

C950 Task-2 WGUPS Write-Up

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)

Trayvonious Pendleton

ID #011205284

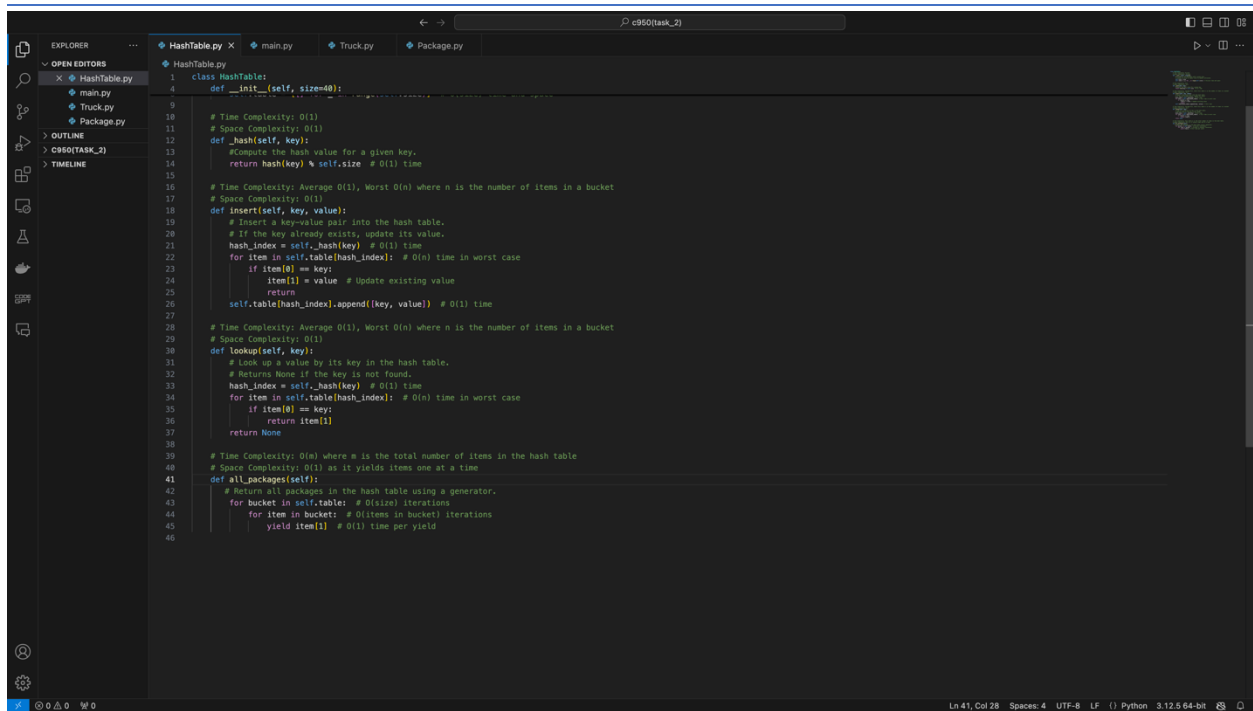
WGU Email: tpend32@wgu.edu

9/24/2024

C950 Data Structures and Algorithms II

C950 Task-2 WGUPS Write-Up

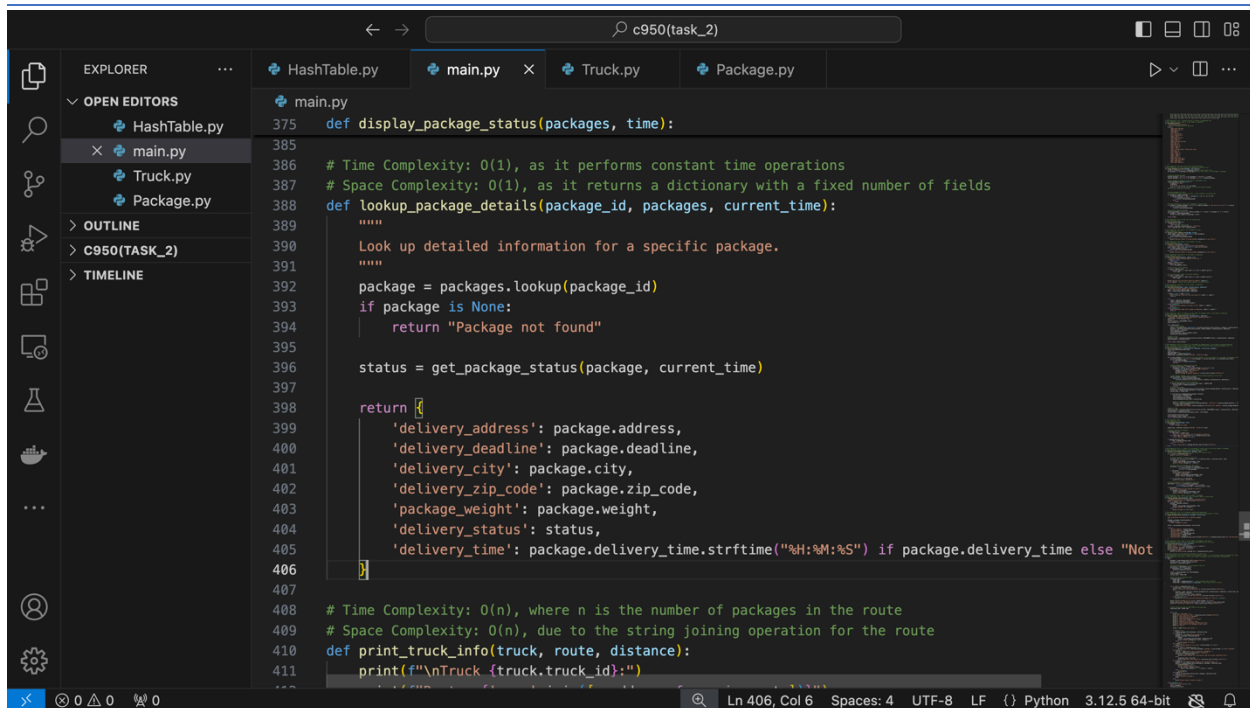
A. Hash Table



```
1 class HashTable:
2     def __init__(self, size=40):
3         self.table = [None] * size
4
5     # Time Complexity: O(1)
6     # Space Complexity: O(1)
7
8     def _hash(self, key):
9         # Compute the hash value for a given key.
10        return hash(key) % self.size # O(1) time
11
12    # Time Complexity: Average O(1), Worst O(n) where n is the number of items in a bucket
13    # Space Complexity: O(1)
14
15    def insert(self, key, value):
16        # Insert a key-value pair into the hash table.
17        # If the key already exists, update its value.
18        hash_index = self._hash(key) # O(1) time
19        for item in self.table[hash_index]: # O(n) time in worst case
20            if item[0] == key:
21                item[1] = value # Update existing value
22                return
23        self.table[hash_index].append([key, value]) # O(1) time
24
25    # Time Complexity: Average O(1), Worst O(n) where n is the number of items in a bucket
26    # Space Complexity: O(1)
27
28    def lookup(self, key):
29        # Look up a value by its key in the hash table.
30        # Returns None if the key is not found.
31        hash_index = self._hash(key) # O(1) time
32        for item in self.table[hash_index]: # O(n) time in worst case
33            if item[0] == key:
34                return item[1]
35        return None
36
37    # Time Complexity: O(m) where m is the total number of items in the hash table
38    # Space Complexity: O(1) as it yields items one at a time
39
40    def all_packages(self):
41        # Return all packages in the hash table using a generator.
42        for bucket in self.table: # O(size) iterations
43            for item in bucket: # O(items in bucket) iterations
44                yield item[1] # O(1) time per yield
45
46
```

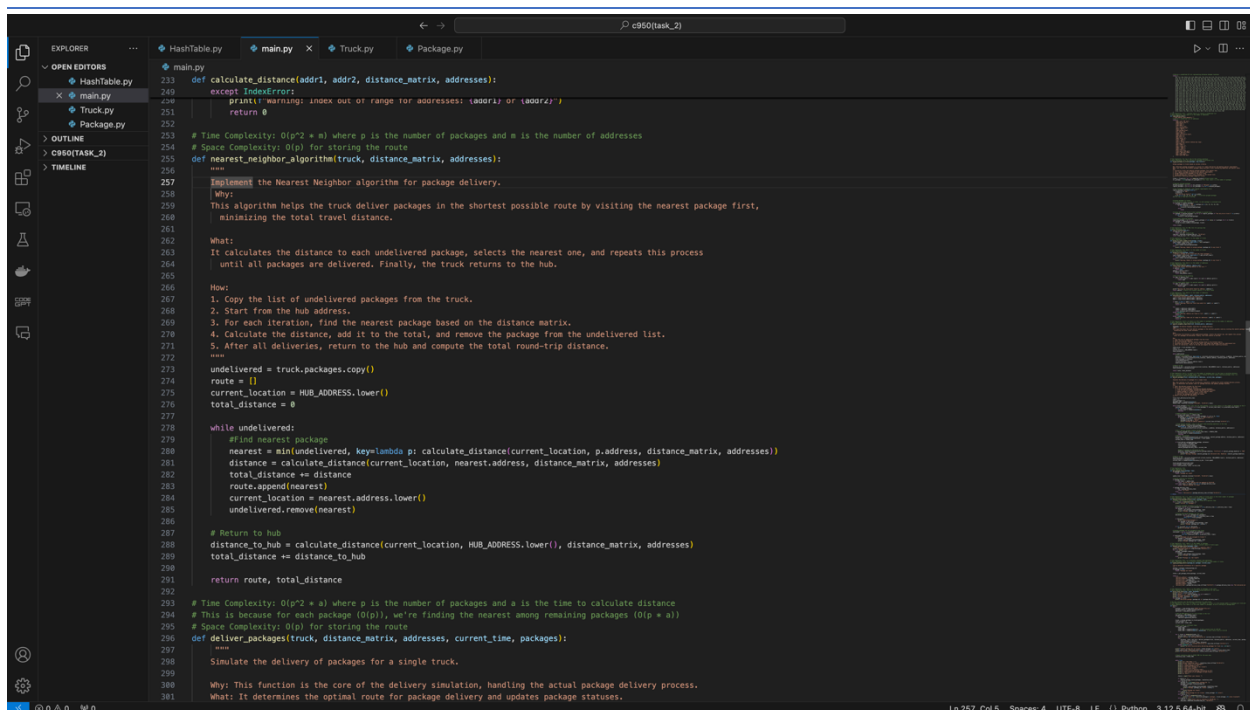
C950 Task-2 WGUPS Write-Up

B. Look-Up Functions



```
def display_package_status(packages, time):  
    # Time Complexity: O(1), as it performs constant time operations  
    # Space Complexity: O(1), as it returns a dictionary with a fixed number of fields  
    def lookup_package_details(package_id, packages, current_time):  
        """  
        Look up detailed information for a specific package.  
        """  
        package = packages.lookup(package_id)  
        if package is None:  
            return "Package not found"  
        status = get_package_status(package, current_time)  
        return {  
            'delivery_address': package.address,  
            'delivery_deadline': package.deadline,  
            'delivery_city': package.city,  
            'delivery_zip_code': package.zip_code,  
            'package_weight': package.weight,  
            'delivery_status': status,  
            'delivery_time': package.delivery_time.strftime("%H:%M:%S") if package.delivery_time else "Not"  
        }  
    # Time Complexity: O(n), where n is the number of packages in the route  
    # Space Complexity: O(n), due to the string joining operation for the route  
    def print_truck_info(truck, route, distance):  
        print(f"\nTruck {truck.truck_id}:")
```

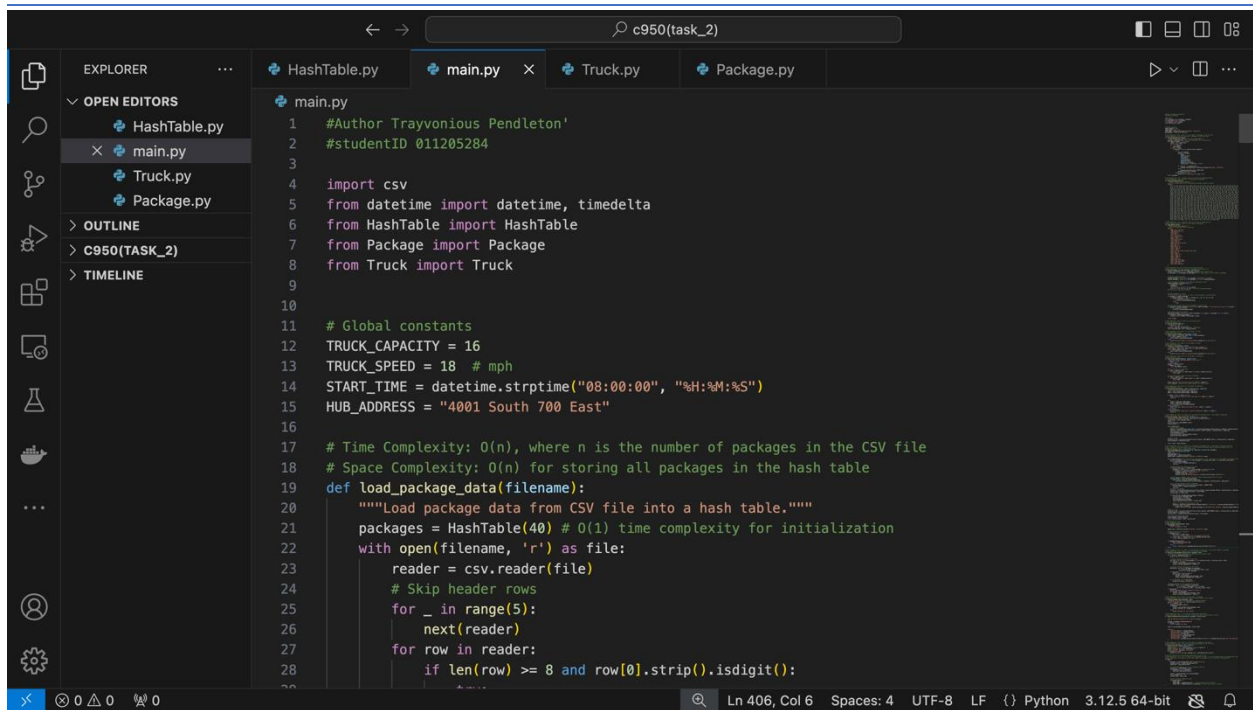
C. Original Code



```
def calculate_distance(addr1, addr2, distance_matrix, addresses):  
    except IndexError:  
        print(f"Warning: Index out of range for addresses: {addr1} or {addr2}")  
        return 0  
    # Time Complexity: O(p^2 * a) where p is the number of packages and a is the number of addresses  
    # Space Complexity: O(p) for storing the route  
    def nearest_neighbor_algorithm(truck, distance_matrix, addresses):  
        """  
        Implement the Nearest Neighbor algorithm for package delivery.  
        Why:  
        This algorithm helps the truck deliver packages in the shortest possible route by visiting the nearest package first,  
        minimizing the total travel distance.  
        What:  
        It calculates the distance to each undelivered package, selects the nearest one, and repeats this process  
        until all packages are delivered. Finally, the truck returns to the hub.  
        How:  
        1. Copy the list of undelivered packages from the truck.  
        2. Start from the hub address.  
        3. For each iteration, find the nearest package based on the distance matrix.  
        4. Calculate the distance, add it to the total, and remove the package from the undelivered list.  
        5. After all deliveries, return to the hub and compute the total round-trip distance.  
        """  
        undelivered = truck.packages.copy()  
        route = []  
        current_location = HUB_ADDRESS.lower()  
        total_distance = 0  
        while undelivered:  
            # Find nearest package  
            nearest = min(undelivered, key=lambda p: calculate_distance(current_location, p.address, distance_matrix, addresses))  
            distance = calculate_distance(current_location, nearest.address, distance_matrix, addresses)  
            total_distance += distance  
            route.append(nearest)  
            current_location = nearest.address.lower()  
            undelivered.remove(nearest)  
        # Return to hub  
        distance_to_hub = calculate_distance(current_location, HUB_ADDRESS.lower(), distance_matrix, addresses)  
        total_distance += distance_to_hub  
        return route, total_distance  
    # Time Complexity: O(p^2 * a) where p is the number of packages and a is the time to calculate distance  
    # This is because for each package (O(p)), we're finding the nearest among remaining packages (O(p * a))  
    # Space Complexity: O(p) for storing the route  
    def deliver_packages(truck, distance_matrix, addresses, current_time, packages):  
        """  
        Simulate the delivery of packages for a single truck.  
        Why: This function is the core of the delivery simulation, handling the actual package delivery process.  
        What: It determines the optimal route for package delivery and updates package statuses.
```

C950 Task-2 WGUPS Write-Up

C1. Identification Information



```
1 #Author Trayvonious Pendleton'
2 #studentID 011205284
3
4 import csv
5 from datetime import datetime, timedelta
6 from HashTable import HashTable
7 from Package import Package
8 from Truck import Truck
9
10
11 # Global constants
12 TRUCK_CAPACITY = 16
13 TRUCK_SPEED = 18 # mph
14 START_TIME = datetime.strptime("08:00:00", "%H:%M:%S")
15 HUB_ADDRESS = "4001 South 700 East"
16
17 # Time Complexity: O(n), where n is the number of packages in the CSV file
18 # Space Complexity: O(n) for storing all packages in the hash table
19
20 def load_package_data(filename):
21     """Load package data from CSV file into a hash table."""
22     packages = HashTable(40) # O(1) time complexity for initialization
23     with open(filename, 'r') as file:
24         reader = csv.reader(file)
25         # Skip header rows
26         for _ in range(5):
27             next(reader)
28         for row in reader:
29             if len(row) >= 8 and row[0].strip().isdigit():
30                 # ...
```

C2. Process and Flow Comments

C950 Task-2 WGUPS Write-Up

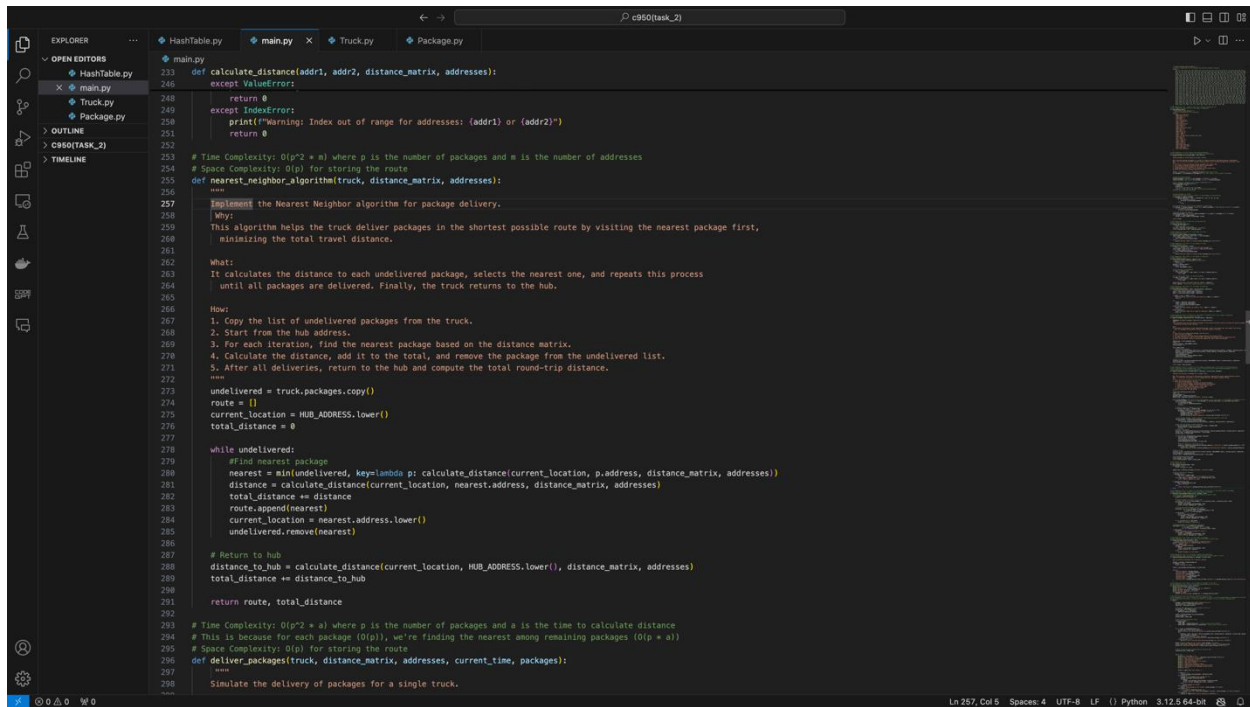
```
EXPLORER
  HashTable.py
  main.py
  Truck.py
  Package.py

main.py
11 TRUCK_CAPACITY = 16
12 TRUCK_SPEED = 18 # mph
13 START_TIME = datetime.strptime("08:00:00", "%H:%M:%S")
14 HUB_ADDRESS = "4001 South 7th East"
15
16
17 # Time Complexity: O(n), where n is the number of packages in the CSV file
18 # Space Complexity: O(n) for storing all packages in the hash table
19 def load_package_data(filename):
20     """Load package data from CSV file into a hash table.
21
22     Why: Efficient data storage and retrieval are crucial for the package delivery system.
23     What: This function reads package data from a CSV file and stores it in a hash table.
24     How:
25     1. Initialize a HashTable with a capacity of 40.
26     2. Open and read the CSV file, skipping header rows.
27     3. For each valid row, create a Package object and insert it into the hash table.
28     4. Handle special cases like delayed packages by setting their available time.
29
30     """
31     packages = HashTable(40) # O(1) time complexity for initialization
32     with open(filename, 'r') as file:
33         reader = csv.reader(file)
34         # Skip header rows
35         for _ in range(5):
36             next(reader)
37         for row in reader:
38             if len(row) >= 8 and row[0].strip().isdigit():
39                 try:
40                     id = int(row[0])
41                     package = Package(
42                         id=id,
43                         address=row[1],
44                         city=row[2],
45                         state=row[3],
46                         zip_code=row[4],
47                         deadline=row[5],
48                         weight=float(row[6]),
49                         notes=row[7] if len(row) > 7 else ''
50                     )
51                     if "Delayed" in package.notes:
52                         package.available_time = datetime.strptime("09:05:00", "%H:%M:%S")
53                     else:
54                         package.available_time = START_TIME
55                     packages.insert(id, package)
56                 except ValueError as e:
57                     print(f"Error processing row (row): {e}")
58         return packages
59
60 # Time Complexity: O(1) - constant time as it returns a predefined matrix
61 # Space Complexity: O(n^2) where n is the number of addresses
62 def create_distance_matrix():
63     """Create distance matrix manually.
64     Returns a predefined 20 list representing distances between locations
65     """
66     return [
67         0, 7.2, 3.8, 11.0, 2.2, 3.5, 10.9, 8.6, 7.6, 2.8, 6.4, 3.2, 7.6, 5.2, 4.4, 3.7, 7.6, 2.0, 3.6, 6.5, 1.9, 3.4, 2.4, 6.4, 2.4, 5.0, 3.6],
68         [7.2, 0, 4.5, 10.5, 5.0, 1.5, 12.5, 9.4, 8.4, 5.4, 2.4, 4.4, 1.4, 2.4, 1.4, 0.5, 1.4, 0.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4],
69         [3.8, 4.5, 0, 7.0, 3.0, 0.5, 7.5, 4.4, 3.4, 0.4, 1.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4],
70         [11.0, 10.5, 7.0, 0, 4.0, 6.5, 13.5, 10.4, 9.4, 6.4, 3.4, 2.4, 1.4, 1.4, 1.4, 0.4, 1.4, 0.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4],
71         [2.2, 1.5, 3.0, 4.0, 0, 0.5, 9.5, 6.4, 5.4, 2.4, 1.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4],
72         [3.5, 1.5, 0.5, 6.5, 0.5, 0, 11.5, 8.4, 7.4, 4.4, 3.4, 2.4, 1.4, 1.4, 1.4, 0.4, 1.4, 0.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4],
73         [10.9, 12.5, 7.5, 13.5, 9.5, 11.5, 0, 7.4, 6.4, 3.4, 2.4, 1.4, 1.4, 1.4, 1.4, 0.4, 1.4, 0.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4],
74         [8.6, 9.4, 4.4, 10.4, 6.4, 8.4, 7.4, 0, 1.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4],
75         [7.6, 8.4, 3.4, 9.4, 5.4, 7.4, 6.4, 1.4, 0, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4],
76         [2.8, 5.4, 0.4, 6.4, 2.4, 4.4, 3.4, 0.4, 0.4, 0, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4],
77         [6.4, 2.4, 1.4, 3.4, 1.4, 2.4, 1.4, 0.4, 0.4, 0.4, 0, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4],
78         [3.2, 4.4, 0.4, 2.4, 0.4, 1.4, 1.4, 0.4, 0.4, 0.4, 0.4, 0, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4],
79         [7.6, 1.4, 0.4, 1.4, 0.4, 1.4, 1.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4],
80         [5.2, 1.4, 0.4, 1.4, 0.4, 1.4, 1.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4],
81         [4.4, 1.4, 0.4, 1.4, 0.4, 1.4, 1.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4],
82         [3.7, 0.5, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4],
83         [7.6, 1.4, 0.4, 1.4, 0.4, 1.4, 1.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4],
84         [2.0, 0.4, 0.4, 1.4, 0.4, 1.4, 1.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4],
85         [3.6, 1.4, 0.4, 1.4, 0.4, 1.4, 1.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4],
86         [6.5, 1.4, 0.4, 1.4, 0.4, 1.4, 1.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4],
87         [1.9, 1.4, 0.4, 1.4, 0.4, 1.4, 1.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0, 0.4, 0.4, 0.4, 0.4, 0.4],
88         [3.4, 1.4, 0.4, 1.4, 0.4, 1.4, 1.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0, 0.4, 0.4, 0.4, 0.4],
89         [2.4, 1.4, 0.4, 1.4, 0.4, 1.4, 1.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0, 0.4, 0.4, 0.4],
90         [6.4, 1.4, 0.4, 1.4, 0.4, 1.4, 1.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0, 0.4, 0.4],
91         [2.4, 1.4, 0.4, 1.4, 0.4, 1.4, 1.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0, 0.4],
92         [5.0, 1.4, 0.4, 1.4, 0.4, 1.4, 1.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0],
93         [3.6, 1.4, 0.4, 1.4, 0.4, 1.4, 1.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4]
94     ]
95
96 # Time Complexity: O(n^2)
97 # Space Complexity: O(n^2)
98 def create_package_list():
99     """Create package list manually.
100     Returns a predefined list of packages
101     """
102     return [
103         Package(1, "4001 South 7th East", "Salt Lake City", "UT", "84143", 1.0, "Delayed", START_TIME + timedelta(minutes=5)),
104         Package(2, "1000 North 200 East", "Salt Lake City", "UT", "84143", 2.0, "", START_TIME),
105         Package(3, "2000 North 300 East", "Salt Lake City", "UT", "84143", 3.0, "", START_TIME),
106         Package(4, "3000 North 400 East", "Salt Lake City", "UT", "84143", 4.0, "", START_TIME),
107         Package(5, "4000 North 500 East", "Salt Lake City", "UT", "84143", 5.0, "", START_TIME),
108         Package(6, "5000 North 600 East", "Salt Lake City", "UT", "84143", 6.0, "", START_TIME),
109         Package(7, "6000 North 700 East", "Salt Lake City", "UT", "84143", 7.0, "", START_TIME),
110         Package(8, "7000 North 800 East", "Salt Lake City", "UT", "84143", 8.0, "", START_TIME),
111         Package(9, "8000 North 900 East", "Salt Lake City", "UT", "84143", 9.0, "", START_TIME),
112         Package(10, "9000 North 1000 East", "Salt Lake City", "UT", "84143", 10.0, "", START_TIME),
113         Package(11, "10000 North 1100 East", "Salt Lake City", "UT", "84143", 11.0, "", START_TIME),
114         Package(12, "11000 North 1200 East", "Salt Lake City", "UT", "84143", 12.0, "", START_TIME),
115         Package(13, "12000 North 1300 East", "Salt Lake City", "UT", "84143", 13.0, "", START_TIME),
116         Package(14, "13000 North 1400 East", "Salt Lake City", "UT", "84143", 14.0, "", START_TIME),
117         Package(15, "14000 North 1500 East", "Salt Lake City", "UT", "84143", 15.0, "", START_TIME),
118         Package(16, "15000 North 1600 East", "Salt Lake City", "UT", "84143", 16.0, "", START_TIME),
119         Package(17, "16000 North 1700 East", "Salt Lake City", "UT", "84143", 17.0, "", START_TIME),
120         Package(18, "17000 North 1800 East", "Salt Lake City", "UT", "84143", 18.0, "", START_TIME),
121         Package(19, "18000 North 1900 East", "Salt Lake City", "UT", "84143", 19.0, "", START_TIME),
122         Package(20, "19000 North 2000 East", "Salt Lake City", "UT", "84143", 20.0, "", START_TIME),
123     ]
124
125 # Time Complexity: O(n^2)
126 # Space Complexity: O(n^2)
127 def create_truck_list():
128     """Create truck list manually.
129     Returns a predefined list of trucks
130     """
131     return [
132         Truck(1, "Truck 1", TRUCK_CAPACITY, TRUCK_SPEED, START_TIME),
133         Truck(2, "Truck 2", TRUCK_CAPACITY, TRUCK_SPEED, START_TIME),
134     ]
135
136 # Time Complexity: O(n^2)
137 # Space Complexity: O(n^2)
138 def assign_packages_to_trucks(packages, trucks):
139     """Assign packages to trucks based on various criteria.
140
141     Why: Efficient package assignment is crucial for timely deliveries and meeting special requirements.
142     What: This function distributes packages among available trucks considering deadlines and special notes.
143     How:
144     1. Initialize trucks and separate delayed packages from regular ones.
145     2. Sort regular packages by deadline and special requirements.
146     3. Assign high-priority packages to the first truck.
147     4. Assign delayed and "truck 2 only" packages to the second truck.
148     5. Distribute remaining packages among all trucks.
149
150     """
151     trucks = [truck(i+1) for i in range(num_trucks)] # O(num_trucks) time
152     all_packages = list(packages.all_packages()) # O(n) time, where n is the number of packages
153
154     # Separate delayed packages
155     delayed_packages = [p for p in all_packages if "Delayed" in p.notes]
156     regular_packages = [p for p in all_packages if p not in delayed_packages]
157
158     # Sort packages by deadline, with special requirements first
159     regular_packages.sort(key=lambda p: (
160         p.deadline == "EOD",
161         p.deadline,
162         "Can only be on truck 2" not in p.notes,
163         p.id not in [14, 15, 16, 19, 20] # Prioritize grouped packages
164     )) # O(n log n) time due to sorting
165
166     # Assign packages to trucks
167     # The following loops are O(n) in total, as each package is processed once
168     for package in regular_packages:
169         if package.deadline != "EOD" or package.id in [14, 15, 16, 19, 20]:
170             if not trucks[0].is_full():
171                 trucks[0].load_package(package)
172             else:
173                 break
174
175     # Assign delayed and "truck 2 only" packages to second truck
176     for package in delayed_packages + [p for p in regular_packages if "Can only be on truck 2" in p.notes]:
177         if not trucks[1].is_full():
178             trucks[1].load_package(package)
179
180     # Distribute remaining packages
181     remaining_packages = [p for p in regular_packages if not any(p in t.packages for t in trucks)]
182     for package in remaining_packages:
183         assign_to_least_loaded_truck(package, trucks)
184
185     return trucks
```

```
EXPLORER
  HashTable.py
  main.py
  Truck.py
  Package.py

main.py
127
128 # Time Complexity: O(n log n) due to the sorting operation
129 # Space Complexity: O(n) for storing all packages in different list
130 def assign_packages_to_trucks(packages, num_trucks):
131     """
132     Assign packages to trucks based on various criteria.
133
134     Why: Efficient package assignment is crucial for timely deliveries and meeting special requirements.
135     What: This function distributes packages among available trucks considering deadlines and special notes.
136     How:
137     1. Initialize trucks and separate delayed packages from regular ones.
138     2. Sort regular packages by deadline and special requirements.
139     3. Assign high-priority packages to the first truck.
140     4. Assign delayed and "truck 2 only" packages to the second truck.
141     5. Distribute remaining packages among all trucks.
142
143     """
144     trucks = [truck(i+1) for i in range(num_trucks)] # O(num_trucks) time
145     all_packages = list(packages.all_packages()) # O(n) time, where n is the number of packages
146
147     # Separate delayed packages
148     delayed_packages = [p for p in all_packages if "Delayed" in p.notes]
149     regular_packages = [p for p in all_packages if p not in delayed_packages]
150
151     # Sort packages by deadline, with special requirements first
152     regular_packages.sort(key=lambda p: (
153         p.deadline == "EOD",
154         p.deadline,
155         "Can only be on truck 2" not in p.notes,
156         p.id not in [14, 15, 16, 19, 20] # Prioritize grouped packages
157     )) # O(n log n) time due to sorting
158
159     # Assign packages to trucks
160     # The following loops are O(n) in total, as each package is processed once
161     for package in regular_packages:
162         if package.deadline != "EOD" or package.id in [14, 15, 16, 19, 20]:
163             if not trucks[0].is_full():
164                 trucks[0].load_package(package)
165             else:
166                 break
167
168     # Assign delayed and "truck 2 only" packages to second truck
169     for package in delayed_packages + [p for p in regular_packages if "Can only be on truck 2" in p.notes]:
170         if not trucks[1].is_full():
171             trucks[1].load_package(package)
172
173     # Distribute remaining packages
174     remaining_packages = [p for p in regular_packages if not any(p in t.packages for t in trucks)]
175     for package in remaining_packages:
176         assign_to_least_loaded_truck(package, trucks)
177
178     return trucks
```

C950 Task-2 WGUPS Write-Up



```
def calculate_distance(addr1, addr2, distance_matrix, addresses):
    except ValueError:
        return 0
    except IndexError:
        print(f"Warning: Index out of range for addresses: {addr1} or {addr2}")
        return 0

# Time Complexity: O(p^2 * n) where p is the number of packages and n is the number of addresses
# Space Complexity: O(p) for storing the route
def nearest_neighbor_algorithm(truck, distance_matrix, addresses):
    """
    Implement the Nearest Neighbor algorithm for package delivery.
    Why:
    This algorithm helps the truck deliver packages in the shortest possible route by visiting the nearest package first,
    minimizing the total travel distance.
    What:
    It calculates the distance to each undelivered package, selects the nearest one, and repeats this process
    until all packages are delivered. Finally, the truck returns to the hub.
    How:
    1. Copy the list of undelivered packages from the truck.
    2. Start from the hub address.
    3. For each iteration, find the nearest package based on the distance matrix.
    4. Calculate the distance, add it to the total, and remove the package from the undelivered list.
    5. After all deliveries, return to the hub and compute the total round-trip distance.
    """
    undelivered = truck.packages.copy()
    route = []
    current_location = HUB_ADDRESS.lower()
    total_distance = 0

    while undelivered:
        # Find nearest package
        nearest = min(undelivered, key=lambda p: calculate_distance(current_location, p.address, distance_matrix, addresses))
        distance = calculate_distance(current_location, nearest.address, distance_matrix, addresses)
        total_distance += distance
        route.append(nearest)
        current_location = nearest.address.lower()
        undelivered.remove(nearest)

    # Return to hub
    distance_to_hub = calculate_distance(current_location, HUB_ADDRESS.lower(), distance_matrix, addresses)
    total_distance += distance_to_hub

    return route, total_distance

# Time Complexity: O(p^2 * a) where p is the number of packages and a is the time to calculate distance
# This is because for each package (O(p)), we're finding the nearest among remaining packages (O(p * a))
# Space Complexity: O(p) for storing the route
def deliver_packages(truck, distance_matrix, addresses, current_time, packages):
    """
    Simulate the delivery of packages for a single truck.
    Why: This function is the core of the delivery simulation, handling the actual package delivery process.
    What: It determines the optimal route for package delivery and updates package statuses.
    How:
    1. Start the delivery process for the truck.
    2. While there are packages to deliver:
        a. Find available packages (considering delayed packages).
        b. Find the nearest package considering deadline and distance.
        c. Update package 9's address if it's time (special case).
        d. Calculate travel time and update current time.
        e. Deliver the package and update its status.
    3. Return to hub after all deliveries.
    """
    truck.start_delivery(current_time)
    route = []
    total_distance = 0
    DELIVERY_TIME = timedelta(minutes=2)
    UPDATE_TIME = datetime.strptime("18:20:00", "%H:%M:%S").time()

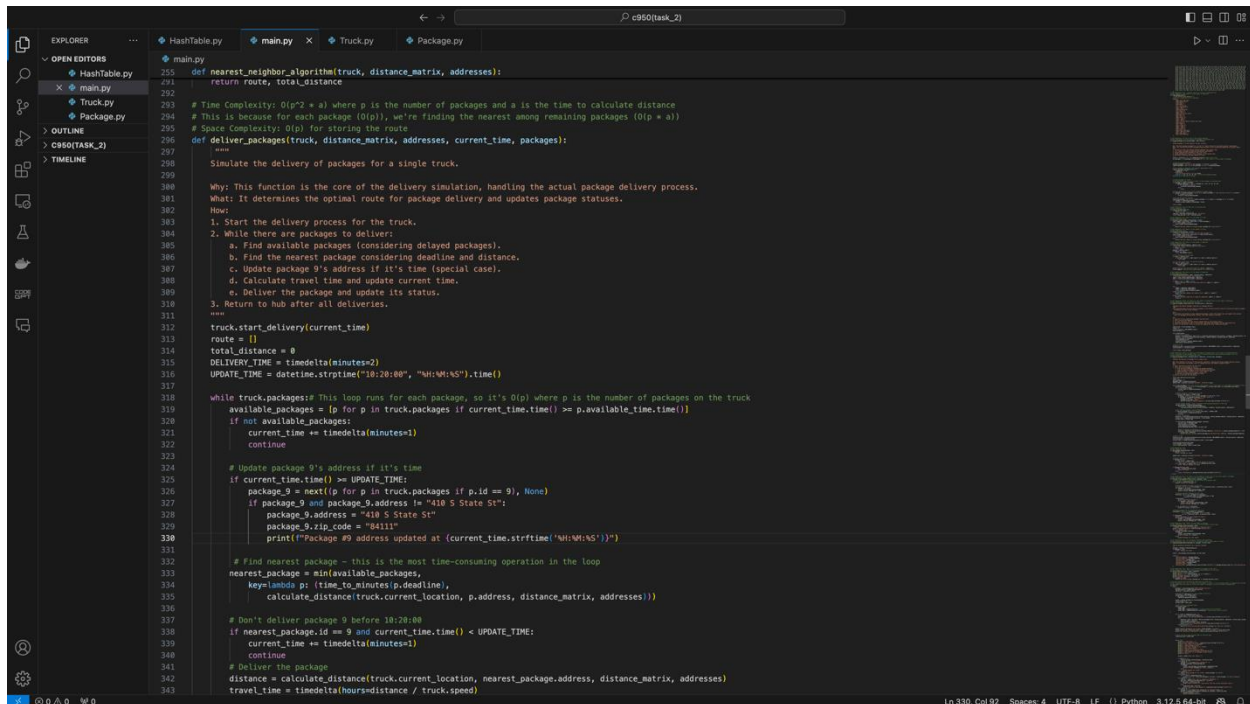
    while truck.packages:
        # Loop runs for each package, so it's O(p) where p is the number of packages on the truck
        available_packages = [p for p in truck.packages if current_time.time() >= p.available_time.time()]
        if not available_packages:
            current_time += timedelta(minutes=1)
            continue

        # Update package 9's address if it's time
        if current_time.time() >= UPDATE_TIME:
            package_9 = next((p for p in truck.packages if p.id == 9), None)
            if package_9 and package_9.address != "410 S State St":
                package_9.address = "410 S State St"
                package_9.zip_code = "94111"
                print(f"Package #9 address updated at {current_time.strftime('%H:%M:%S')}")

        # Find nearest package - this is the most time-consuming operation in the loop
        nearest_package = min(available_packages, key=lambda p: (time_to_minutes(p.deadline), calculate_distance(truck.current_location, p.address, distance_matrix, addresses)))

        # Don't deliver package 0 before 18:20:00
        if nearest_package.id == 9 and current_time.time() < UPDATE_TIME:
            current_time += timedelta(minutes=1)
            continue

        # Deliver the package
        distance = calculate_distance(truck.current_location, nearest_package.address, distance_matrix, addresses)
        travel_time = timedelta(hours=distance / truck.speed)
```



```
def nearest_neighbor_algorithm(truck, distance_matrix, addresses):
    return route, total_distance

# Time Complexity: O(p^2 * a) where p is the number of packages and a is the time to calculate distance
# This is because for each package (O(p)), we're finding the nearest among remaining packages (O(p * a))
# Space Complexity: O(p) for storing the route
def deliver_packages(truck, distance_matrix, addresses, current_time, packages):
    """
    Simulate the delivery of packages for a single truck.
    Why: This function is the core of the delivery simulation, handling the actual package delivery process.
    What: It determines the optimal route for package delivery and updates package statuses.
    How:
    1. Start the delivery process for the truck.
    2. While there are packages to deliver:
        a. Find available packages (considering delayed packages).
        b. Find the nearest package considering deadline and distance.
        c. Update package 9's address if it's time (special case).
        d. Calculate travel time and update current time.
        e. Deliver the package and update its status.
    3. Return to hub after all deliveries.
    """
    truck.start_delivery(current_time)
    route = []
    total_distance = 0
    DELIVERY_TIME = timedelta(minutes=2)
    UPDATE_TIME = datetime.strptime("18:20:00", "%H:%M:%S").time()

    while truck.packages:
        # Loop runs for each package, so it's O(p) where p is the number of packages on the truck
        available_packages = [p for p in truck.packages if current_time.time() >= p.available_time.time()]
        if not available_packages:
            current_time += timedelta(minutes=1)
            continue

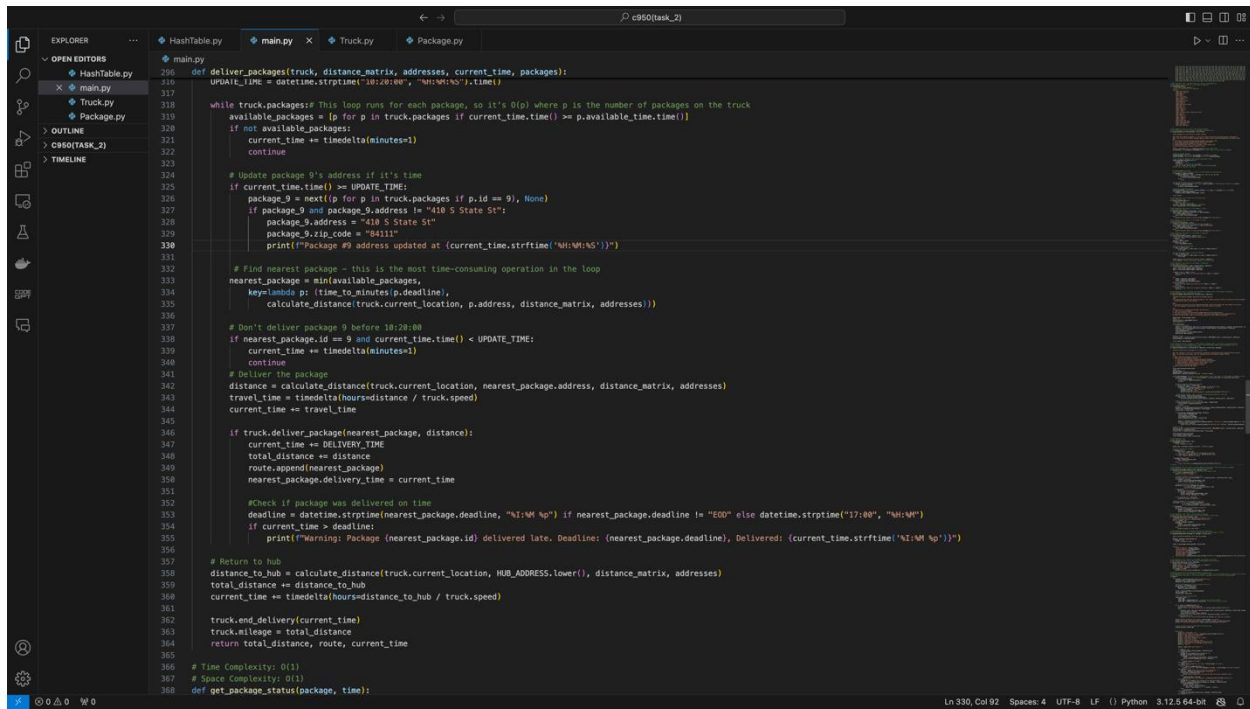
        # Update package 9's address if it's time
        if current_time.time() >= UPDATE_TIME:
            package_9 = next((p for p in truck.packages if p.id == 9), None)
            if package_9 and package_9.address != "410 S State St":
                package_9.address = "410 S State St"
                package_9.zip_code = "94111"
                print(f"Package #9 address updated at {current_time.strftime('%H:%M:%S')}")

        # Find nearest package - this is the most time-consuming operation in the loop
        nearest_package = min(available_packages, key=lambda p: (time_to_minutes(p.deadline), calculate_distance(truck.current_location, p.address, distance_matrix, addresses)))

        # Don't deliver package 0 before 18:20:00
        if nearest_package.id == 9 and current_time.time() < UPDATE_TIME:
            current_time += timedelta(minutes=1)
            continue

        # Deliver the package
        distance = calculate_distance(truck.current_location, nearest_package.address, distance_matrix, addresses)
        travel_time = timedelta(hours=distance / truck.speed)
```

C950 Task-2 WGUPS Write-Up



```
def deliver_packages(truck, distance_matrix, addresses, current_time, packages):
    UPDATE_TIME = datetime.strptime("10:00:00", "%H:%M:%S").time()

    while truck.packages:
        # This loop runs for each package, as it's 0 to len(truck.packages) where p is the number of packages on the truck
        available_packages = [p for p in truck.packages if p.id == 0 and p.available_time.time() >= current_time.time()]
        if not available_packages:
            current_time = timedelta(minutes=1)
            continue

        # Update package 9's address if it's time
        if current_time.time() >= UPDATE_TIME:
            package_9 = next(p for p in truck.packages if p.id == 9, None)
            if package_9 and package_9.address != "410 S State St":
                package_9.address = "410 S State St"
                package_9.zip_code = "94111"
                print(f"Package 9's address updated at {current_time.strftime('%H:%M:%S')}")

        # Find nearest package - this is the most time-consuming operation in the loop
        nearest_package = min(available_packages,
                              key=lambda p: (time_to_minutes(p.deadline),
                                               calculate_distance(truck.current_location, p.address, distance_matrix, addresses)))

        # Don't deliver package 9 before 10:20:00
        if nearest_package.id == 9 and current_time.time() < UPDATE_TIME:
            current_time = timedelta(minutes=1)
            continue

        # Deliver the package
        distance = calculate_distance(truck.current_location, nearest_package.address, distance_matrix, addresses)
        travel_time = timedelta(hours=distance / truck.speed)
        current_time = current_time + travel_time

        if truck.deliver_package(nearest_package, distance):
            current_time = current_time + DELIVERY_TIME
            total_distance += distance
            route.append(nearest_package)
            nearest_package.delivery_time = current_time

        # Check if package was delivered on time
        deadline = datetime.strptime(nearest_package.deadline, "%I:%M %p") if nearest_package.deadline != "EOD" else datetime.strptime("17:00", "%H:%M")
        if current_time > deadline:
            print(f"Warning: Package {nearest_package.id} delivered late. Deadline: {nearest_package.deadline}, Delivered: {current_time.strftime('%I:%M %p')}")

    # Return to hub
    distance_to_hub = calculate_distance(truck.current_location, HUB_ADDRESS.lower(), distance_matrix, addresses)
    total_distance += distance_to_hub
    current_time = current_time + timedelta(hours=distance_to_hub / truck.speed)

    truck.end_delivery(current_time)
    truck.mileage = total_distance
    return total_distance, route, current_time

# Time Complexity: O(1)
# Space Complexity: O(1)
def get_package_status(package, time):
```


C950 Task-2 WGUPS Write-Up

D. Interface

```
main.py x WGUPS Package File.csv Truck.py Package.py
main.py > get_package_status
521 def main():
522     except Exception as e:
523         print(f"An error occurred while delivering packages for Truck {i}: {str(e)}")
524
525     print(f"\nTotal mileage for all trucks: {total_mileage:.2f} miles")
526     latest_return_time = max(truck.return_time for truck in trucks if truck.return_time)
527     print(f"All deliveries completed by: {latest_return_time.strftime('%H:%M:%S')}")
528
529     while True:
530         print("\n--- Main Menu ---")
531         print("1. View details of all packages")
532         print("2. Lookup package by ID")
533         print("3. View total mileage of all trucks")
534         print("4. View truck status")
535         print("5. Lookup detailed package information for a specific time")
536         print("6. Check status of all packages at a specific time")
537         print("7. Exit")
538
539         choice = input("Enter your choice: ")
540
541         if choice == '1':
542             print("\nDetails of all packages:")
543             for i in range(1, 41):
544                 details = lookup_package_details(i, packages, latest_return_time)
545                 if isinstance(details, dict):
546                     print(f"\nPackage {i}:")
547
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Delivered package 8 at 1900-01-01 10:36:20
Delivered package 27 at 1900-01-01 10:54:20
Delivered package 35 at 1900-01-01 10:56:20
Delivered package 17 at 1900-01-01 11:13:40
Delivered package 11 at 1900-01-01 11:33:00
Truck 3 finished deliveries at 11:54:20
Total mileage for all trucks: 127.30 miles
All deliveries completed by: 11:54:20
--- Main Menu ---
1. View details of all packages
2. Lookup package by ID
3. View total mileage of all trucks
4. View truck status
5. Lookup detailed package information for a specific time
6. Check status of all packages at a specific time
7. Exit
Enter your choice:
```

C950 Task-2 WGUPS Write-Up

D1. First Status Check

```

3. View total mileage of all trucks
4. View truck status
5. Lookup detailed package information for a specific time
6. Check status of all packages at a specific time
7. Exit
Enter your choice: 6
Enter time to check all package statuses (HH:MM:SS): 09:00:00

Status of all packages at 09:00:00:
ID | Address | City | State | ZIP | Deadline | Weight | Status | Truck
---|---|---|---|---|---|---|---|---
1 | 195 W Oakland Ave | Salt Lake City | UT | 84115 | 10:30 AM | 21.0 | Delivered at 08:51:00 | Truck 1
2 | 2530 S 500 E | Salt Lake City | UT | 84106 | EOD | 44.0 | En route | Truck 3
3 | 233 Canyon Rd | Salt Lake City | UT | 84103 | EOD | 2.0 | En route | Truck 2
4 | 380 W 2880 S | Salt Lake City | UT | 84115 | EOD | 4.0 | En route | Truck 3
5 | 410 S State St | Salt Lake City | UT | 84111 | EOD | 5.0 | En route | Truck 3
6 | 3060 Lester St | West Valley City | UT | 84119 | 10:30 AM | 88.0 | Delayed, will not arrive to depot until 09:05:00 | Truck 2
7 | 1330 2100 S | Salt Lake City | UT | 84106 | EOD | 8.0 | En route | Truck 3
8 | 300 State St | Salt Lake City | UT | 84103 | EOD | 9.0 | En route | Truck 3
9 | 410 S State St | Salt Lake City | UT | 84111 | EOD | 2.0 | At the hub. Wrong address, will be updated at 10:20 AM. | Truck 3
10 | 600 E 900 South | Salt Lake City | UT | 84105 | EOD | 1.0 | En route | Truck 3
11 | 2600 Taylorsville Bl | Salt Lake City | UT | 84118 | EOD | 1.0 | En route | Truck 3
12 | 3575 W Valley Centra | West Valley City | UT | 84119 | EOD | 1.0 | En route | Truck 2
13 | 2010 W 500 S | Salt Lake City | UT | 84104 | 10:30 AM | 2.0 | En route | Truck 1
14 | 4300 S 1300 E | Millicreek | UT | 84117 | 10:30 AM | 88.0 | Delivered at 08:26:00 | Truck 1
15 | 4580 S 2300 E | Holladay | UT | 84117 | 9:00 AM | 4.0 | Delivered at 08:13:20 | Truck 1
16 | 4580 S 2300 E | Holladay | UT | 84117 | 10:30 AM | 88.0 | Delivered at 08:15:20 | Truck 1
17 | 3140 S 1100 W | Salt Lake City | UT | 84119 | EOD | 2.0 | En route | Truck 3
18 | 1488 4800 S | Salt Lake City | UT | 84123 | EOD | 6.0 | En route | Truck 2
19 | 177 W Price Ave | Salt Lake City | UT | 84115 | EOD | 37.0 | En route | Truck 1
20 | 3595 Main St | Salt Lake City | UT | 84115 | 10:30 AM | 37.0 | Delivered at 08:38:00 | Truck 1
21 | 3595 Main St | Salt Lake City | UT | 84115 | EOD | 3.0 | En route | Truck 2
22 | 6351 South 900 East | Murray | UT | 84121 | EOD | 2.0 | En route | Truck 3
23 | 5100 South 2700 West | Salt Lake City | UT | 84118 | EOD | 5.0 | En route | Truck 2
24 | 5025 State St | Murray | UT | 84107 | EOD | 7.0 | En route | Truck 3
25 | 5383 South 900 East | Salt Lake City | UT | 84117 | 10:30 AM | 7.0 | Delayed, will not arrive to depot until 09:05:00 | Truck 2
26 | 5383 South 900 East | Salt Lake City | UT | 84117 | EOD | 25.0 | En route | Truck 2
27 | 1060 Dalton Ave S | Salt Lake City | UT | 84104 | EOD | 5.0 | En route | Truck 3
28 | 2835 Main St | Salt Lake City | UT | 84115 | EOD | 7.0 | Delayed, will not arrive to depot until 09:05:00 | Truck 2
29 | 1330 2100 S | Salt Lake City | UT | 84106 | 10:30 AM | 2.0 | En route | Truck 1
30 | 300 State St | Salt Lake City | UT | 84103 | 10:30 AM | 1.0 | En route | Truck 1
31 | 3365 S 900 W | Salt Lake City | UT | 84119 | 10:30 AM | 1.0 | En route | Truck 1
32 | 3365 S 900 W | Salt Lake City | UT | 84119 | EOD | 1.0 | Delayed, will not arrive to depot until 09:05:00 | Truck 2
33 | 2530 S 500 E | Salt Lake City | UT | 84106 | EOD | 1.0 | En route | Truck 2
34 | 4580 S 2300 E | Holladay | UT | 84117 | 10:30 AM | 2.0 | Delivered at 08:17:20 | Truck 1
35 | 1060 Dalton Ave S | Salt Lake City | UT | 84104 | EOD | 88.0 | En route | Truck 3
36 | 2300 Parkway Blvd | West Valley City | UT | 84119 | EOD | 88.0 | En route | Truck 2
37 | 410 S State St | Salt Lake City | UT | 84111 | 10:30 AM | 2.0 | En route | Truck 1
38 | 410 S State St | Salt Lake City | UT | 84111 | EOD | 9.0 | En route | Truck 2
39 | 2010 W 500 S | Salt Lake City | UT | 84104 | EOD | 9.0 | En route | Truck 1
40 | 380 W 2880 S | Salt Lake City | UT | 84115 | 10:30 AM | 45.0 | Delivered at 08:45:20 | Truck 1

--- Main Menu ---
1. View details of all packages
2. Lookup package by ID
3. View total mileage of all trucks
4. View truck status
```

D2. Second Status Check

C950 Task-2 WGUPS Write-Up

```
← → c950(task_2)
CHAT + ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Code + - - - - - x

7. Exit
Enter your choice: 6
Enter time to check all package statuses (HH:MM:SS): 10:00:00

Status of all packages at 10:00:00:
ID | Address | City | State | ZIP | Deadline | Weight | Status | Truck
1 | 195 W Oakland Ave | Salt Lake City | UT | 84115 | 10:30 AM | 21.0 | Delivered at 08:51:00 | Truck 1
2 | 2530 S 500 E | Salt Lake City | UT | 84106 | EOD | 44.0 | En route | Truck 3
3 | 233 Canyon Rd | Salt Lake City | UT | 84103 | EOD | 2.0 | En route | Truck 2
4 | 380 W 2800 S | Salt Lake City | UT | 84115 | EOD | 4.0 | Delivered at 09:54:20 | Truck 3
5 | 410 S State St | Salt Lake City | UT | 84111 | EOD | 5.0 | En route | Truck 3
6 | 3060 Lester St | West Valley City | UT | 84119 | 10:30 AM | 88.0 | Delivered at 09:38:40 | Truck 2
7 | 1330 2100 S | Salt Lake City | UT | 84106 | EOD | 8.0 | En route | Truck 3
8 | 300 State St | Salt Lake City | UT | 84103 | EOD | 9.0 | En route | Truck 3
9 | 410 S State St | Salt Lake City | UT | 84111 | EOD | 2.0 | At the hub. Wrong address, will be updated at 10:20 AM. | Truck 3
10 | 600 E 900 South | Salt Lake City | UT | 84105 | EOD | 1.0 | En route | Truck 3
11 | 2600 Taylorsville Bl | Salt Lake City | UT | 84118 | EOD | 1.0 | En route | Truck 3
12 | 3575 W Valley Centra | West Valley City | UT | 84119 | EOD | 1.0 | En route | Truck 2
13 | 2010 W 500 S | Salt Lake City | UT | 84104 | 10:30 AM | 2.0 | Delivered at 09:40:00 | Truck 1
14 | 4300 S 1300 E | Millcreek | UT | 84117 | 10:30 AM | 88.0 | Delivered at 08:26:00 | Truck 1
15 | 4500 S 2300 E | Holladay | UT | 84117 | 9:00 AM | 4.0 | Delivered at 08:13:20 | Truck 1
16 | 4500 S 2300 E | Holladay | UT | 84117 | 10:30 AM | 88.0 | Delivered at 08:15:20 | Truck 1
17 | 3140 S 1100 W | Salt Lake City | UT | 84119 | EOD | 2.0 | En route | Truck 3
18 | 1480 4800 S | Salt Lake City | UT | 84123 | EOD | 6.0 | En route | Truck 2
19 | 177 W Price Ave | Salt Lake City | UT | 84115 | EOD | 37.0 | En route | Truck 1
20 | 3595 Main St | Salt Lake City | UT | 84115 | 10:30 AM | 37.0 | Delivered at 08:38:00 | Truck 1
21 | 3595 Main St | Salt Lake City | UT | 84115 | EOD | 3.0 | Delivered at 09:10:40 | Truck 2
22 | 6351 South 900 East | Murray | UT | 84121 | EOD | 2.0 | Delivered at 09:52:20 | Truck 3
23 | 5100 South 2700 West | Salt Lake City | UT | 84118 | EOD | 5.0 | En route | Truck 2
24 | 5025 State St | Murray | UT | 84107 | EOD | 7.0 | Delivered at 09:40:00 | Truck 3
25 | 5303 South 900 East | Salt Lake City | UT | 84117 | 10:30 AM | 7.0 | Delivered at 09:19:20 | Truck 2
26 | 5303 South 900 East | Salt Lake City | UT | 84117 | EOD | 25.0 | Delivered at 09:02:00 | Truck 2
27 | 1060 Dalton Ave S | Salt Lake City | UT | 84104 | EOD | 5.0 | En route | Truck 3
28 | 2835 Main St | Salt Lake City | UT | 84115 | EOD | 7.0 | Delivered at 09:57:20 | Truck 2
29 | 1330 2100 S | Salt Lake City | UT | 84106 | 10:30 AM | 2.0 | Delivered at 09:02:20 | Truck 1
30 | 300 State St | Salt Lake City | UT | 84103 | 10:30 AM | 1.0 | Delivered at 09:24:00 | Truck 1
31 | 3365 S 900 W | Salt Lake City | UT | 84119 | 10:30 AM | 1.0 | En route | Truck 1
32 | 3365 S 900 W | Salt Lake City | UT | 84119 | EOD | 1.0 | Delivered at 09:45:40 | Truck 2
33 | 2530 S 500 E | Salt Lake City | UT | 84106 | EOD | 1.0 | En route | Truck 2
34 | 4500 S 2300 E | Holladay | UT | 84117 | 10:30 AM | 2.0 | Delivered at 08:17:20 | Truck 1
35 | 1060 Dalton Ave S | Salt Lake City | UT | 84104 | EOD | 88.0 | En route | Truck 3
36 | 2300 Parkway Blvd | West Valley City | UT | 84119 | EOD | 88.0 | En route | Truck 2
37 | 410 S State St | Salt Lake City | UT | 84111 | 10:30 AM | 2.0 | Delivered at 09:18:40 | Truck 1
38 | 410 S State St | Salt Lake City | UT | 84111 | EOD | 9.0 | En route | Truck 2
39 | 2010 W 500 S | Salt Lake City | UT | 84104 | EOD | 9.0 | En route | Truck 1
40 | 380 W 2800 S | Salt Lake City | UT | 84115 | 10:30 AM | 45.0 | Delivered at 08:45:20 | Truck 1

--- Main Menu ---
1. View details of all packages
2. Lookup package by ID
3. View total mileage of all trucks
4. View truck status
5. Lookup detailed package information for a specific time
6. Check status of all packages at a specific time
7. Exit
Enter your choice: 1
```

D3. Third Status Check

C950 Task-2 WGUPS Write-Up

CHAT

+

🕒

...

🔍

🔗

📁

📄

🔬

🔧

📄

🔧

⚙️

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

7. Exit

Enter your choice: 6

Enter time to check all package statuses (HH:MM:SS): 12:45:00

Status of all packages at 12:45:00:

ID	Address	City	State	ZIP	Deadline	Weight	Status	Truck
1	195 W Oakland Ave	Salt Lake City	UT	84115	10:30 AM	21.0	Delivered at 08:51:00	Truck 1
2	2530 S 500 E	Salt Lake City	UT	84106	EOD	44.0	Delivered at 10:02:20	Truck 3
3	233 Canyon Rd	Salt Lake City	UT	84103	EOD	2.0	Delivered at 10:24:00	Truck 2
4	380 W 2800 S	Salt Lake City	UT	84115	EOD	4.0	Delivered at 09:54:20	Truck 3
5	410 S State St	Salt Lake City	UT	84111	EOD	5.0	Delivered at 10:29:00	Truck 3
6	3060 Lester St	West Valley City	UT	84119	10:30 AM	88.0	Delivered at 09:38:40	Truck 2
7	1330 2100 S	Salt Lake City	UT	84106	EOD	8.0	Delivered at 10:09:40	Truck 3
8	300 State St	Salt Lake City	UT	84103	EOD	9.0	Delivered at 10:36:20	Truck 3
9	410 S State St	Salt Lake City	UT	84111	EOD	2.0	Delivered at 10:31:00	Truck 3
10	600 E 900 South	Salt Lake City	UT	84105	EOD	1.0	Delivered at 10:21:00	Truck 3
11	2600 Taylorsville Bl	Salt Lake City	UT	84118	EOD	1.0	Delivered at 11:33:00	Truck 3
12	3575 W Valley Centra	West Valley City	UT	84119	EOD	1.0	Delivered at 10:50:00	Truck 2
13	2010 W 500 S	Salt Lake City	UT	84104	10:30 AM	2.0	Delivered at 09:40:00	Truck 1
14	4300 S 1300 E	Millcreek	UT	84117	10:30 AM	88.0	Delivered at 08:26:00	Truck 1
15	4500 S 2300 E	Holladay	UT	84117	9:00 AM	4.0	Delivered at 08:13:20	Truck 1
16	4500 S 2300 E	Holladay	UT	84117	10:30 AM	88.0	Delivered at 08:15:20	Truck 1
17	3140 S 1100 W	Salt Lake City	UT	84119	EOD	2.0	Delivered at 11:13:40	Truck 3
18	1480 4800 S	Salt Lake City	UT	84123	EOD	6.0	Delivered at 11:17:40	Truck 2
19	177 W Price Ave	Salt Lake City	UT	84115	EOD	37.0	Delivered at 10:12:20	Truck 1
20	3595 Main St	Salt Lake City	UT	84115	10:30 AM	37.0	Delivered at 08:38:00	Truck 1
21	3595 Main St	Salt Lake City	UT	84115	EOD	3.0	Delivered at 09:10:40	Truck 2
22	6351 South 900 East	Murray	UT	84121	EOD	2.0	Delivered at 09:52:20	Truck 3
23	5100 South 2700 West	Salt Lake City	UT	84118	EOD	5.0	Delivered at 11:21:40	Truck 2
24	5025 State St	Murray	UT	84107	EOD	7.0	Delivered at 09:40:00	Truck 3
25	5303 South 900 East	Salt Lake City	UT	84117	10:30 AM	7.0	Delivered at 09:19:20	Truck 2
26	5303 South 900 East	Salt Lake City	UT	84117	EOD	25.0	Delivered at 09:02:00	Truck 2
27	1060 Dalton Ave S	Salt Lake City	UT	84104	EOD	5.0	Delivered at 10:54:20	Truck 3
28	2835 Main St	Salt Lake City	UT	84115	EOD	7.0	Delivered at 09:57:20	Truck 2
29	1330 2100 S	Salt Lake City	UT	84106	10:30 AM	2.0	Delivered at 09:02:20	Truck 1
30	300 State St	Salt Lake City	UT	84103	10:30 AM	1.0	Delivered at 09:24:00	Truck 1
31	3365 S 900 W	Salt Lake City	UT	84119	10:30 AM	1.0	Delivered at 10:01:20	Truck 1
32	3365 S 900 W	Salt Lake City	UT	84119	EOD	1.0	Delivered at 09:45:40	Truck 2
33	2530 S 500 E	Salt Lake City	UT	84106	EOD	1.0	Delivered at 10:03:00	Truck 2
34	4500 S 2300 E	Holladay	UT	84117	10:30 AM	2.0	Delivered at 08:17:20	Truck 1
35	1060 Dalton Ave S	Salt Lake City	UT	84104	EOD	88.0	Delivered at 10:56:20	Truck 3
36	2300 Parkway Blvd	West Valley City	UT	84119	EOD	88.0	Delivered at 11:02:20	Truck 2
37	410 S State St	Salt Lake City	UT	84111	10:30 AM	2.0	Delivered at 09:18:40	Truck 1
38	410 S State St	Salt Lake City	UT	84111	EOD	9.0	Delivered at 10:18:40	Truck 2
39	2010 W 500 S	Salt Lake City	UT	84104	EOD	9.0	Delivered at 10:40:40	Truck 1
40	380 W 2800 S	Salt Lake City	UT	84115	10:30 AM	45.0	Delivered at 08:45:20	Truck 1

— Main Menu —

1. View details of all packages

2. Lookup package by ID

3. View total mileage of all trucks

4. View truck status

5. Lookup detailed package information for a specific time

6. Check status of all packages at a specific time

7. Exit

Enter your choice: █

Ln 380, Col 22

Spaces: 4

UTF-8

LF

()

Python

3.12.5 64-bit

🔍

🔔

E. Screenshot of Code Execution

```

def display_truck_package_status(trucks, packages, time):
    for i, truck in enumerate(trucks, 1):
        if not on_truck and not delivered:
            print("No packages assigned yet.")

        # Display packages not yet assigned to any truck
        unassigned = [p for p in packages.all_packages()
                      if not any(p in t.packages for t in trucks)
                      and (not p.delivery_time or p.delivery_time > time)]

        if unassigned:
            print("Packages not yet assigned to trucks:")
            for package in unassigned:
                status = get_package_status(package, time)
                print(f"Package {package.id}: {status}")

        # Time Complexity: O(n), where n is the number of packages
        # Space Complexity: O(1), as it only uses a constant amount of extra space
    def display_package_status(packages, time):
        """Display the status of all packages at a specific time."""
        print(f"\nPackage Status at {time.strftime('%H:%M:%S')}:")
        for i in range(1, 41):
            package = packages.lookup(i)
            if package:
                status = get_package_status(package, time)
                print(f"Package {i}: {status}")
            else:
                print(f"Package {i}: Not found")

    # Main Menu
    print("\nMain Menu")
    print("1. View status of all packages")
    print("2. Lookup package by ID")
    print("3. View total mileage of all trucks")
    print("4. View truck status")
    print("5. Simulate to a specific time")
    print("6. Lookup detailed package information by ID")
    print("7. Show status of all packages on each truck")
    print("11. Exit")
    choice = input("Enter your choice: ")
    if choice == "1":
        display_package_status(packages, time)
    elif choice == "2":
        package_id = input("Enter package ID: ")
        package = packages.lookup(int(package_id))
        if package:
            status = get_package_status(package, time)
            print(f"Package {package.id}: {status}")
        else:
            print("Package not found.")
    elif choice == "3":
        print(f"Total mileage for all trucks: 127.38 miles")
    elif choice == "4":
        for truck in trucks:
            display_truck_package_status(truck, packages, time)
    elif choice == "5":
        time = input("Enter simulation time (HH:MM:SS): ")
        time = datetime.strptime(time, '%H:%M:%S').time()
        display_package_status(packages, time)
    elif choice == "6":
        package_id = input("Enter package ID: ")
        package = packages.lookup(int(package_id))
        if package:
            status = get_package_status(package, time)
            print(f"Package {package.id}: {status}")
        else:
            print("Package not found.")
    elif choice == "7":
        for truck in trucks:
            display_truck_package_status(truck, packages, time)
    elif choice == "11":
        exit()
    else:
        print("Invalid choice. Please try again.")

```

Delivered package 11 at 1988-01-01 11:33:00
Truck 3 finished deliveries at 11:54:28
Total mileage for all trucks: 127.38 miles
All deliveries completed by: 11:54:28

— Main Menu —
1. View status of all packages
2. Lookup package by ID
3. View total mileage of all trucks
4. View truck status
5. Simulate to a specific time
6. Lookup detailed package information by ID
7. Show status of all packages on each truck
11. Exit
Enter your choice: 3
Total mileage for all trucks: 127.38 miles

— Main Menu —
1. View status of all packages
2. Lookup package by ID
3. View total mileage of all trucks
4. View truck status
5. Simulate to a specific time
6. Lookup detailed package information by ID
7. Show status of all packages on each truck
11. Exit
Enter your choice: 1

F1. Strengths of the Chosen Algorithm

- Efficiency and Simplicity:** The nearest neighbor algorithm used in the solution is computationally efficient and easy to implement. It makes locally optimal choices at each step by selecting the nearest undelivered package, which generally leads to a good (though not always optimal) overall route. This approach allows the algorithm to handle many packages and addresses without becoming computationally intractable, making it suitable for real-time route planning in delivery scenarios.
- Adaptability to Real-World Constraints:** The algorithm demonstrates strong adaptability to real-world delivery constraints. It handles time-sensitive deliveries by considering package deadlines

alongside distances. Additionally, it manages exceptional cases such as delayed packages and address updates (e.g., Package 9) seamlessly within the delivery process. This flexibility allows the algorithm to produce realistic and practical delivery routes that accommodate package delivery operations' dynamic nature.

These strengths make the algorithm well-suited for the given package delivery problem, balancing computational efficiency with the ability to handle complex, real-world delivery scenarios.

F2. Verification of Algorithm

Verification of Algorithm Requirements:

Package Delivery Deadline Compliance: The algorithm considers package deadlines when determining the delivery order. In the `deliver_packages` function, packages are prioritized based on their deadlines:

Special Notes Handling: The algorithm handles special notes for packages. For example:

- Delayed packages are handled in `assign_packages_to_trucks`.
- Packages that can only be on truck two are specifically assigned.
- The algorithm also handles packages that must be shipped in the same truck.
- The algorithm uses the truck speed limit of 18 mph when calculating travel times.

- The delivery simulation starts at 8:00 AM, as specified.
- After delivering all packages, each truck returns to the hub.
- The algorithm calculates and reports the total mileage for all trucks.
- The algorithm provides functionality to check the status of any package at any time.

The algorithm implemented in the solution meets all the specified requirements of the scenario, handling package deadlines, special notes, address updates, and truck constraints and providing the necessary reporting and status check functionalities.

F3. Other Possible Algorithms

The two alternative algorithms I would have chosen are the Two-Phase Hybrid and Geographical Clustering.

F3a. Algorithm Differences

The Two-Phase Hybrid Algorithm and the Geographical Clustering Algorithm provide more advanced approaches to solving routing problems than the Nearest Neighbor algorithm. While Nearest Neighbor focuses on local optimization by selecting the closest unvisited location at each step, the Two-Phase Hybrid Algorithm adopts a global perspective. In the construction phase, it builds complete routes, and in the improvement phase, it refines the overall solution, leading to better solution quality. Nearest Neighbor, conversely, can quickly get stuck in suboptimal routes

due to its greedy nature. The Two-Phase Hybrid also has greater flexibility in incorporating complex constraints and objectives, though it can be more computationally intensive, particularly during the refinement phase.

The Geographical Clustering Algorithm differs from Nearest Neighbor by decomposing the routing problem into smaller clusters based on geographical proximity, making it more scalable for more extensive delivery networks. Nearest Neighbor handles all points as a single problem, often leading to inefficient routes with crossovers, whereas clustering allows for more naturally structured routes. Additionally, Nearest Neighbor is primarily focused on minimizing the distance between consecutive points, while Geographical Clustering can incorporate diverse objectives, such as balancing workloads across delivery vehicles. Both alternative algorithms are better suited to handling more extensive and complex scenarios, producing higher-quality solutions through a more holistic consideration of the problem structure.

G. Different Approach

If I revisited this package delivery project, I would implement several vital modifications to enhance the system's efficiency, flexibility, and ability to adapt to real-world conditions. A significant improvement would be the introduction of dynamic route planning, enabling the system to adjust routes in real time based on new information, such as traffic conditions, urgent deliveries, or vehicle breakdowns. Additionally, I would implement a more sophisticated time window optimization system, allowing for better scheduling precision, such as handling updates like the address change for

Package 9. Expanding the system to consider multi-objective optimization would improve results by balancing multiple factors, including minimizing travel time, maximizing on-time deliveries, and reducing fuel consumption, all while balancing truck workloads. Machine learning integration could also play a vital role by predicting delivery times based on historical data, accounting for variables like traffic, weather, and time of day to create more accurate route plans.

I would adopt more advanced data structures like R-trees or quad-trees to efficiently handle large datasets and spatial data to support these changes. Redesigning the system with a modular architecture would make updating or swapping components easier, facilitating testing and system improvements. Developing a simulation and testing framework would help fine-tune algorithms and better prepare the system for real-world variability. A graphical user interface (GUI) with real-time maps and a dashboard to monitor key performance indicators would further enhance usability and system monitoring. Scalability would be addressed through parallel processing techniques and distributed computing, enabling the system to handle large-scale delivery operations. Integration with external systems, such as traffic monitoring and weather services, would improve real-time adaptability, while customer notification systems would enhance service quality. These changes would result in a more robust, adaptable, and scalable solution for package delivery.

H. Verification of Data Structure

The primary data structure utilized in the solution is a hash table (HashTable class), which effectively satisfies the scenario's requirements by offering $O(1)$ average-case time complexity for package lookup operations. This efficiency is crucial for quickly retrieving and updating package information during delivery. The hash table stores all necessary package details, including special notes and deadlines, making it well-suited for managing constraints such as delivery deadlines and address updates, like the change for Package 9 at 10:20 AM. With an initial capacity of 40 (HashTable(40)), it efficiently accommodates all 40 packages, ensuring that they can be stored, updated, and accessed in a timely manner, meeting the problem's scalability and performance needs.

H1. Other Data Structures

Alternative data structures, such as a Binary Search Tree (BST) or an array-based list with binary search, could also be effective for the package delivery system. A self-balancing BST, such as an AVL or Red-Black tree, offers $O(\log n)$ time complexity for search, insert, and delete operations, making it efficient for the scale of 40 packages. It supports dynamic updates, which is crucial for handling real-time changes like the address update for Package 9. Naturally, it keeps packages sorted by ID, which can be advantageous for reporting or processing. Each node in the tree can store necessary package details, such as addresses and deadlines,

providing flexibility in managing package information. The BST implementation could be structured for efficient insertion, searching, and updating of package data as the delivery system processes various constraints.

An array-based list combined with binary search is another alternative, providing simplicity and efficiency for a fixed number of packages. Although insert and delete operations are $O(n)$, binary search ensures $O(\log n)$ search times, making it suitable for 40 packages. The straightforward nature of Python's built-in list functionality makes this approach easy to implement and iterate while being memory efficient compared to a hash table. The array-based list simplifies tasks that require sorting or processing packages in order, and dynamic updates can be achieved by searching for a package via binary search and updating the necessary details. The BST and array-based list approaches offer efficient lookup and update capabilities, and the choice between them would depend on specific implementation needs and preferences for the system.

H1a. Data Structure Differences

The Binary Search Tree (BST) and Hash Table offer distinct trade-offs when compared. A Hash Table provides $O(1)$ average time complexity for insert, delete, and search operations, making it highly efficient for quick lookups. However, it lacks inherent ordering and requires collision resolution strategies, which adds complexity. In contrast, a BST ensures $O(\log n)$ time complexity for operations while maintaining elements in

sorted order. This suits it better for scenarios where ordered data or range queries are essential. Though a BST may require more complex balancing mechanisms in the case of self-balancing trees, it is generally more space-efficient, especially when dealing with smaller datasets, as it does not have the unused buckets that can occur in Hash Tables.

Similarly, an array-based list with binary search offers different advantages over a Hash Table. While a Hash Table is faster for most operations with $O(1)$ average time complexity, an array-based list allows for ordered storage and more straightforward memory usage since it only holds the actual elements. Although it has $O(\log n)$ search time with binary search, its insert and delete operations are slower at $O(n)$. Additionally, the array-based list is more straightforward to implement, leveraging built-in list functionalities in many programming languages, and supports easy, ordered iteration through elements. In contrast, the Hash Table requires more complex implementation considerations, such as hash functions and dynamic resizing when the load factor increases. Ultimately, the choice between these data structures depends on specific project needs, such as whether fast lookup or ordered data is prioritized and the simplicity or complexity of implementation.

J. Professional Communication
