 1330 2100 S, Salt Lake City, UT, 84106, 10:30 AM, 2, Delivered by Truck #1, 08:29:40
```

```
30, 300 State St, Salt Lake City, UT, 84103, 10:30 AM, 1, Delivered by Truck #1, 09:26:40
31, 3365 S 900 W, Salt Lake City, UT, 84119, 10:30 AM, 1, Delivered by Truck #1, 08:53:20
32, 3365 S 900 W, Salt Lake City, UT, 84119, EOD, 1, Delivered by Truck #3, 11:53:20
33, 2530 S 500 E, Salt Lake City, UT, 84106, EOD, 1, Delivered by Truck #2, 09:38:00
34, 4580 S 2300 E, Holladay, UT, 84117, 10:30 AM, 2, Delivered by Truck #1, 08:13:00
35, 1060 Dalton Ave S, Salt Lake City, UT, 84104, EOD, 88, Delivered by Truck #3, 12:51:00
36, 2300 Parkway Blvd, West Valley City, UT, 84119, EOD, 88, Delivered by Truck #2, 10:20:20
37, 410 S State St, Salt Lake City, UT, 84111, 10:30 AM, 2, Delivered by Truck #1, 09:23:20
38, 410 S State St, Salt Lake City, UT, 84111, EOD, 9, Delivered by Truck #2, 09:51:40
39, 2010 W 500 S, Salt Lake City, UT, 84104, EOD, 9, Delivered by Truck #3, 12:45:40
40, 380 W 2880 S, Salt Lake City, UT, 84115, 10:30 AM, 45, Delivered by Truck #1, 08:42:40
Distance Traveled for Truck #1: 33.6 miles.
Distance Traveled for Truck #2: 42.3 miles.
Distance Traveled for Truck #3: 41.7 miles.
Cumulative Distance for All Trucks: 117.60000000000001 miles.
```

F1. Strengths of the Chosen Algorithm

The algorithm chosen for this project was the Nearest Neighbor Algorithm. In simple terms, this algorithm is greedy in nature, meaning that it chooses the option that is "best now" (i.e. "in the short-term") to achieve an optimal, if not perfectly ideal, solution. This algorithm's strengths include:

- Being self-adjusting in that its implementation is not dependent on a specific data set, but can account for increasing, decreasing, or completely different data points
- A time-complexity of only O(N^2) when implemented correctly
- It is relatively easy to implement, and does not require extensively detailed knowledge in the field of computer science or heuristics

F2. Verification of Algorithm

The implementation of the Nearest Neighbor Algorithm in this project was able to keep the total cumulative miles for all three trucks less than 120 miles. Additionally, all packages were delivered on time as required. With further optimization, even less mileage, with potential for quicker delivery times, could be achieved.

F3. Other Possible Algorithms

Two other algorithms that could potentially replace the Nearest Neighbor implementation in this project are the 2-Opt Algorithm and Dijkstra's Shortest Path. Both of these algorithms are concerned with finding an ideal path between points of "travel" by reducing total length of said travel.

F3a. Algorithm Differences

In simple terms, 2-Opt is similar to the Nearest Neighbor (and may even implement the Nearest Neighbor algorithm as part of its implementation), but differs in how it gets its name — by iteratively "swapping" pairs of edges in a route and seeing if that improves travel time and distance. Dijkstra's Shortest Path on the other hand, systematically considers the entirety of paths between two points and chooses the shortest one, while Nearest Neighbor only makes local decisions based on neighboring nodes.

G. Different Approach

If I were to do this project again, one of the main things I would do is to create an algorithm that automatically loads the trucks in an optimal manner. This could be done relatively simply by implementing a version of the Nearest Neighbor algorithm that chooses which packages go best with others based on their delivery addresses. Next, from an organizational perspective, I would like to ensure that *all* methods not belonging to specific classes are kept in a specific file so that code neatness can be maintained to the utmost within the main program itself.

H. Verification of Data Structure

The hash table satisfies all requirements for this project and helps improve WGUPS's systems in a number of ways. Mainly, it is a fast and self-adjusting data structure. Meaning, all operations directly related to updating, inserting, removing, and searching with the hash table operate in O(N) time. Additionally, the hash table can easily and automatically be increased or decreased in size in relation to changing data sets without sacrificing any of this speed.

H1. Other Data Structures

Two other data structures that could meet the same requirements for this scenario are lists and arrays. Both of these data structures are not inherently self-adjusting, but can be made to be so with extra steps during implementation.

H1a. Data Structure Differences

Lists differ from hash tables in that this data structure is linear in nature, meaning that all data is stored sequentially within them. Additionally, values are generally accessed via indices rather than via key-value mapping like in hash tables. Arrays on the other hand have a tendency of being fixed in size and only storing one data type. However, because we usually will know how many packages we'll be delivering, along with the fact that the array would only store package objects, it's possible it could be used. However, it is definitely not the ideal data structure to use as a solution to this scenario.

I. Sources

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

Retrieved March 22, 2021, from

https://learn.zybooks.com/zybook/WGUC950AY20182019/