

C950 Task-2 WGUPS Write-Up

Eric Jacobs

Student ID: 010580832

WGU Email: ejaco70@wgu.edu

11/17/2023

C950 Data Structures and Algorithms II

A. Hash Table Screenshot

```

1  class HashTableEntry:
2      """Represents a single entry within a hash table, to be used when handling collisions."""
3      def __init__(self, key, value): # Constructor
4          self.key = key
5          self.value = value
6          self.next = None # Points to the next entry in case of a collision
7
8  class HashTable:
9      """Creates a hash table to house hash table entries."""
10     def __init__(self):
11         self.size = 40 # Fixed size for the table
12         self.table = [None] * self.size # Initialize table with empty buckets
13
14     def hash_function(self, key):
15         """Computes the index for a key using the modulo operation."""
16         return hash(key) % self.size # Compute hash value.
17
18     def insert(self, key, value):
19         """Inserts a key-value pair into the hash table."""
20         index = self.hash_function(key) # Use 'hash_function' with 'key' to compute index
21         entry = self.table[index] # Retrieve entry from hash table
22         if entry is None: # If bucket is empty
23             self.table[index] = HashTableEntry(key, value) # Create new 'HashTableEntry'
24         else:
25             # Traverse the chain to find the right place to insert
26             while entry.next is not None: # When bucket is not empty, iterate through chain
27                 if entry.key == key: # If entry with same key is found
28                     entry.value = value # Update existing entry
29                     return
30                 entry = entry.next # Move to next entry in chain
31             entry.next = HashTableEntry(key, value) # Add new 'HashTableEntry' at end of chain
32
33     def lookup(self, key):
34         """Looks up the value associated with a key in the hash table."""
35         index = self.hash_function(key) # Use 'hash_function' with 'key' to compute index
36         entry = self.table[index] # Retrieve entry from hash table
37         while entry is not None: # Iterate through chain as long as 'entry' is not 'None' in case of collision
38             if entry.key == key: # Check if entry's key matches lookup key
39                 return entry.value # If match is found, return value
40             entry = entry.next # Move to next entry in chain
41         return None # Key not found
42
43     def delete(self, key):
44         """Removes a key-value pair from the hash table."""
45         index = self.hash_function(key) # Use 'hash_function' with 'key' to compute index
46         entry = self.table[index] # Retrieve entry from hash table
47         prev = None # Initialize previous node to 'None'
48         while entry is not None: # Iterate through chain as long as 'entry' is not 'None'
49             if entry.key == key: # Check if entry's key matches lookup key
50                 if prev is None: # If first in chain
51                     self.table[index] = entry.next # Point to next node
52                 else: # If not the first in chain
53                     prev.next = entry.next # Skips over current 'entry'
54                 return # Exits method
55             prev = entry # Set up for next iteration
56             entry = entry.next

```

Ln 42, Col 1 Spaces: 4 UTF-8 LF Python 3.12.0 64-bit

```

16     return hash(key) % self.size # Compute hash value.
17
18     def insert(self, key, value):
19         """Inserts a key-value pair into the hash table."""
20         index = self.hash_function(key) # Use 'hash_function' with 'key' to compute index
21         entry = self.table[index] # Retrieve entry from hash table
22         if entry is None: # If bucket is empty
23             self.table[index] = HashTableEntry(key, value) # Create new 'HashTableEntry'
24         else:
25             # Traverse the chain to find the right place to insert
26             while entry.next is not None: # When bucket is not empty, iterate through chain
27                 if entry.key == key: # If entry with same key is found
28                     entry.value = value # Update existing entry
29                     return
30                 entry = entry.next # Move to next entry in chain
31             entry.next = HashTableEntry(key, value) # Add new 'HashTableEntry' at end of chain
32
33     def lookup(self, key):
34         """Looks up the value associated with a key in the hash table."""
35         index = self.hash_function(key) # Use 'hash_function' with 'key' to compute index
36         entry = self.table[index] # Retrieve entry from hash table
37         while entry is not None: # Iterate through chain as long as 'entry' is not 'None' in case of collision
38             if entry.key == key: # Check if entry's key matches lookup key
39                 return entry.value # If match is found, return value
40             entry = entry.next # Move to next entry in chain
41         return None # Key not found
42
43     def delete(self, key):
44         """Removes a key-value pair from the hash table."""
45         index = self.hash_function(key) # Use 'hash_function' with 'key' to compute index
46         entry = self.table[index] # Retrieve entry from hash table
47         prev = None # Initialize previous node to 'None'
48         while entry is not None: # Iterate through chain as long as 'entry' is not 'None'
49             if entry.key == key: # Check if entry's key matches lookup key
50                 if prev is None: # If first in chain
51                     self.table[index] = entry.next # Point to next node
52                 else: # If not the first in chain
53                     prev.next = entry.next # Skips over current 'entry'
54                 return # Exits method
55             prev = entry # Set up for next iteration
56             entry = entry.next

```

Ln 42, Col 1 Spaces: 4 UTF-8 LF Python 3.12.0 64-bit

B. Lookup Function Screenshot

-The ‘lookup’ function in the hash table retrieves the ‘Package’ object associated with a given package ID. The ‘Package’ object itself contains all the detailed information required, making the retrieval of specific package aligned with the stated requirements as demonstrated in Option 1 “Check Package Status” in user interface (bottom screenshot)

```

EXPLORER          Package.py M  Main.py M  HashTable.py M  Truck.py M
C950              > __pycache__  > HashTable
> CSV             > __init__.py
> Distance.csv   > Address.csv
> Package.csv    > venv
> Documents       > C950.lml
> HashTable.py    > Main.py
> Package.py      > Truck.py
> venv

```

```

33     def lookup(self, key):
34         """Looks up the value associated with a key in the hash table."""
35         index = self.hash_function(key) # Use 'hash_function' with 'key' to compute index
36         entry = self.table[index] # Retrieve entry from hash table
37         while entry is not None: # Iterate through chain as long as 'entry' is not 'None' in case of collision
38             if entry.key == key: # Check if entry's key matches lookup key
39                 return entry.value # If match is found, return value
40             entry = entry.next # Move to next entry in chain
41         return None # Key not found
42
43     def delete(self, key):
44         """Removes a key-value pair from the hash table."""
45         index = self.hash_function(key) # Use 'hash_function' with 'key' to compute index
46         entry = self.table[index] # Retrieve entry from hash table
47         prev = None # Initialize previous node to 'None'
48         while entry is not None: # Iterate through chain as long as 'entry' is not 'None'
49             if entry.key == key: # Check if entry's key matches lookup key
50                 if prev is None: # If first in chain
51                     self.table[index] = entry.next # Point to next node
52                 else: # If not the first in chain
53                     prev.next = entry.next # Skips over current 'entry'
54                 return # Exits method
55             prev = entry # Set up for next iteration
56             entry = entry.next

```

```

EXPLORER          Package.py M  Main.py M  HashTable.py M  Truck.py M
C950              > simulate_delivery
> CSV             > __init__.py
> Distance.csv   > Address.csv
> Package.csv    > venv
> Documents       > C950.lml
> HashTable.py    > Main.py
> Package.py      > Truck.py
> venv

```

```

191     while True: # Infinite loop
192         print("\nOptions:\n1. Check Package Status\n2. Display All Package Statuses\n3. Display Truck Information\n4. Display Total Mileage\n5. Exit") # Print options to user
193         choice = input("Enter your choice: ") # Ask user for input choice
194         if choice == '1': # Option 1: Check package status
195             package_id_input = input("Enter package ID to check status: ") # Ask user for package ID
196             if package_id_input.isdigit(): # Check for only digits
197                 package_id = int(package_id_input) # Convert 'package_id_input' to int, then store as 'package_id'
198                 package = package_hash_table.lookup(package_id) # Look up package object
199                 if package: # Check if package is found
200                     # Display all details of the found package
201                     print(f"Package ID: {package.package_id}")
202                     print(f"Address: {package.address}")
203                     print(f"City: {package.city}")
204                     print(f"Zip Code: {package.zip_code}")
205                     print(f"Deadline: {package.deadline}")
206                     print(f"Weight: {package.weight}")
207                     print(f"Status: {package.status}")
208                 if package.delivery_time: # Check if delivery time is available
209                     print(f"Delivery Time: {package.delivery_time.strftime('%I:%M %p')}") # Print delivery time
210                 else: # If delivery is unavailable
211                     print("Delivery Time: N/A") # Print N/A
212                 else: # If package is not found
213                     print("No package found with ID {package_id}") # Print no package found
214                 else: # If not digit
215                     print("Invalid input. Please enter a numeric package ID.") # Print invalid input message
216             elif choice == '2': # Option 2: Display all package statuses at a specific time
217                 time_input = input("Enter the time (HH:MM AM/PM) to check package statuses: ") # Request time input
218                 try: # Create a try block
219                     specific_time = datetime.strptime(time_input, "%I:%M %p") # Parse user's input into 'datetime' object
220                     display_package_status_at_time(package_hash_table, specific_time) # If parse is successful, display statuses of all packages at given time
221                 except ValueError: # Catch error if parse is unsuccessful
222                     print("Invalid time format. Please enter time in HH:MM AM/PM format.") # Print invalid format message
223             elif choice == '3': # Option 3: Display truck information
224                 display_truck_info(trucks) # Display truck information
225             elif choice == '4': # Option 4: Display total mileage
226                 total_mileage = sum(truck.total_miles for truck in trucks) # Calculate 'total_mileage'
227                 print(f"Total mileage by all trucks: {total_mileage}") # Print total mileage
228             elif choice == '5': # Option 5: Exit the program
229                 print("Exiting program.") # Print exit message
230                 break # Terminate 'while True' loop
231             else: # If input is not 1-5
232                 print("Invalid choice. Please enter a number between 1 and 5.") # Print input error message
233
234     if __name__ == "__main__":
235         main()

```

C. Original Code

Major Code Blocks

```
Package.py M Main.py M HashTable.py M Truck.py M
Main.py > load_address_data

1 # Main.py
2 # Author: Eric Jacobs
3 # Student ID: 010580832
4 # Title: C950 WGUPS Routing Program
5
6 import csv
7 from datetime import datetime, timedelta
8 from HashTable import HashTable
9 from Package import Package
10 from Truck import Truck
11
12 # Constants for CSV file paths
13 CSV_DIRECTORY = '/Users/ericjacobs/Desktop/Software I/Project/C950/CSV/'
14
15 def load_address_data(filename):
16     """Load address data from CSV file to create a mapping of address to index."""
17     address_index_map = {} # Initialize empty library
18     with open(filename) as csvfile: # Temporary refer to file as 'csvfile' within 'with' block
19         address_reader = csv.reader(csvfile) # Creates CSV reader object to read from 'csvfile'
20         for index, row in enumerate(address_reader):# Iterate over each row in CSV file
21             address = row[2].strip() #Extract 'address' from third row
22             address_index_map[address] = index # Map 'address' to 'index'
23
24     # Print the address_index_map
25     print("Address Index Map:")
26     for address, index in address_index_map.items():
27         print(f"({address}): {index}")
28     return address_index_map
29
30 def load_package_data(filename, hash_table, address_index_map): # address_index_map is created in load_address_data function
31     """Load package data from CSV file and insert into the hash table."""
32     with open(filename) as csvfile: # 'with' makes sure file closes after execution
33         package_reader = csv.reader(csvfile) # Creates CSV reader object to read from 'csvfile'
34         for row in package_reader: # Iterate over each row in CSV file
35             package_id = int(row[0]) # Assign first column of CSV fil as package_id
36             full_address = row[1].strip() # Retrieve address.. strip() removes whitespace
37             address_index = address_index_map.get(full_address) # Use 'full_address' to find index in 'address index map'
38             if address_index is None: # Error tracking
39                 print(f"Error: Address '{full_address}' not found for package {package_id}.")
40             continue
41             package_data = Package(package_id, row[1], row[2], row[4], row[5], row[6], address_index, 'At Hub') # Create Paackage data object
42             hash_table.insert(package_id, package_data) # Insert package data into hash table
43
44 def load_distance_data(filename):
45     """Load distance data from CSV file."""
46     distances = [] # Initialize empty list
47     with open(filename) as csvfile: # Open file as 'csvfile'
48         distance_reader = csv.reader(csvfile) # Open CSV reader object
49         for row_index, row in enumerate(distance_reader): # iterate over each row
50             # Convert each element to float and handle empty strings
51             processed_row = [float(dist.strip()) if dist.strip() else 0.0 for dist in row] # Process each row
52             distances.append(processed_row) # Add row to 'distances' list
53
54             #Debugging
55             #print(f"Raw row {row_index}: {row}")
56             #print(f"Processed row {row_index}: {processed_row}")...
57
58             # Mirror the lower triangular part to the upper part
59             for i in range(len(distances)):
60                 for j in range(i + 1, len(distances)):
61                     distances[i][j] = distances[j][i]
62
63             #Debugging
64             #print("\nComplete Distance Matrix:")
65             for row in distances:
66                 print(row)
67
68             return distances
69
70 def find_nearest_neighbor(current_location, packages, distance_matrix):
71     """Find the nearest unvisited address for delivery."""
72     nearest_distance = float('inf') #Initialize to infinity large value
73     nearest_package = None # Initialize 'nearest_package' to 'None'
74     for package in packages: #Iterate over packages
75         package_address_index = package.address_index # Retrieve 'address_index' from current 'package'
76         distance_to_package = distance_matrix[current_location][package_address_index] # Calculate distance to package
77         if distance_to_package < nearest_distance and not package.is_delivered: # Set conditions
78             nearest_distance = distance_to_package # Set new 'nearest_distance'
79             nearest_package = package # Set new 'nearest_package'
80
81     return nearest_package, nearest_distance # Return updated values
82
83 def load_packages_onto_trucks(trucks, hash_table):
84     """Manually load packages onto specified trucks."""
85     # Packages assignment for each truck
86     truck_packages = {
87         1: [1, 14, 16, 15, 25, 29, 30, 31, 34, 37, 40, 2, 4, 8, 10, 11],
88         2: [3, 6, 18, 36, 38, 13, 20, 19, 39],
89         3: [9, 32, 28, 5, 7, 12, 17, 21, 22, 23, 24, 26, 27, 33, 35]
90     } #Create a dictionary where truck ID is key and values are packages
91
92     for truck_id, package_ids in truck_packages.items(): # Iterate over 'truck_packages' dictionary
93         for package_id in package_ids: # Iterate over each package ID
94             package = hash_table.get(package_id) # Get package object
95             truck = trucks[truck_id] # Get truck object
96             truck.load_package(package)
```

Ln 51, Col 74 Spaces: 4 UTF-8 LF ↴ Python 3.12.0 64-bit

```
Package.py M Main.py M HashTable.py M Truck.py M
Main.py > load_distance_data

42 def load_distance_data(filename):
43     """Load distance data from CSV file."""
44     distances = [] # Initialize empty list
45     with open(filename) as csvfile: # Open file as 'csvfile'
46         distance_reader = csv.reader(csvfile) # Open CSV reader object
47         for row_index, row in enumerate(distance_reader): # iterate over each row
48             # Convert each element to float and handle empty strings
49             processed_row = [float(dist.strip()) if dist.strip() else 0.0 for dist in row] # Process each row
50             distances.append(processed_row) # Add row to 'distances' list
51
52             #Debugging
53             #print(f"Raw row {row_index}: {row}")
54             #print(f"Processed row {row_index}: {processed_row}")...
55
56             # Mirror the lower triangular part to the upper part
57             for i in range(len(distances)):
58                 for j in range(i + 1, len(distances)):
59                     distances[i][j] = distances[j][i]
60
61             #Debugging
62             #print("\nComplete Distance Matrix:")
63             for row in distances:
64                 print(row)
65
66             return distances
67
68 def find_nearest_neighbor(current_location, packages, distance_matrix):
69     """Find the nearest unvisited address for delivery."""
70     nearest_distance = float('inf') #Initialize to infinity large value
71     nearest_package = None # Initialize 'nearest_package' to 'None'
72     for package in packages: #Iterate over packages
73         package_address_index = package.address_index # Retrieve 'address_index' from current 'package'
74         distance_to_package = distance_matrix[current_location][package_address_index] # Calculate distance to package
75         if distance_to_package < nearest_distance and not package.is_delivered: # Set conditions
76             nearest_distance = distance_to_package # Set new 'nearest_distance'
77             nearest_package = package # Set new 'nearest_package'
78
79     return nearest_package, nearest_distance # Return updated values
80
81 def load_packages_onto_trucks(trucks, hash_table):
82     """Manually load packages onto specified trucks."""
83     # Packages assignment for each truck
84     truck_packages = {
85         1: [1, 14, 16, 15, 25, 29, 30, 31, 34, 37, 40, 2, 4, 8, 10, 11],
86         2: [3, 6, 18, 36, 38, 13, 20, 19, 39],
87         3: [9, 32, 28, 5, 7, 12, 17, 21, 22, 23, 24, 26, 27, 33, 35]
88     } #Create a dictionary where truck ID is key and values are packages
89
90     for truck_id, package_ids in truck_packages.items(): # Iterate over 'truck_packages' dictionary
91         for package_id in package_ids: # Iterate over each package ID
92             package = hash_table.get(package_id) # Get package object
93             truck = trucks[truck_id] # Get truck object
94             truck.load_package(package)
```

Ln 51, Col 74 Spaces: 4 UTF-8 LF ↴ Python 3.12.0 64-bit

```
Package.py M Main.py M HashTable.py M Truck.py M

Main.py > load_distance_data
80 def load_packages_onto_trucks(trucks, hash_table):
81     """Manually load packages onto specified trucks."""
82     # Packages assignment for each truck
83     truck_packages = {
84         1: [1, 14, 16, 15, 25, 29, 30, 31, 34, 37, 40, 2, 4, 8, 10, 11],
85         2: [3, 6, 18, 36, 38, 13, 20, 19, 39],
86         3: [9, 32, 28, 5, 7, 12, 17, 21, 22, 23, 24, 26, 27, 33, 35]
87     } # Create a dictionary where truck ID is key and values are packages
88
89     for truck_id, package_ids in truck_packages.items(): # Iterate over 'truck_packages' dictionary
90         for package_id in package_ids: # Iterate over each 'package_id' in 'package_ids'
91             package = hash_table.lookup(package_id) # Look up 'package' in 'hash_table' using 'package_id'
92             if package: # If package is found
93                 trucks[truck_id - 1].load_package(package) # Load package. Truck index starts at 0
94
95     def simulate_truck_delivery(truck, hash_table, distance_matrix, address_index_map):
96         """Simulate delivery for a single truck."""
97         current_location = 0 # Initialize 'current_location'. Start from the hub
98         while truck.packages: # 'while' loop as long as 'packages' are in 'truck'
99             nearest_package, distance_to_package = find_nearest_neighbor(current_location, truck.packages, distance_matrix)
100            if nearest_package: # Check for 'nearest_package'
101                # Debugging
102                """Print("Delivering package (nearest_package.package_id) from location index (current_location) to (nearest_package.address_index).")"""
103                print(f"Distance Matrix Entry: {distance_matrix[current_location][nearest_package.address_index]}")
104                truck.deliver_package(nearest_package, distance_to_package) # Deliver package
105                current_location = nearest_package.address_index # Update 'current_location' to 'address_index' of delivered package
106            else:
107                break # Break loop if no more packages are found
108        # Calculate the distance from the last delivery location back to the hub
109        distance_to_hub = distance_matrix[current_location][0] # Assuming the hub is at index 0
110        truck.total_miles += distance_to_hub # Update the total miles of the truck
111        print(f"Truck {truck.truck_id} returning to hub from location index (current_location). Distance: {distance_to_hub} miles.")
112
113    def simulate_delivery(trucks, hash_table, distance_matrix, address_index_map):
114        """Simulate the delivery process for all trucks."""
115        truck_3_departure_time = datetime.strptime('10:30 AM', '%I:%M %p') # Initiate 'truck_3_departure_time'
116        start_time = datetime.strptime('08:00 AM', '%I:%M %p') # Initiate 'start_time' for other trucks. Convert string to 'datetime' object
117        trucks[0].current_time = start_time # Assign 'start_time'
118        trucks[1].current_time = start_time # Assign 'start_time'
119        simulate_truck_delivery(trucks[0], hash_table, distance_matrix, address_index_map) # Simulate delivery truck 1
120        simulate_truck_delivery(trucks[1], hash_table, distance_matrix, address_index_map) # Simulate delivery truck 2
121        trucks[2].current_time = truck_3_departure_time # Set 'current_time' to 10:30 AM
122        simulate_truck_delivery(trucks[2], hash_table, distance_matrix, address_index_map) # Simulate delivery truck 3
123
124    def display_all_package_statuses(hash_table):
125        """Display the status of all packages."""
126        for package_id in range(1, 41): # Loop through package IDs
127            package = hash_table.lookup(package_id) # Look up each package. Returns object.
128            if package: # Check if package is found
129                print(str(package)) # Print package information as a string
130            else: # If package doesn't exist
131                print(f"Package ID ({package_id}) not found.") # Print not found message
132
133    def display_truck_info(trucks):
134        """Display information about each truck."""
135        for truck in trucks: # Iterate through each 'truck' in 'trucks' list
136            print(f"Truck {truck.truck_id} -- Return Time: {truck.current_time.strftime('%I:%M %p')}, Total Mileage: {truck.total_miles}, Packages: {[p.package_id for p in truck.packages]}")
137
138    def check_package_status(package_id, hash_table):
139        """Check the status of a specific package."""
140        package = hash_table.lookup(package_id) # Look up package.
141        if package: # Check if package exists
142            return str(package) # Return package information as a string
143        else: # If package doesn't exist
144            return f"No package found with ID {package_id}" # Print no package found message
145
146    def calculate_package_delivery_times(trucks, distance_matrix):
147        """Calculate the delivery times for all packages."""
148        for truck in trucks: # Iterate over each 'truck' in 'trucks' list
149            current_time = truck.current_time # Set 'current_time'
150            current_location = 0 # Set 'current_location' to hub
151            for package in truck.packages: # Initialize nested loop that iterates over each 'package' in 'truck'
152                package_location = package.address_index # Retrieve 'address_index' and set as 'package_location'
153                travel_time = calculate_time(distance_matrix[current_location][package_location]) # Calculate travel time to package
154                current_time += timedelta(minutes=travel_time) # Add 'travel_time' to 'current_time'
155                package.delivery_time = current_time # Set package's delivery time
156                current_location = package_location # Update current location
157
158    def calculate_delivery_time(distance):
159        """Calculate delivery time in minutes given a distance."""
160        speed = 18 # Truck speed in miles per hour
161        return distance / speed * 60 # Calculate and return delivery time
162
163    def display_package_status_at_time(hash_table, specific_time):
164        """Display the status of all packages at a specific time."""
165        for package_id in range(1, 41): # Iterate through package IDs
166            package = hash_table.lookup(package_id) # Look up 'package-id', set as 'package'
167            if package: # Check if package is found
168                status = "At Hub" # Initial status assignment
169                if package.delivery_time: # Check for delivery time
170                    if package.delivery_time <= specific_time: # If 'delivery_time' is less or equal to 'specific-time'
171                        status = "Delivered at {package.delivery_time.strftime('%I:%M %p')}!" # Update status
```

Ln 51, Col 74 Spaces: 4 UTF-8 LF ↵ Python 3.12.0 64-bit ⌂

```
Package.py M Main.py M HashTable.py M Truck.py M

Main.py > load_distance_data
124    def display_all_package_statuses(hash_table):
125        """Display the status of all packages."""
126        for package_id in range(1, 41): # Loop through package IDs
127            package = hash_table.lookup(package_id) # Look up each package. Returns object.
128            if package: # Check if package is found
129                print(str(package)) # Print package information as a string
130            else: # If package doesn't exist
131                print(f"Package ID ({package_id}) not found.") # Print not found message
132
133    def display_truck_info(trucks):
134        """Display information about each truck."""
135        for truck in trucks: # Iterate through each 'truck' in 'trucks' list
136            print(f"Truck {truck.truck_id} -- Return Time: {truck.current_time.strftime('%I:%M %p')}, Total Mileage: {truck.total_miles}, Packages: {[p.package_id for p in truck.packages]}")
137
138    def check_package_status(package_id, hash_table):
139        """Check the status of a specific package."""
140        package = hash_table.lookup(package_id) # Look up package.
141        if package: # Check if package exists
142            return str(package) # Return package information as a string
143        else: # If package doesn't exist
144            return f"No package found with ID {package_id}" # Print no package found message
145
146    def calculate_package_delivery_times(trucks, distance_matrix):
147        """Calculate the delivery times for all packages."""
148        for truck in trucks: # Iterate over each 'truck' in 'trucks' list
149            current_time = truck.current_time # Set 'current_time'
150            current_location = 0 # Set 'current_location' to hub
151            for package in truck.packages: # Initialize nested loop that iterates over each 'package' in 'truck'
152                package_location = package.address_index # Retrieve 'address_index' and set as 'package_location'
153                travel_time = calculate_time(distance_matrix[current_location][package_location]) # Calculate travel time to package
154                current_time += timedelta(minutes=travel_time) # Add 'travel_time' to 'current_time'
155                package.delivery_time = current_time # Set package's delivery time
156                current_location = package_location # Update current location
157
158    def calculate_delivery_time(distance):
159        """Calculate delivery time in minutes given a distance."""
160        speed = 18 # Truck speed in miles per hour
161        return distance / speed * 60 # Calculate and return delivery time
162
163    def display_package_status_at_time(hash_table, specific_time):
164        """Display the status of all packages at a specific time."""
165        for package_id in range(1, 41): # Iterate through package IDs
166            package = hash_table.lookup(package_id) # Look up 'package-id', set as 'package'
167            if package: # Check if package is found
168                status = "At Hub" # Initial status assignment
169                if package.delivery_time: # Check for delivery time
170                    if package.delivery_time <= specific_time: # If 'delivery_time' is less or equal to 'specific-time'
171                        status = "Delivered at {package.delivery_time.strftime('%I:%M %p')}!" # Update status
```

Ln 51, Col 74 Spaces: 4 UTF-8 LF ↵ Python 3.12.0 64-bit ⌂

```

163     def display_package_status_at_time(hash_table, specific_time):
164         """Display the status of all packages at a specific time."""
165         for package_id in range(1, 41): # Iterate through package IDs
166             package = hash_table.lookup(package_id) # Look up 'package-id', set as 'package'
167             if package: # Check if package is found
168                 status = "At Hub" # Initial status assignment
169                 if package.delivery_time <= specific_time: # If 'delivery_time' is less or equal to 'specific-time'
170                     status = f"Delivered at {package.delivery_time.strftime('%I:%M %p')}" # Update status
171                 else: # If 'delivery_time' is greater than 'specific_time'
172                     status = "En Route" # Set status
173             address = package.address # Retrieve address
174             deadline = package.deadline
175             print(f"Package ID: {package_id}, Status: {status}, Deadline: {deadline} Address: {address}") # Print package information
176         else: # If no package is found
177             print("Package ID {package_id} not found.") # Print not found message
178
179     def main():
180         """Main function to execute the program."""
181         package_hash_table = HashTable() # Create hash table instance
182         address_index_map = load_address_data(CSV_DIRECTORY + 'Address.csv') # Load address data
183         load_package_data(CSV_DIRECTORY + 'Package.csv', package_hash_table, address_index_map) # Load package data
184         distance_matrix = load_distance_data(CSV_DIRECTORY + 'Distance.csv') # Load distance data
185
186         #Debugging
187         """print("Complete Distance Matrix:")
188         for row in distance_matrix:
189             print(row)"""
189
190         truck1, truck2, truck3 = Truck(1), Truck(2), Truck(3) # Create truck instances
191         trucks = [truck1, truck2, truck3] # Create truck list
192         load_packages_onto_trucks(trucks, package_hash_table) # Load packages onto trucks
193         calculate_package_delivery_times(trucks, distance_matrix) # Calculate package delivery times
194         simulate_delivery(trucks, package_hash_table, distance_matrix, address_index_map) # Simulate package delivery
195
196     while True: # Infinite loop
197         print("\nOptions:\n1. Check Package Status\n2. Display All Package Statuses\n3. Display Truck Information\n4. Display Total Mileage\n5. Exit") # Print options to user
198         choice = input("Enter your choice: ") # Ask user for input choice
199         if choice == '1': # Option 1: Check package status
200             package_id_input = input("Enter package ID to check status: ") # Ask user for package ID
201             if package_id_input.isdigit(): # Check for only digits
202                 package_id = int(package_id_input) # Convert 'package_id_input' to int, then store as 'package_id'
203                 package = package_hash_table.lookup(package_id) # Look up package object
204                 if package: # Check if package is found
205                     # Display all details of the found package
206                     print(f"Package ID: {package.package_id}")
207                     print(f"Address: {package.address}")
208                     print(f"City: {package.city}")
209                     print(f"Zip Code: {package.zip_code}")
210                     print(f"Deadline: {package.deadline}")
211
212             else: # If package is not found
213                 print("No package found with ID {package_id}") # Print no package found
214
215         else: # If not digit
216             print("Invalid input. Please enter a numeric package ID.") # Print invalid input message
217
218     elif choice == '2': # Option 2: Display all package statuses at a specific time
219         time_input = input("Enter the time (HH:MM AM/PM) to check package statuses: ") # Request time input
220         try: # Create a try block
221             specific_time = datetime.strptime(time_input, '%I:%M %p') # Parse user's input into 'datetime' object
222             display_package_status_at_time(package_hash_table, specific_time) # If parse is successful, display statuses of all packages at given time
223         except ValueError: # Catch error if parse is unsuccessful
224             print("Invalid time format. Please enter time in HH:MM AM/PM format.") # Print invalid format message
225
226     elif choice == '3': # Option 3: Display truck information
227         display_truck_info(trucks) # Display truck information
228
229     elif choice == '4': # Option 4: Display total mileage
230         total_mileage = sum(truck.total_miles for truck in trucks) # Calculate 'total_mileage'
231         print(f"Total mileage by all trucks: {total_mileage}") # Print total mileage
232
233     elif choice == '5': # Option 5: Exit the program
234         print("Exiting program.") # Print exit message
235         break # Terminate 'while True' loop
236
237     else: # If input is not 1-5
238         print("Invalid choice. Please enter a number between 1 and 5.") # Print input error message
239
240 if __name__ == "__main__":
241     main()

```

Ln 51, Col 74 Spaces:4 UTF-8 LF ↵ Python 3.12.0 64-bit ⌂

```

195     while True: # Infinite loop
196         print("\nOptions:\n1. Check Package Status\n2. Display All Package Statuses\n3. Display Truck Information\n4. Display Total Mileage\n5. Exit") # Print options to user
197         choice = input("Enter your choice: ") # Ask user for input choice
198         if choice == '1': # Option 1: Check package status
199             package_id_input = input("Enter package ID to check status: ") # Ask user for package ID
200             if package_id_input.isdigit(): # Check for only digits
201                 package_id = int(package_id_input) # Convert 'package_id_input' to int, then store as 'package_id'
202                 package = package_hash_table.lookup(package_id) # Look up package object
203                 if package: # Check if package is found
204                     # Display all details of the found package
205                     print(f"Package ID: {package.package_id}")
206                     print(f"Address: {package.address}")
207                     print(f"City: {package.city}")
208                     print(f"Zip Code: {package.zip_code}")
209                     print(f"Deadline: {package.deadline}")
210                     print(f"Weight: {package.weight}")
211                     print(f"Status: {package.status}")
212
213                     if package.delivery_time: # Check if delivery time is available
214                         print(f"Delivery Time: {package.delivery_time.strftime('%I:%M %p')}") # Print delivery time
215                     else: # If delivery is unavailable
216                         print("Delivery Time: N/A") # Print N/A
217
218                 else: # If package is not found
219                     print("No package found with ID {package_id}") # Print no package found
220
221             else: # If not digit
222                 print("Invalid input. Please enter a numeric package ID.") # Print invalid input message
223
224         elif choice == '2': # Option 2: Display all package statuses at a specific time
225             time_input = input("Enter the time (HH:MM AM/PM) to check package statuses: ") # Request time input
226             try: # Create a try block
227                 specific_time = datetime.strptime(time_input, '%I:%M %p') # Parse user's input into 'datetime' object
228                 display_package_status_at_time(package_hash_table, specific_time) # If parse is successful, display statuses of all packages at given time
229             except ValueError: # Catch error if parse is unsuccessful
230                 print("Invalid time format. Please enter time in HH:MM AM/PM format.") # Print invalid format message
231
232         elif choice == '3': # Option 3: Display truck information
233             display_truck_info(trucks) # Display truck information
234
235         elif choice == '4': # Option 4: Display total mileage
236             total_mileage = sum(truck.total_miles for truck in trucks) # Calculate 'total_mileage'
237             print(f"Total mileage by all trucks: {total_mileage}") # Print total mileage
238
239         elif choice == '5': # Option 5: Exit the program
240             print("Exiting program.") # Print exit message
241             break # Terminate 'while True' loop
242
243         else: # If input is not 1-5
244             print("Invalid choice. Please enter a number between 1 and 5.") # Print input error message
245
246 if __name__ == "__main__":
247     main()

```

Ln 51, Col 74 Spaces:4 UTF-8 LF ↵ Python 3.12.0 64-bit ⌂

Package.py M | Main.py M | HashTable.py M | Truck.py M

```

1  class Package:
2      def __init__(self, package_id, address, city, zip_code, deadline, weight, address_index, status='At the hub'): # Constructor
3          """
4              Initialize a new package with given details.
5
6              :param package_id: Unique identifier for the package.
7              :param address: Delivery address of the package.
8              :param city: City for the delivery address.
9              :param zip_code: Zip code for the delivery address.
10             :param deadline: Delivery deadline for the package.
11             :param weight: Weight of the package.
12             :param address_index: Index of the package's address in the distance matrix.
13             :param status: Current status of the package (default is 'At the hub').
14         """
15
16         self.package_id = package_id
17         self.address = address
18         self.city = city
19         self.zip_code = zip_code
20         self.deadline = deadline
21         self.weight = weight
22         self.address_index = address_index
23         self.status = status # Current status of package
24         self.delivery_time = None # Time when the package is delivered, None if not yet delivered.
25         self.is_delivered = False # Boolean to track whether the package has been delivered.
26
27     def update_status(self, status, delivery_time=None):
28         """
29             Update the status and delivery time of the package.
30
31             :param status: New status to be assigned to the package.
32             :param delivery_time: The time when the package was delivered.
33         """
34
35         self.status = status # Update status
36         self.is_delivered = (status == 'Delivered') # Set 'is_delivered' to 'True' if status is 'Delivered'
37         if delivery_time: # If 'delivery_time' is provided
38             self.delivery_time = delivery_time # Update 'delivery_time'
39             self.delivery_time_formatted = delivery_time.strftime('%I:%M %p') # Formatted delivery time.
40
41     def __str__(self):
42         """
43             String representation of the package's details including delivery time.
44
45             delivery_time_formatted = self.delivery_time.strftime('%I:%M %p') if self.delivery_time else 'N/A' # Convert 'delivery_time' to string, otherwise N/A
46             return f"Package ID: {self.package_id}, Address: {self.address}, City: {self.city}, "
47             "Zip: {self.zip_code}, Deadline: {self.deadline}, Weight: {self.weight}, "
48             f"Status: {self.status}, Delivery Time: {self.delivery_time_formatted}" # Return formatted string

```

Ln 46, Col 110 Spaces:4 UTF-8 LF ↵ Python 3.12.0 64-bit ⌂

Package.py M | Main.py M | HashTable.py M | Truck.py M

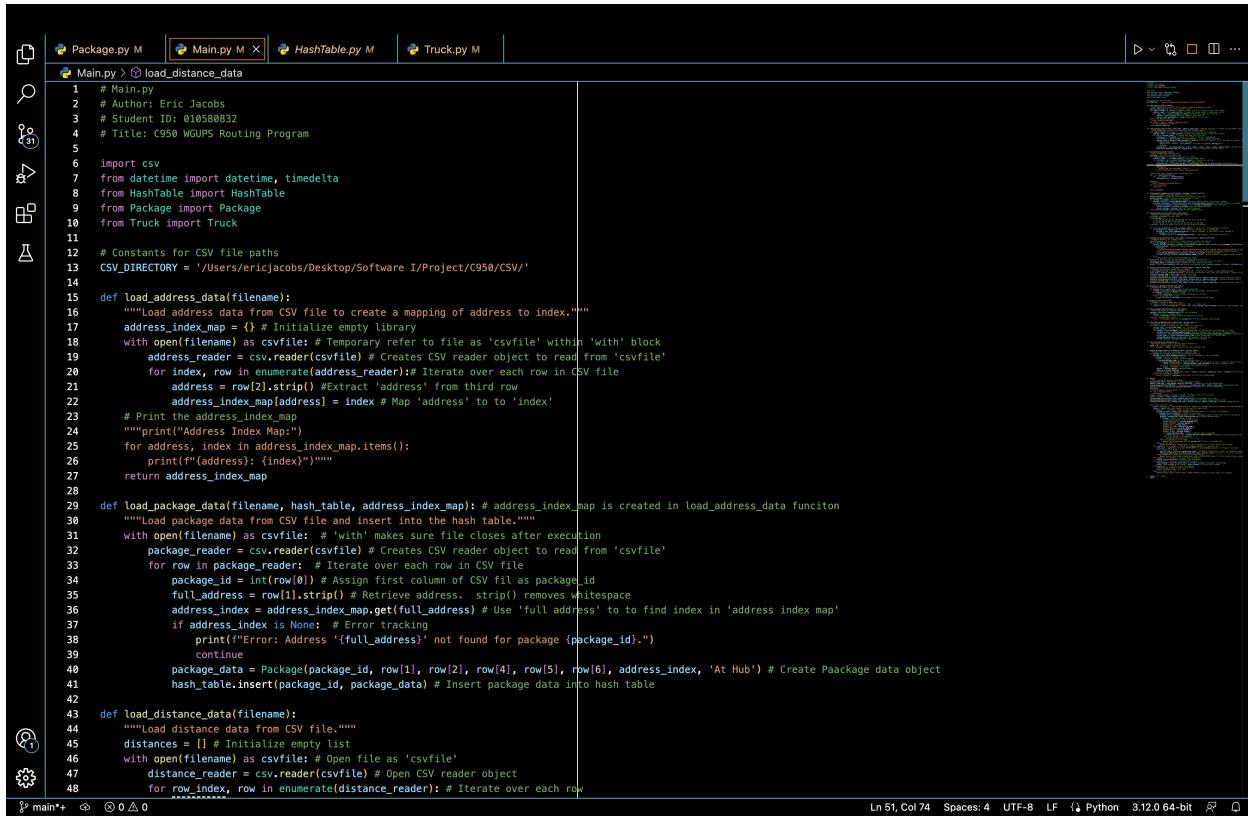
```

5  class Truck:
6      def __init__(self, truck_id, delivery_speed=18, capacity=16): # Constructor
7          self.truck_id = truck_id # Initialize truck ID
8          self.delivery_speed = delivery_speed # Speed of the truck in miles per hour
9          self.capacity = capacity # Maximum number of packages the truck can carry
10         self.packages = [] # List to store packages loaded onto the truck
11         self.total_miles = 0 # Total miles traveled by the truck
12         self.current_location_index = 0 # Assuming index 0 is the hub
13         self.current_time = datetime.strptime('08:00 AM', '%I:%M %p') # Default starting time
14
15     def load_package(self, package):
16         """
17             Load a package onto the truck if there's available capacity.
18             if len(self.packages) < self.capacity: # Check if number of 'packages' is less than trucks 'capacity'
19                 self.packages.append(package) # Add 'package' to 'packages'
20                 package.update_status('En Route') # Update package status to 'En Route'
21             else: # If truck is full
22                 print(f"Truck {self.truck_id} is full and cannot load any more packages.") # Print full statement
23
24     def deliver_package(self, package, distance):
25         """
26             Deliver a package and update truck's total mileage and time.
27             # Ensure package #9 is not delivered before 10:20 AM
28             update_time = datetime.strptime('10:20 AM', '%I:%M %p') # Initialize 'update_time'
29             if package.package_id == 9 and self.current_time < update_time: # Check if package #9 and before 10:20 AM
30                 print("Package #9 cannot be delivered before 10:20 AM. It will be reattempted later.") # Print unsuccessful delivery message
31             return False # Delivery attempt was unsuccessful
32
33         # Print the distance being used for delivery
34         print(f"Delivering package {package.package_id} from location index {self.current_location_index} to {package.address_index}. Distance: {distance} miles.")
35
36         # Calculate delivery time based on distance
37         delivery_time = self.current_time + timedelta(hours=distance / self.delivery_speed)
38         package.update_status('Delivered', delivery_time) # Update package status to 'Delivered'
39         self.total_miles += distance # Update total miles
40         self.current_location_index = package.address_index # Update current location
41         self.current_time = delivery_time # Update current time
42
43     def return_to_hub(self, distance_matrix):
44         """
45             Simulate the truck's return to the hub and update its mileage.
46             return_distance = distance_matrix[self.current_location_index][0] # Calculate 'return_distance'
47             self.total_miles += return_distance # Update total miles for the return trip
48             self.current_location_index = 0 # Reset location to the hub
49             self.current_time += timedelta(hours=return_distance / self.delivery_speed) # Calculate and update 'current_time'
50             print(f"Truck {self.truck_id} returning to hub, total miles: {self.total_miles}") # Print return message
51
52     def __str__(self):
53         """
54             String representation of the truck's current package load.
55             return f"Truck {self.truck_id} -> Packages: {[p.package_id for p in self.packages]}" # Create and return formatted string

```

Ln 44, Col 4 Spaces:4 UTF-8 LF ↵ Python 3.12.0 64-bit ⌂

C1: Identification Information



The screenshot shows a terminal window with the following details:

- File tabs: Package.py M, Main.py M (highlighted), HashTable.py M, Truck.py M.
- Code area:

```
1 # Main.py
2 # Author: Eric Jacobs
3 # Student ID: 0105080832
4 # Title: C950 WGUPS Routing Program
5
6 import csv
7 from datetime import datetime, timedelta
8 from HashTable import HashTable
9 from Package import Package
10 from Truck import Truck
11
12 # Constants for CSV file paths
13 CSV_DIRECTORY = '/Users/ericjacobs/Desktop/Software I/Project/C950/CSV/'
14
15 def load_address_data(filename):
16     """Load address data from CSV file to create a mapping of address to index."""
17     address_index_map = {} # Initialize empty library
18     with open(filename) as csvfile: # csvfile refer to file as 'csvfile' within 'with' block
19         address_reader = csv.reader(csvfile) # Creates CSV reader object to read from 'csvfile'
20         for index, row in enumerate(address_reader):# Iterate over each row in CSV file
21             address = row[2].strip() #Extract 'address' from third row
22             address_index_map[address] = index # Map 'address' to 'index'
23     # Print the address_index_map
24     """print("Address Index Map:")
25     for address, index in address_index_map.items():
26         print(f'{address}: {index}')"""
27     return address_index_map
28
29 def load_package_data(filename, hash_table, address_index_map): # address_index_map is created in load_address_data function
30     """Load package data from CSV file and insert into the hash table."""
31     with open(filename) as csvfile: # 'with' makes sure file closes after execution
32         package_reader = csv.reader(csvfile) # Creates CSV reader object to read from 'csvfile'
33         for row in package_reader: # Iterate over each row in CSV file
34             package_id = int(row[0]) # Assign first column of CSV fil as package_id
35             full_address = row[1].strip() # Retrieve address, strip() removes whitespace
36             address_index = address_index_map.get(full_address) # Use 'full address' to find index in 'address index map'
37             if address_index is None: # Error tracking
38                 print(f"Error: Address '{full_address}' not found for package {package_id}.")
39                 continue
40             package_data = Package(package_id, row[1], row[2], row[4], row[5], row[6], address_index, 'At Hub') # Create Paackage data object
41             hash_table.insert(package_id, package_data) # Insert package data into hash table
42
43 def load_distance_data(filename):
44     """Load distance data from CSV file."""
45     distances = [] # Initialize empty list
46     with open(filename) as csvfile: # Open file as 'csvfile'
47         distance_reader = csv.reader(csvfile) # Open CSV reader object
48         for row_index, row in enumerate(distance_reader): # iterate over each row
```
- Status bar: Ln 51, Col 74, Spaces:4, UTF-8, LF, Python 3.12.0 64-bit.

C2: Process and Flow Comments

```

1  #!/usr/bin/python
2
3  # Import required modules
4  import csv
5  import sys
6  import time
7  import datetime
8  import math
9
10 # Define class HashTable
11 class HashTable:
12     def __init__(self):
13         self.size = 10
14         self.table = [[None] * 2 for i in range(self.size)]
15
16     def insert(self, key, value):
17         index = self.get_index(key)
18         if self.table[index][0] == None:
19             self.table[index] = [key, value]
20         else:
21             self.table[index].append(value)
22
23     def get(self, key):
24         index = self.get_index(key)
25         if self.table[index][0] == None:
26             return None
27         else:
28             return self.table[index][1]
29
30     def get_index(self, key):
31         hash = self.get_hash(key)
32         index = hash % self.size
33         return index
34
35     def get_hash(self, key):
36         hash = 0
37         for c in key:
38             hash += ord(c)
39         return hash
40
41     def print_table(self):
42         for row in self.table:
43             print(row)
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140

```

This screenshot shows the `Main.py` file in a code editor. The code is annotated with extensive comments explaining the process flow for truck delivery. It starts by importing necessary modules like `csv`, `sys`, `time`, and `datetime`. It defines a `HashTable` class for managing package data. The main logic involves simulating truck delivery for a single truck or multiple trucks, calculating distances, and updating package status. The code uses `if` statements to handle edge cases like no packages found or reaching the hub.

```

1  #!/usr/bin/python
2
3  # Import required modules
4  import csv
5  import sys
6  import time
7  import datetime
8  import math
9
10 # Define class HashTable
11 class HashTable:
12     def __init__(self):
13         self.size = 10
14         self.table = [[None] * 2 for i in range(self.size)]
15
16     def insert(self, key, value):
17         index = self.get_index(key)
18         if self.table[index][0] == None:
19             self.table[index] = [key, value]
20         else:
21             self.table[index].append(value)
22
23     def get(self, key):
24         index = self.get_index(key)
25         if self.table[index][0] == None:
26             return None
27         else:
28             return self.table[index][1]
29
30     def get_index(self, key):
31         hash = self.get_hash(key)
32         index = hash % self.size
33         return index
34
35     def get_hash(self, key):
36         hash = 0
37         for c in key:
38             hash += ord(c)
39         return hash
40
41     def print_table(self):
42         for row in self.table:
43             print(row)
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140

```

This screenshot shows the same `Main.py` file with additional comments. These comments focus on the calculation of delivery times. They explain how the code iterates over trucks and packages, calculates travel times based on distance and speed, and updates package delivery times. The comments also describe how package status is updated and printed.

D: Interface

D1: First Status Check

```
1.195 W Oakland Ave, Salt Lake City, UT 84115, 10:30 AM, 21 Kilos.....
```

Truck 3 -> Return Time: 12:56 PM, Total Mileage: 51.6, Packages: [9, 32, 28, 5, 7, 12, 17, 21, 22, 23, 24, 26, 27, 33, 35]

```
Options:
1. Check Package Status
2. Display All Package Statuses
3. Display Truck Information
4. Display Total Mileage
5. Exit

Enter your choice: 2

Enter the time (HH:MM AM/PM) to check package statuses: 09:00 AM
Package ID: 1, Status: Delivered at 08:39 AM, Deadline: 10:30 AM Address: 195 W Oakland Ave
Package ID: 2, Status: En Route, Deadline: EOD Address: 2530 S 500 E
Package ID: 3, Status: En Route, Deadline: EOD Address: 233 Canyon Rd
Package ID: 4, Status: Delivered at 08:42 AM, Deadline: EOD Address: 380 W 2880 S
Package ID: 5, Status: En Route, Deadline: EOD Address: 410 S State St
Package ID: 6, Status: Delivered at 08:42 AM, Deadline: EOD Address: 3068 Lester St
Package ID: 7, Status: En Route, Deadline: EOD Address: 1330 2100 S
Package ID: 8, Status: En Route, Deadline: EOD Address: 300 State St
Package ID: 9, Status: En Route, Deadline: EOD Address: 300 State St
Package ID: 10, Status: En Route, Deadline: EOD Address: 600 E 900 South
Package ID: 11, Status: En Route, Deadline: EOD Address: 2600 Taylorsville Blvd
Package ID: 12, Status: Delivered at 08:42 AM, Deadline: 10:30 AM Address: 1330 2100 W Valley Central Station bus Loop
Package ID: 13, Status: En Route, Deadline: 10:30 AM Address: 2110 W 500 S
Package ID: 14, Status: Delivered at 08:42 AM, Deadline: 10:30 AM Address: 4300 S 1300 E
Package ID: 15, Status: Delivered at 08:13 AM, Deadline: 9:00 AM Address: 4580 S 2300 E
Package ID: 16, Status: Delivered at 08:13 AM, Deadline: 10:30 AM Address: 4580 S 2300 E
Package ID: 17, Status: Delivered at 08:42 AM, Deadline: 10:30 AM Address: 1330 2100 W
Package ID: 18, Status: En Route, Deadline: EOD Address: 1480 S 800 S
Package ID: 19, Status: Delivered at 08:49 AM, Deadline: EOD Address: 177 W Price Ave
Package ID: 20, Status: Delivered at 08:48 AM, Deadline: 10:30 AM Address: 3595 Main St
Package ID: 21, Status: En Route, Deadline: EOD Address: 3595 Main St
Package ID: 22, Status: En Route, Deadline: EOD Address: 6351 South 900 East
Package ID: 23, Status: Delivered at 08:42 AM, Deadline: 10:30 AM Address: 1330 2100 W
Package ID: 24, Status: En Route, Deadline: EOD Address: 5025 State St
Package ID: 25, Status: En Route, Deadline: 10:30 AM Address: 5383 South 900 East #104
Package ID: 26, Status: En Route, Deadline: EOD Address: 1060 Dalton Ave S
Package ID: 27, Status: En Route, Deadline: EOD Address: 410 S State St
Package ID: 28, Status: Delivered at 08:42 AM, Deadline: 10:30 AM Address: 3068 Lester St
Package ID: 29, Status: Delivered at 08:29 AM, Deadline: 10:30 AM Address: 1330 2100 S
Package ID: 30, Status: En Route, Deadline: EOD Address: 300 State St
Package ID: 31, Status: Delivered at 08:58 AM, Deadline: 10:30 AM Address: 3365 S 900 W
Package ID: 32, Status: En Route, Deadline: EOD Address: 3365 S 900 W
Package ID: 33, Status: En Route, Deadline: EOD Address: 2530 S 500 E
Package ID: 34, Status: Delivered at 08:42 AM, Deadline: 10:30 AM Address: 4580 S 2300 E
Package ID: 35, Status: En Route, Deadline: EOD Address: 1060 Dalton Ave S
Package ID: 36, Status: En Route, Deadline: EOD Address: 2300 Parkway Blvd
Package ID: 37, Status: En Route, Deadline: 10:30 AM Address: 410 S State St
Package ID: 38, Status: En Route, Deadline: EOD Address: 410 S State St
Package ID: 39, Status: En Route, Deadline: EOD Address: 2010 W 500 S
Package ID: 40, Status: Delivered at 08:42 AM, Deadline: 10:30 AM Address: 380 W 2880 S

Options:
1. Check Package Status
2. Display All Package Statuses
3. Display Truck Information
4. Display Total Mileage
5. Exit

Enter your choice: 1
```

Col 13 Ln 19, Col 63 Spaces: 4 UTF-8 LF CSV ⌂

D2: Second Status Check

```
1.195 W Oakland Ave, Salt Lake City, UT 84115, 10:30 AM, 21 Kilos.....
```

Package ID: 40, Status: Delivered at 08:42 AM, Deadline: 10:30 AM Address: 380 W 2880 S

```
Options:
1. Check Package Status
2. Display All Package Statuses
3. Display Truck Information
4. Display Total Mileage
5. Exit

Enter your choice: 1

Enter the time (HH:MM AM/PM) to check package statuses: 10:00 AM
Package ID: 1, Status: Delivered at 08:39 AM, Deadline: 10:30 AM Address: 195 W Oakland Ave
Package ID: 2, Status: En Route, Deadline: EOD Address: 2530 S 500 E
Package ID: 3, Status: En Route, Deadline: EOD Address: 233 Canyon Rd
Package ID: 4, Status: Delivered at 08:42 AM, Deadline: EOD Address: 380 W 2880 S
Package ID: 5, Status: En Route, Deadline: EOD Address: 410 S State St
Package ID: 6, Status: Delivered at 08:42 AM, Deadline: EOD Address: 3068 Lester St
Package ID: 7, Status: En Route, Deadline: EOD Address: 1330 2100 S
Package ID: 8, Status: Delivered at 09:27 AM, Deadline: EOD Address: 300 State St
Package ID: 9, Status: En Route, Deadline: EOD Address: 300 State St
Package ID: 10, Status: Delivered at 09:17 AM, Deadline: EOD Address: 600 E 900 South
Package ID: 11, Status: En Route, Deadline: EOD Address: 2600 Taylorsville Blvd
Package ID: 12, Status: Delivered at 09:41 AM, Deadline: 10:30 AM Address: 1330 2100 W Valley Central Station bus Loop
Package ID: 13, Status: Delivered at 09:41 AM, Deadline: 10:30 AM Address: 2110 W 500 S
Package ID: 14, Status: Delivered at 08:06 AM, Deadline: 10:30 AM Address: 4300 S 1300 E
Package ID: 15, Status: Delivered at 08:13 AM, Deadline: 9:00 AM Address: 4580 S 2300 E
Package ID: 16, Status: Delivered at 08:13 AM, Deadline: 10:30 AM Address: 4580 S 2300 E
Package ID: 17, Status: Delivered at 08:13 AM, Deadline: 10:30 AM Address: 1330 2100 W
Package ID: 18, Status: En Route, Deadline: EOD Address: 1480 S 800 S
Package ID: 19, Status: Delivered at 08:49 AM, Deadline: EOD Address: 177 W Price Ave
Package ID: 20, Status: Delivered at 08:48 AM, Deadline: 10:30 AM Address: 3595 Main St
Package ID: 21, Status: En Route, Deadline: EOD Address: 3595 Main St
Package ID: 22, Status: En Route, Deadline: EOD Address: 6351 South 900 East
Package ID: 23, Status: Delivered at 08:42 AM, Deadline: 10:30 AM Address: 1330 2100 W
Package ID: 24, Status: En Route, Deadline: EOD Address: 5025 State St
Package ID: 25, Status: Delivered at 09:51 AM, Deadline: 10:30 AM Address: 5383 South 900 East #104
Package ID: 26, Status: En Route, Deadline: EOD Address: 1060 Dalton Ave S
Package ID: 27, Status: En Route, Deadline: EOD Address: 410 S State St
Package ID: 28, Status: Delivered at 08:42 AM, Deadline: 10:30 AM Address: 2835 Main St
Package ID: 29, Status: Delivered at 09:27 AM, Deadline: 10:30 AM Address: 1330 2100 S
Package ID: 30, Status: Delivered at 09:27 AM, Deadline: 10:30 AM Address: 300 State St
Package ID: 31, Status: Delivered at 08:58 AM, Deadline: 10:30 AM Address: 3365 S 900 W
Package ID: 32, Status: En Route, Deadline: EOD Address: 2530 S 500 E
Package ID: 33, Status: Delivered at 08:42 AM, Deadline: 10:30 AM Address: 4580 S 2300 E
Package ID: 34, Status: Delivered at 08:42 AM, Deadline: EOD Address: 1060 Dalton Ave S
Package ID: 35, Status: En Route, Deadline: EOD Address: 2300 Parkway Blvd
Package ID: 36, Status: En Route, Deadline: EOD Address: 2300 Parkway Blvd
Package ID: 37, Status: Delivered at 09:23 AM, Deadline: 10:30 AM Address: 410 S State St
Package ID: 38, Status: En Route, Deadline: EOD Address: 2010 W 500 S
Package ID: 39, Status: En Route, Deadline: EOD Address: 380 W 2880 S
Package ID: 40, Status: Delivered at 08:42 AM, Deadline: 10:30 AM Address: 380 W 2880 S

Options:
1. Check Package Status
2. Display All Package Statuses
3. Display Truck Information
4. Display Total Mileage
5. Exit

Enter your choice: 1
```

Col 13 Ln 19, Col 63 Spaces: 4 UTF-8 LF CSV ⌂

D3: Third Status Check

The screenshot shows a terminal window with several tabs open at the top: Package.py M, Main.py M, Package.csv X, and Truck.py M. The main area displays a CSV file with package status information. The first few rows of the CSV are:

| 1 | 1.195 W Oakland Ave.Salt Lake City,UT.84115.10:30 AM.21 Kilos..... |
|---|--|
| Package ID: 40, Status: Delivered at 08:42 AM, Deadline: 10:30 AM Address: 388 W 2888 S | |
| Options: | |
| 1. Check Package Status | |
| 2. Display All Package Statuses | |
| 3. Display Truck Information | |
| 4. Display Total Mileage | |
| 5. Exit | |

The terminal then prompts the user to enter a choice:

```
Enter your choice: 2
Enter the time (HH:MM AM/PM) to check package statuses: 01:00 PM
```

Following this, a large list of packages is displayed, each with its ID, status, deadline, and address. The list continues for many more packages, ending with:

```
Options:
1. Check Package Status
2. Display All Package Statuses
3. Display Truck Information
4. Display Total Mileage
5. Exit
Enter your choice: ■
```

At the bottom of the terminal window, there are various status indicators and file paths.

E. Screenshot of Code Execution

```
(base) ericjacobs@Erics-MacBook-Pro:~/Desktop/Software 1/Project/C950/Main.py"
CSV > Package.csv
1 1.195 W Oakland Ave,Salt Lake City,UT,84115,10:30 AM,21 Kilos.....
DELIVERING
Delivering package 14 from location index 0 to 20. Distance: 1.0 miles.
Delivering package 16 from location index 0 to 21. Distance: 2.0 miles.
Delivering package 16 from location index 20 to 21. Distance: 2.0 miles.
Delivering package 15 from location index 21 to 21. Distance: 0.0 miles.
Delivering package 15 from location index 21 to 21. Distance: 0.0 miles.
Delivering package 34 from location index 21 to 21. Distance: 0.0 miles.
Delivering package 34 from location index 21 to 21. Distance: 0.0 miles.
Delivering package 29 from location index 21 to 2. Distances: 5.0 miles.
Delivering package 29 from location index 21 to 2. Distance: 5.0 miles.
Delivering package 1 from location index 2 to 5. Distance: 2.8 miles.
Delivering package 1 from location index 2 to 5. Distance: 2.8 miles.
Delivering package 40 from location index 5 to 18. Distance: 1.1 miles.
Delivering package 40 from location index 5 to 18. Distance: 1.1 miles.
Delivering package 4 from location index 18 to 18. Distance: 0.0 miles.
Delivering package 4 from location index 18 to 18. Distance: 0.0 miles.
Delivering package 20 from location index 18 to 17. Distance: 1.6 miles.
Delivering package 20 from location index 18 to 17. Distance: 1.6 miles.
Delivering package 19 from location index 17 to 4. Distance: 0.5 miles.
Delivering package 19 from location index 17 to 4. Distance: 0.5 miles.
Delivering package 31 from location index 4 to 15. Distance: 2.7 miles.
Delivering package 31 from location index 4 to 15. Distance: 2.7 miles.
Delivering package 10 from location index 15 to 25. Distance: 5.7 miles.
Delivering package 10 from location index 15 to 25. Distance: 5.7 miles.
Delivering package 37 from location index 25 to 19. Distance: 1.8 miles.
Delivering package 37 from location index 25 to 19. Distance: 1.8 miles.
Delivering package 30 from location index 19 to 12. Distance: 1.0 miles.
Delivering package 30 from location index 19 to 12. Distance: 1.0 miles.
Delivering package 9 from location index 12 to 12. Distance: 0.0 miles.
Delivering package 9 from location index 12 to 6. Distance: 4.2 miles.
Delivering package 13 from location index 12 to 6. Distance: 4.2 miles.
Truck 1 returning to hub from location index 6. Distance: 10.9 miles.
Delivering package 5 from location index 10 to 13. Distance: 1.2 miles.
Delivering package 5 from location index 10 to 13. Distance: 5.2 miles.
Delivering package 25 from location index 13 to 24. Distance: 7.2 miles.
Delivering high-priority package 25 from location index 13 to 24. Distance: 7.2 miles.
Delivering package 2 from location index 24 to 9. Distance: 4.8 miles.
Delivering package 2 from location index 24 to 9. Distance: 4.8 miles.
Delivering package 38 from location index 9 to 10. Distance: 0.8 miles.
Delivering package 3 from location index 19 to 8. Distance: 1.0 miles.
Delivering package 3 from location index 19 to 8. Distance: 1.0 miles.
Delivering package 39 from location index 8 to 6. Distance: 4.2 miles.
Delivering package 39 from location index 8 to 6. Distance: 4.2 miles.
Delivering package 36 from location index 6 to 7. Distance: 0.8 miles.
Delivering package 36 from location index 6 to 7. Distance: 0.8 miles.
Delivering package 18 from location index 7 to 3. Distance: 4.0 miles.
Delivering package 18 from location index 7 to 3. Distance: 4.0 miles.
Delivering package 11 from location index 3 to 10. Distance: 1.0 miles.
Delivering package 11 from location index 3 to 10. Distance: 1.0 miles.
Truck 2 returning to hub from location index 10. Distance: 6.0 miles.
Delivering package 21 from location index 8 to 12. Distance: 2.0 miles.
Delivering package 21 from location index 8 to 12. Distance: 2.0 miles.
Delivering package 28 from location index 17 to 11. Distance: 1.2 miles.
Delivering package 28 from location index 17 to 11. Distance: 1.2 miles.
Delivering package 33 from location index 11 to 9. Distance: 1.1 miles.
```

Col 13: Ln 19, Col 63 Spaces: 4 UTF-8 LF CSV □

```
(base) ericjacobs@Erics-MacBook-Pro:~/Desktop/Software 1/Project/C950/Main.py"
CSV > Package.csv
1 1.195 W Oakland Ave,Salt Lake City,UT,84115,10:30 AM,21 Kilos.....
DELIVERING
Delivering package 28 from location index 17 to 11. Distance: 1.2 miles.
Delivering package 33 from location index 17 to 9. Distance: 1.1 miles.
Delivering package 19 from location index 11 to 12. Distance: 1.1 miles.
Delivering package 7 from location index 9 to 2. Distance: 1.6 miles.
Delivering package 7 from location index 9 to 2. Distance: 1.6 miles.
Delivering package 5 from location index 2 to 19. Distance: 4.3 miles.
Delivering package 5 from location index 2 to 19. Distance: 4.3 miles.
Delivering package 9 from location index 19 to 12. Distance: 1.0 miles.
Delivering package 9 from location index 19 to 12. Distance: 1.0 miles.
Delivering package 27 from location index 12 to 1. Distance: 4.8 miles.
Delivering package 27 from location index 12 to 1. Distance: 4.8 miles.
Delivering package 35 from location index 1 to 1. Distance: 0.0 miles.
Delivering package 35 from location index 1 to 1. Distance: 0.0 miles.
Delivering package 32 from location index 1 to 15. Distance: 4.1 miles.
Delivering package 32 from location index 1 to 15. Distance: 4.1 miles.
Delivering package 32 from location index 1 to 15. Distance: 4.5 miles.
Delivering package 17 from location index 15 to 14. Distance: 0.6 miles.
Delivering package 17 from location index 15 to 14. Distance: 0.6 miles.
Delivering package 24 from location index 14 to 22. Distance: 4.6 miles.
Delivering package 24 from location index 14 to 22. Distance: 4.6 miles.
Delivering package 26 from location index 22 to 24. Distance: 1.1 miles.
Delivering package 26 from location index 22 to 24. Distance: 1.1 miles.
Delivering package 22 from location index 24 to 26. Distance: 1.3 miles.
Delivering package 22 from location index 24 to 26. Distance: 1.3 miles.
Delivering package 23 from location index 26 to 25. Distance: 7.8 miles.
Delivering package 23 from location index 26 to 25. Distance: 7.8 miles.
Delivering package 12 from location index 23 to 16. Distance: 7.5 miles.
Delivering package 12 from location index 23 to 16. Distance: 7.5 miles.
Truck 3 returning to hub from location index 16. Distance: 7.6 miles.

Options:
1. Check Package Status
2. Display All Package Statuses
3. Display Truck Information
4. Display Total Mileage
5. Exit

Enter your choice: 3
Truck 1 -> Return Time: 09:41 AM, Total Mileage: 41.19999999999999, Packages: [1, 14, 16, 15, 29, 30, 31, 34, 37, 40, 19, 28, 13, 4, 8, 10]
Truck 2 -> Return Time: 11:08 AM, Total Mileage: 41.9, Packages: [3, 18, 36, 38, 2, 11, 39]
Truck 3 -> Return Time: 12:56 PM, Total Mileage: 51.6, Packages: [9, 32, 28, 5, 7, 12, 17, 21, 22, 23, 24, 26, 27, 33, 35]

Options:
1. Check Package Status
2. Display All Package Statuses
3. Display Truck Information
4. Display Total Mileage
5. Exit

Enter your choice: 5
Exiting program.

○ (base) ericjacobs@Erics-MacBook-Pro C950 %
```

Col 13: Ln 19, Col 63 Spaces: 4 UTF-8 LF CSV □

F1: Strengths of Chosen Algorithm

Simplicity and Speed:

The Nearest Neighbor algorithm's primary strength lies in its simplicity, making it easy to implement and understand. It requires no complex data structures or intricate heuristics, making the development and debugging processes more straightforward. This simplicity also translates to speed, as the algorithm can quickly make delivery decisions, allowing for rapid progression through the delivery route without the need for extensive pre-computation or planning.

Local Optimization and Real-Time Decision Making:

Another significant advantage of the Nearest Neighbor algorithm is its ability to make real-time, locally optimized decisions. By always selecting the closest next destination, it minimizes the travel time between stops, potentially reducing the overall delivery time for packages. This local optimization is particularly effective when deliveries are clustered in certain areas, as it allows the algorithm to minimize travel distance incrementally.

Low Overhead:

The Nearest Neighbor algorithm operates with low overhead, as it does not require storing or managing large sets of possible routes. Instead, it dynamically selects the next step based on current conditions. This is advantageous for systems with limited computational resources or when the delivery parameters can change rapidly, such as in real-time traffic conditions or when dealing with last-minute delivery schedule changes.

In the context of the WGUPS Routing Program, these strengths align with the operational requirements of quick dispatch and the ability to adapt to changes throughout the delivery day. The decision to use the Nearest Neighbor algorithm is also justified by the scale of the operation, where the relatively small number of destinations does not necessitate the overhead of more complex algorithms that may offer global optimization but at the cost of increased complexity and computation time.

F2: Verification of Algorithm

Effective Route Optimization:

The Nearest Neighbor algorithm adeptly aligns with the WGUPS routing program's need for efficient route planning. By prioritizing the closest unvisited locations for each delivery, this algorithm streamlines the delivery process, potentially reducing the overall time taken between consecutive deliveries.

Compatibility with Non-Linear Data Structures:

This algorithm seamlessly integrates with non-linear data structures like hash tables used in the program. The hash table effectively manages package data, including tracking and updating delivery statuses. The Nearest Neighbor algorithm leverages this data to make informed decisions on routing, thereby enhancing its efficiency.

Adherence to Operational Constraints: The Nearest Neighbor algorithm operates within the key constraints of the delivery system. It respects the truck capacity limits, ensuring that each truck's package load remains within the permissible 16-package maximum. Moreover, it aids in adhering to delivery deadlines by selecting the most immediately accessible destinations, thus minimizing the risk of delays.

Simple and Efficient Implementation: The algorithm's simplicity makes it a practical choice for the WGUPS routing program. Its straightforward implementation ensures that the system remains robust and less prone to errors, which is critical in a real-world logistics operation.

Scalability and Flexibility: Despite its simplicity, the Nearest Neighbor algorithm offers a degree of scalability and flexibility. It can adapt to varying numbers of packages and different route configurations, making it a versatile tool for daily delivery operations.

Local Optimization for Quick Decisions: The algorithm's approach to local optimization, focusing on the immediate best choice without overcomplicating the overall route, enables quick and efficient decision-making. This aspect is particularly beneficial in dynamic delivery environments where time efficiency is paramount.

Facilitates Real-Time Adjustments: Given the real-time nature of package delivery, the Nearest Neighbor algorithm is apt as it allows for on-the-go route adjustments. This adaptability is crucial for handling unforeseen changes in delivery schedules or routes.

In summary, the Nearest Neighbor algorithm's use in the WGUPS routing program satisfies the scenario's requirements. Its ability to efficiently plan routes, work in tandem with complex data structures, and operate within the given logistical constraints, all while maintaining simplicity and adaptability, makes it a well-suited choice for the task at hand.

F3: Other Possible Algorithms

Dijkstra's Algorithm:

Dijkstra's algorithm excels in mapping out the shortest path from one origin to various destinations, which is beneficial for optimizing the route for a single truck from the hub to multiple delivery points. Unlike the Nearest Neighbor algorithm, which selects the next closest destination without considering future paths, Dijkstra's algorithm takes a broader view of the network, potentially finding a shorter overall path. This could result in substantial mileage savings, especially in a complex urban delivery network. For the given program, I prioritize simplicity and real-time decision-making, which Nearest Neighbor provides, over the complete route optimization that Dijkstra's algorithm offers.

Dynamic Programming:

Dynamic programming approach would analyze all possible delivery sequences and choose the one with the optimal outcome, considering every package's constraints and delivery windows. This method is substantially different from the Nearest Neighbor algorithm, which does not consider future deliveries when choosing the next destination. Dynamic programming could potentially optimize the entire delivery day, but at the cost of increased initial computation time. Given that the package delivery system requires quick adjustments and real-time tracking, the Nearest Neighbor algorithm's simplicity and speed are more aligned with operational needs than the comprehensive but slower dynamic programming approach.

F3a: Algorithm Differences

Dijkstra's Algorithm:

Dijkstra's Algorithm differs from the Nearest Neighbor approach primarily in its route optimization and computational complexity. Dijkstra's focuses on calculating the shortest possible route from a single point to all others in the network, considering the overall path. This comprehensive view potentially yields a more efficient total route, but at the cost of higher computational demands. Unlike Nearest Neighbor, which selects the next closest point without considering the total journey, Dijkstra's Algorithm ensures a globally optimized path. However, its complexity makes it less adaptable to real-time changes, as any alterations in the delivery plan might necessitate recalculating the entire route.

Dynamic Programming:

Dynamic Programming stands apart from the Nearest Neighbor method in its optimization scope and approach to computational complexity. It breaks down the delivery routing problem into smaller sub-problems, optimizing the entire route by considering all constraints and future decisions. This method potentially offers a more efficient solution for the entire delivery schedule but requires significant computational resources upfront. Unlike the Nearest Neighbor approach, which makes decisions based solely on immediate proximity, Dynamic Programming's holistic approach is less adaptable to last-minute changes, as it involves pre-computing the entire delivery plan and struggles to incorporate real-time adjustments.

G: Different Approach

Enhanced User Interface for Monitoring and Updates:

A more sophisticated user interface would significantly improve the user experience. Implementing a dashboard with real-time tracking, alerts for delivery updates, and interactive maps would allow users and supervisors to monitor progress effectively and make informed decisions.

Integration of Machine Learning for Predictive Analytics:

Machine learning models could be developed to predict traffic patterns, estimate delivery times more accurately, and suggest route alterations in response to unforeseen circumstances, such as road closures or delivery delays.

Use of API Integration for Real-Time Data: Integrating APIs from traffic and weather services could provide real-time data that can dynamically adjust routes to avoid delays, thereby improving punctuality and customer satisfaction.

H: Verification of Data Structure

Efficient Access to Package Data:

The hash table structure in the WGUPS Routing Program excels in providing fast access to package information. Each package is uniquely identified by its ID, serving as the key in the hash table. This structure enables constant-time complexity ($O(1)$) for critical operations like inserting, retrieving, and updating package data, which is pivotal in a dynamic delivery environment.

Flexibility in Data Management: The hash table's design allows for flexibility in handling package data. For instance, when the address for package #9 needed updating at 10:20 AM, the hash table efficiently facilitated this change. This adaptability is essential for real-world scenarios where package information can change due to various factors.

Handling Multiple Package Attributes: The hash table effectively stores and manages various attributes for each package, such as delivery address, deadline, and status. This comprehensive data handling ensures that all necessary details are readily available for each package, enabling informed decision-making during the delivery process.

Compatibility with Routing Algorithm: The integration of the hash table with the Nearest Neighbor algorithm demonstrates a harmonious relationship between data storage and routing logic. The hash table provides the algorithm with quick access to package locations and statuses, aiding in efficient route calculation and package prioritization.

Scalability and Maintenance: The hash table's structure supports scalability, allowing the system to handle a growing number of packages without significant performance degradation. Moreover, the simplicity and clarity of the hash table's implementation contribute to easier maintenance and potential future enhancements of the program.

Resilience to Dynamic Changes: The hash table's ability to quickly adapt to changes, such as updating delivery statuses from 'At the hub' to 'En route' or 'Delivered', showcases its suitability for dynamic operational environments like package delivery, where statuses are frequently updated throughout the day.

In conclusion, the hash table used in the WGUPS Routing Program fulfills all the requirements set out in the scenario. Its efficiency in data retrieval and modification, combined with the ability to handle complex data structures and integrate seamlessly with the routing algorithm, makes it an ideal choice for this application.

H1: Other Data Structures

Graphs:

A graph data structure consists of nodes (vertices) and edges, which naturally represent networks such as city maps and delivery routes, making it an ideal choice for routing problems. In a graph, each delivery location can be a node, with edges representing the direct paths between them, along with the distances as weights. This allows for the use of graph-specific algorithms like Dijkstra's or A* for finding the shortest path, which could potentially be more efficient than the nearest neighbor approach when dealing with complex routing scenarios. Graphs also facilitate the representation of various constraints and conditions of the delivery scenario, such as time windows, package dependencies, and truck capacities. Unlike hash tables, which are excellent for key-value pair storage and retrieval, graphs excel in modeling relational data and spatial structures, providing a more holistic view of the relationships between different delivery points.

Binary Search Trees (BSTs):

A binary search tree is a hierarchical data structure where each node has at most two children, referred to as the left child and the right child. BSTs are sorted in a way that allows for efficient searching, insertion, and deletion operations that, on average, have a time complexity of $O(\log n)$. However, in the worst case, such as when the tree becomes unbalanced, these operations can degrade to $O(n)$. In the context of package delivery, a BST could be used to organize packages by delivery deadlines or other sortable criteria, facilitating an ordered traversal that could assist in prioritizing deliveries. However, unlike hash tables which provide rapid access to package data via direct hashing, BSTs would require traversal from the root to the appropriate node, which can be less efficient for lookups. Balancing a BST to ensure its efficiency, such as in AVL trees or Red-Black trees, adds additional complexity to the implementation, which is not required for hash tables.

H1a: Data Structure Differences

Graphs:

While graphs offer a visual and relational representation of the delivery network, the complexity of graph algorithms and the overhead of maintaining such structures may not be necessary for smaller scale routing problems. In the program, I opted for hash tables over graphs because they provide faster direct access for package data retrieval, which is crucial for real-time updates on package status, and are simpler to implement for the specific use case that does not require visual mapping of routes.

Binary Search Trees (BSTs):

Although BSTs can maintain an ordered structure of packages, their performance can become inefficient in skewed or unbalanced scenarios, requiring rebalancing algorithms to maintain optimal search times. I chose hash tables in the program because they offer consistent average-time complexity for insertions, deletions, and lookups, which is more suitable for the dynamic nature of package tracking and doesn't necessitate the ordered structure that BSTs provide.