

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)

Jessica Short

ID #010500487

WGU Email: jnic331@wgu.edu

12/01/2024

C950 Data Structures and Algorithms II

PART A. Hash Table

Here is a screenshot of the code for the hash table creation:

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

    # Inserts a new item into the hash table and will update an item in the list already
    def insert(self, key, item): # does both insert and update
        bucket = hash(key) % len(self.table)
        bucket_list = self.table[bucket]

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

        # if not, insert the item to the end of the bucket list.
        key_value = [key, item]
        bucket_list.append(key_value)
        return True

    # Searches for an item with matching key in the hash table.
    # Returns the item if found, or None if not found.
    def search(self, key):
        bucket = hash(key) % len(self.table)
        bucket_list = self.table[bucket]
        # print(bucket_list)
        # search key in bucket
        for kv in bucket_list:
            # print(key_value)
            if kv[0] == key:
                return kv[1] # value
        return None

    # Removes an item with matching key from the hash table.
    def remove(self, key):
        # get the bucket list where this item will be removed from.
        bucket = hash(key) % len(self.table)
        bucket_list = self.table[bucket]

        # remove the item from the bucket list if it is present.
        for kv in bucket_list:
            # print (key_value)
            if kv[0] == key:
                bucket_list.remove([kv[0], kv[1]])
```

Here is a screenshot of the code for the function that loads package data from a CSV file and inserts it into a hash table:

```
#This function loads package data from a CSV file and inserts it into a hash table
def loadPackagesFromCSV(file_path): 1 usage
    with open(file_path) as file:
        package_data = csv.reader(file, delimiter=',')
        for data in package_data:
            package_id = int(data[0])
            street_address = data[1]
            city = data[2]
            state = data[3]
            zip_code = data[4]
            deadline = data[5]
            weight = data[6]
            notes = data[7]
            status = "At the Hub"
            departure_time = None
            delivery_time = None

            # Create a new Package instance
            package = Package(package_id, street_address, city, state, zip_code, deadline, weight, notes, status,
                               departure_time, delivery_time)

            # Insert the Package object into the hash table
            packageMap.insert(package_id, package)

# Initialize the hash table and assigns it to the variable packageMap
packageMap = ChainingHashTable()
```

```
# pulls data from CSV into the function
loadPackagesFromCSV('CSV/csvpackage.csv')
```

PART B. Lookup functions

A screenshot of the code for the Package class is shown below:

```
# Define Package class
class Package:
    """usage"""
    def __init__(self, ID, address, city, state, zipcode, deadline, weight, notes, status, departureTime, deliveryTime):
        self.ID = ID
        self.address = address
        self.city = city
        self.state = state
        self.zipcode = zipcode
        self.deadline = deadline
        self.weight = weight
        self.notes = notes
        self.status = status
        self.departureTime = None
        self.deliveryTime = None

# Defines the __str__ method of the Package class for readable output
def __str__(self):
    bold_status_label = f"\033[1mStatus:\033[0m" # Bold the "Status:" label
    bold_status_value = f"\033[1m{self.status}\033[0m" # Bold the status value
    return (
        f"ID: {self.ID}, Address: {self.address}, {self.city}, {self.state}, {self.zipcode}, "
        f"Weight: {self.weight} lbs, "
        f"Departure Time: {self.departureTime or 'Not set'}, "
        f"Deadline: {self.deadline}, "
        f"Delivery Time: {self.deliveryTime or 'Not set'}, | "
        f"{bold_status_label} {bold_status_value}"
    )

# This method updates the status of a Package object based on a given time
def updateStatus(self, userTime):
    if self.deliveryTime is None or userTime < self.departureTime:
        self.status = "At the hub"
    elif userTime < self.deliveryTime:
        self.status = "En route"
    else:
        self.status = "Delivered"

# Handle address update for package 9 after it's received at 10:20AM
if self.ID == 9:
    self._updatePackage9Address(userTime)

def _updatePackage9Address(self, userTime):
    """usage"""
    cutoff_time = datetime.timedelta(hours=10, minutes=20)
    if userTime > cutoff_time:
        self.address = "410 S State St"
        self.zipcode = "84111"
    else:
        self.address = "300 State St"
        self.zipcode = "84103"
```

Here is a screenshot of the code that updates and prints the package status based on the given time:

```
# Function to update and print package status based on the given time
def update_and_print_status(packageID, userTime): 1 usage
    package = packageMap.search(packageID)
    if package:
        package.updateStatus(userTime)
        print(str(package))
    else:
        print(f"No package found with ID {packageID}.")
```

PART C. Original Code

The screenshot below shows the code that reads the CSV files and converts their contents into Python lists:

```
# Read the CSV files and converts their contents into Python lists
with open("CSV/csvaddress.csv") as address_file:
    address_data = csv.reader(address_file)
    address_data = list(address_data)

with open("CSV/csvdistance.csv") as distance_file:
    distance_data = csv.reader(distance_file)
    distance_data = list(distance_data)
```

The screenshot below shows the code that defines the Truck class:

```
# Define Truck class
class Truck: 3usages
    def __init__(self, capacity, speed, mileage, current_location, depart_time, packages):
        self.capacity = capacity
        self.speed = speed
        self.mileage = mileage
        self.current_location = current_location
        self.time = depart_time
        self.depart_time = depart_time
        self.packages = packages

# Defines the _str_ method of the Truck class
def __str__(self):
    return (
        f"Truck Capacity: {self.capacity}, Speed: {self.speed} mph, "
        f"Mileage: {self.mileage} miles, Location: {self.current_location}, "
        f"Time: {self.time}, Departure Time: {self.depart_time}, "
        f"Packages: {' '.join(map(str, self.packages)) if self.packages else 'No packages'}"
    )
```


The screenshots below show the code that calculates the distance between 2 locations. This function is utilized in the nearest neighbor algorithm.

```
# This function searches through the address_data list to find the ID number
# of a given address. It returns the ID as an integer if the address is found.
# The ID number is also the row/column number in the distance matrix distance_data.
def find_address_id(target_address):    4 usages
    for row in address_data:
        if target_address in row[2]:
            return int(row[0])

# The function looks up and returns the distance between two addresses
# based on the distance matrix distance_data.
def calculate_distance(location1, location2):    2 usages
    dist = distance_data[location1][location2]
    if dist == '':
        dist = distance_data[location2][location1]
    return float(dist)
```

The code below loads the trucks and stores the package assignments for later use:

```
# Load the trucks and assign them a departure time
# The special instructions that are listed in the package notes
# have been taken into consideration when assigning packages to a truck
# and choosing a departure time.
truck1 = Truck(capacity: 16, speed: 18, mileage: 0.0, current_location: "4001 South 700 East", datetime.timedelta(hours=8),
               packages: [1, 4, 13, 14, 15, 16, 19, 20, 21, 29, 30, 34, 37, 39, 40])
truck2 = Truck(capacity: 16, speed: 18, mileage: 0.0, current_location: "4001 South 700 East", datetime.timedelta(hours=10, minutes=21),
               packages: [2, 5, 7, 8, 9, 10, 11, 12, 17, 22, 23, 24, 27, 33, 35])
truck3 = Truck(capacity: 16, speed: 18, mileage: 0.0, current_location: "4001 South 700 East", datetime.timedelta(hours=9, minutes=6),
               packages: [3, 6, 18, 25, 26, 28, 31, 32, 36, 38])

# Store the original package assignments for each truck. For use on line 335 for printing to the user interface.
original_truck_packages = {
    "Truck 1": truck1.packages.copy(),
    "Truck 2": truck2.packages.copy(),
    "Truck 3": truck3.packages.copy()
}
```

The implementation of the nearest neighbor algorithm in this application is shown in the screenshot below. The function `deliverTruckPacakges` calls the helper functions `find_next_package` and `update_truck_and_package_status`:

```
# This function simulates the process of delivering packages assigned to a truck
# Uses the nearest neighbor algorithm
def deliverTruckPacakges(truck, priority_packages=None): 3 usages
    # Default priority packages if not provided
    # Packages 25 and 6 are delayed on their flight. They will not arrive at the
    # depot until 9:05am yet they have a 10:30AM deadline.
    priority_packages = priority_packages or [25, 6]

    # Initialize an empty list to store packages to go out for delivery
    packages_in_transit = []

    # Loop through each package ID currently assigned to the truck
    for packageID in truck.packages:
        # Use the packageMap to find the detailed information for the package
        package = packageMap.search(packageID)

        # Add the retrieved package object to the packages_in_transit list
        packages_in_transit.append(package)

    # Clear the truck's package list after adding the packages to packages_in_transit
    truck.packages.clear()

    # Represents a large initial distance to ensure correct comparison
    # The current max distance between 2 locations is 14.2
    LARGE_INITIAL_DISTANCE = 30

    # Process deliveries
    # This while loop will continue to execute as long as the list packages_in_transit is not empty.
    while packages_in_transit:
        # Find the next package to deliver
        next_delivery, delivery_distance = find_next_package(truck, packages_in_transit, priority_packages,
                                                             LARGE_INITIAL_DISTANCE)

        if next_delivery:
            # Update truck's status after delivery
            update_truck_and_package_status(truck, next_delivery, delivery_distance)

            # Remove the package from the list of packages in transit
            packages_in_transit.remove(next_delivery)
```



```

def find_next_package(truck, packages_in_transit, priority_packages, max_distance): 1usage
    # Find the next package to deliver, prioritizing urgent ones and then selecting the nearest.
    # First, try to find the nearest priority package
    for package in packages_in_transit:
        if package.ID in priority_packages:
            distance = calculate_distance(find_address_id(truck.current_location), find_address_id(package.address))
            return package, distance

    # If no priority package found, find the nearest package
    shortest_distance = max_distance
    nearest_package = None
    for package in packages_in_transit:
        distance = calculate_distance(find_address_id(truck.current_location), find_address_id(package.address))
        if distance < shortest_distance:
            shortest_distance = distance
            nearest_package = package

    return nearest_package, shortest_distance

def update_truck_and_package_status(truck, package, distance): 1usage
    # Update truck's mileage, location, and time after delivering a package.
    # Update truck's mileage and location
    truck.mileage += distance
    truck.current_location = package.address

    # Update truck's time based on distance traveled (using truck's speed)
    truck.time += datetime.timedelta(hours=distance / truck.speed)

    # Set package's delivery and departure time
    package.deliveryTime = truck.time
    package.departureTime = truck.depart_time

```

The screenshot below shows the code to deliver the packages for trucks 1, 2, and 3.

```

# Example usage: Deliver packages for multiple trucks
deliverTruckPackages(truck1)
deliverTruckPackages(truck3)

# Truck 2 does not leave until either truck1 or truck3 are finished
truck2.depart_time = min(truck1.time, truck3.time)
deliverTruckPackages(truck2)

```

The code for the user interface is shown in the screenshot below:

```
# Main loop for user interaction
while True:
    print()
    # Display options for package status
    print("1. View the status of a single package at a specific time.")
    print("2. View the status of all packages at a specific time.")
    print("3. Exit.")
    print()
    user_option = input("Choose an option: ")
    print()

    # Option 1: View a specific package's status
    if user_option == "1":
        try:
            userTime = input("Enter a time to check package status. Format is HH:MM 24-Hour: ")
            (h, m) = userTime.split(":")
            userTime = datetime.timedelta(hours=int(h), minutes=int(m))

            packageID = int(input("Enter Package ID: "))
            if packageID <= 0 or packageID > 40:
                print("Invalid Package ID.")
                continue
            update_and_print_status(packageID, userTime)
        except (ValueError, IndexError):
            print("Invalid input. Please check the time or package ID format.")
            continue

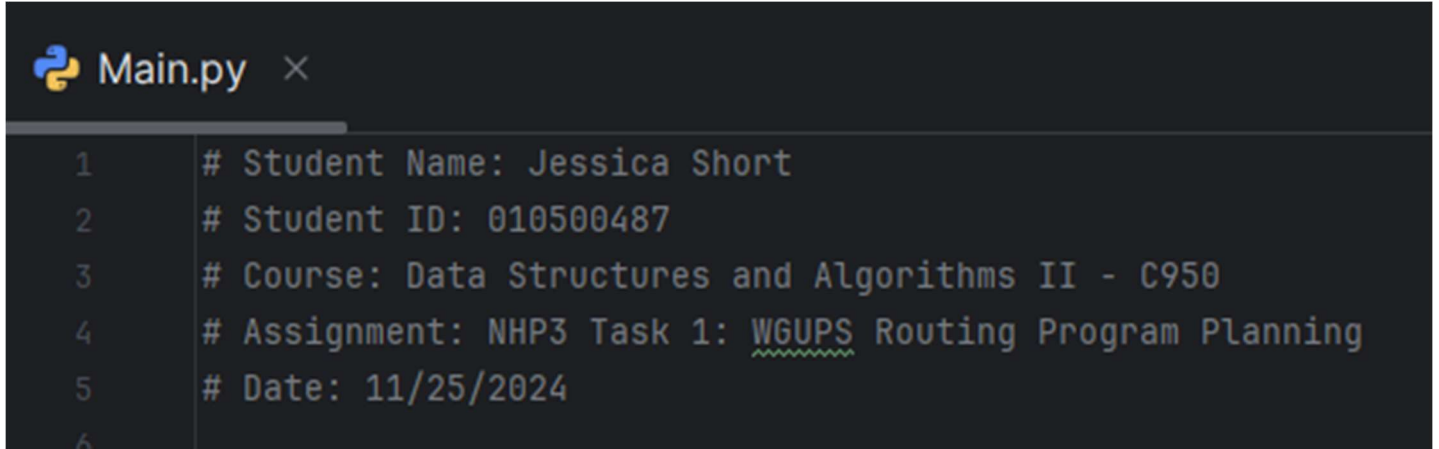
    # Option 2: View the status of all packages at a specific time for all trucks
    elif user_option == "2":
        try:
            userTime = input("Enter a time to check package status. Format is HH:MM 24-Hour: ")
            (h, m) = userTime.split(":")
            userTime = datetime.time(hour=int(h), minute=int(m))
            display_package_status_by_truck(userTime)
        except (ValueError, IndexError):
            print("Invalid time format. Please use HH:MM.")
            continue

    # Option 3: Exit the program
    elif user_option == "3":
        print("Exiting the program.")
        break

    # Invalid option input
    else:
        print("Invalid input. Choose 1, 2, or 3.")
```

PART C1. Identification Information

The screenshot below displays the first lines of the file “Main.py”:

A screenshot of a code editor window. The title bar at the top shows a Python logo icon, the filename "Main.py", and a close button (X). The editor area has a dark background with light-colored text. On the left side, there is a vertical line of numbers from 1 to 6, representing line numbers. The code consists of five lines of comments, each starting with a hash symbol (#). The text of the comments is: "# Student Name: Jessica Short", "# Student ID: 010500487", "# Course: Data Structures and Algorithms II - C950", "# Assignment: NHP3 Task 1: WGUPS Routing Program Planning", and "# Date: 11/25/2024". The word "WGUPS" in the fourth line is underlined with a green wavy line.

```
1 # Student Name: Jessica Short
2 # Student ID: 010500487
3 # Course: Data Structures and Algorithms II - C950
4 # Assignment: NHP3 Task 1: WGUPS Routing Program Planning
5 # Date: 11/25/2024
6
```

PART C2. Process and Flow Comments

Comments are included throughout the application. Comments are marked with a “#” and appear as gray text in PyCharm. The comments describe what each section of code does and the flow of the program. Here is a screenshot of a section of comments from the function `deliverTruckPackages`:

```
# This function simulates the process of delivering packages assigned to a truck
# Uses the nearest neighbor algorithm
def deliverTruckPackages(truck, priority_packages=None): 3 usages
    # Default priority packages if not provided
    # Packages 25 and 6 are delayed on their flight. They will not arrive at the
    # depot until 9:05am yet they have a 10:30AM deadline.
    priority_packages = priority_packages or [25, 6]

    # Initialize an empty list to store packages to go out for delivery
    packages_in_transit = []

    # Loop through each package ID currently assigned to the truck
    for packageID in truck.packages:
        # Use the packageMap to find the detailed information for the package
        package = packageMap.search(packageID)

        # Add the retrieved package object to the packages_in_transit list
        packages_in_transit.append(package)

    # Clear the truck's package list after adding the packages to packages_in_transit
    truck.packages.clear()

    # Represents a large initial distance to ensure correct comparison
    # The current max distance between 2 locations is 14.2
    LARGE_INITIAL_DISTANCE = 30

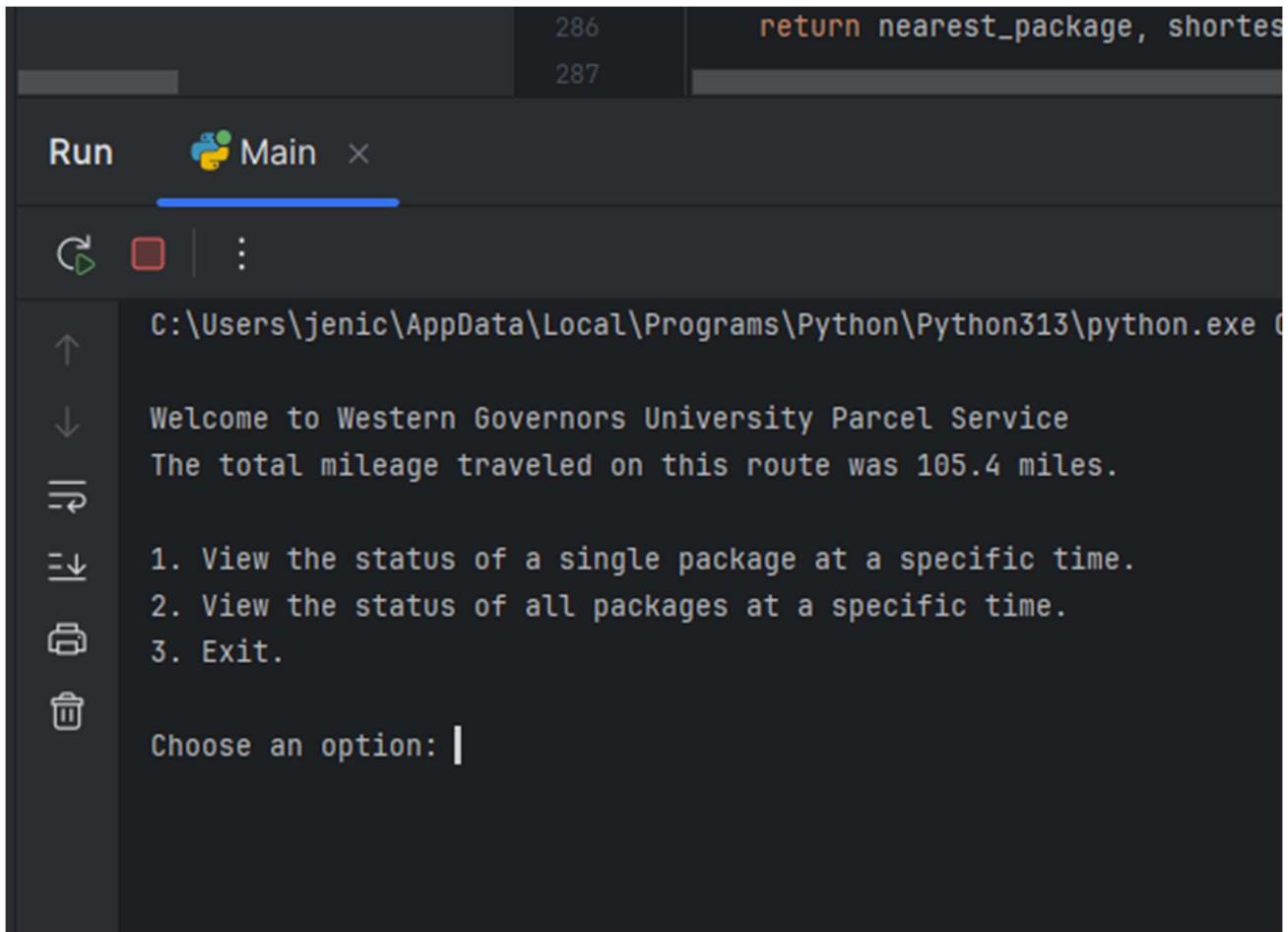
    # Process deliveries
    # This while loop will continue to execute as long as the list packages_in_transit is not empty.
    while packages_in_transit:
        # Find the next package to deliver
        next_delivery, delivery_distance = find_next_package(truck, packages_in_transit, priority_packages,
                                                             LARGE_INITIAL_DISTANCE)

        if next_delivery:
            # Update truck's status after delivery
            update_truck_and_package_status(truck, next_delivery, delivery_distance)

            # Remove the package from the list of packages in transit
            packages_in_transit.remove(next_delivery)
```

PART D. Interface

A screenshot of the user interface is shown below:



The screenshot shows a Python IDE with a dark theme. At the top, there are two tabs: 'Run' and 'Main'. The 'Main' tab is active, showing a Python script. The script contains a function that returns the nearest package and the shortest distance. Below the script, there is a console window showing the output of the program. The output includes a welcome message, the total mileage traveled, and a list of three options: 1. View the status of a single package at a specific time. 2. View the status of all packages at a specific time. 3. Exit. The console prompt is 'Choose an option: |'.

```
286 return nearest_package, shortest_distance
287
Run Main x
C:\Users\jenic\AppData\Local\Programs\Python\Python313\python.exe C:\Users\jenic\AppData\Local\Programs\Python\Python313\python.exe
Welcome to Western Governors University Parcel Service
The total mileage traveled on this route was 105.4 miles.
1. View the status of a single package at a specific time.
2. View the status of all packages at a specific time.
3. Exit.
Choose an option: |
```


PART D1. First Status Check

Requirement: A screenshot to show the status of *all* packages loaded onto *each* truck at a time between 8:35 a.m. and 9:25 a.m.

Chosen Time: 08:50 AM

```
1. View the status of a single package at a specific time.
2. View the status of all packages at a specific time.
3. Exit.

Choose an option: 2

Enter a time to check package status. Format is HH:MM 24-Hour: 08:50

Package status at 08:50:00:

Truck 1:

ID: 1, Address: 195 W Oakland Ave, Salt Lake City, UT, 84115, Weight: 21 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:40:40, | Status: Delivered
ID: 4, Address: 380 W 2880 S, Salt Lake City, UT, 84115, Weight: 4 lbs, Departure Time: 8:00:00, Deadline: EOD, Delivery Time: 8:37:00, | Status: Delivered
ID: 13, Address: 2010 W 500 S, Salt Lake City, UT, 84104, Weight: 2 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 9:21:40, | Status: En route
ID: 14, Address: 4300 S 1300 E, Millcreek, UT, 84117, Weight: 88 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:06:20, | Status: Delivered
ID: 15, Address: 4580 S 2300 E, Holladay, UT, 84117, Weight: 4 lbs, Departure Time: 8:00:00, Deadline: 9:00 AM, Delivery Time: 8:13:00, | Status: Delivered
ID: 16, Address: 4580 S 2300 E, Holladay, UT, 84117, Weight: 88 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:13:00, | Status: Delivered
ID: 19, Address: 177 W Price Ave, Salt Lake City, UT, 84115, Weight: 37 lbs, Departure Time: 8:00:00, Deadline: EOD, Delivery Time: 8:31:20, | Status: Delivered
ID: 20, Address: 3595 Main St, Salt Lake City, UT, 84115, Weight: 37 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:29:40, | Status: Delivered
ID: 21, Address: 3595 Main St, Salt Lake City, UT, 84115, Weight: 3 lbs, Departure Time: 8:00:00, Deadline: EOD, Delivery Time: 8:29:40, | Status: Delivered
ID: 29, Address: 1330 2100 S, Salt Lake City, UT, 84106, Weight: 2 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:50:00, | Status: Delivered
ID: 30, Address: 300 State St, Salt Lake City, UT, 84103, Weight: 1 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 9:07:40, | Status: En route
ID: 34, Address: 4580 S 2300 E, Holladay, UT, 84117, Weight: 2 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:13:00, | Status: Delivered
ID: 37, Address: 410 S State St, Salt Lake City, UT, 84111, Weight: 2 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 9:04:20, | Status: En route
ID: 39, Address: 2010 W 500 S, Salt Lake City, UT, 84104, Weight: 9 lbs, Departure Time: 8:00:00, Deadline: EOD, Delivery Time: 9:21:40, | Status: En route
ID: 40, Address: 380 W 2880 S, Salt Lake City, UT, 84115, Weight: 45 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:37:00, | Status: Delivered

Truck 2:

ID: 2, Address: 2530 S 500 E, Salt Lake City, UT, 84106, Weight: 44 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 10:59:20, | Status: At the hub
ID: 5, Address: 410 S State St, Salt Lake City, UT, 84111, Weight: 5 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:20:00, | Status: At the hub
ID: 7, Address: 1330 2100 S, Salt Lake City, UT, 84106, Weight: 8 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:04:40, | Status: At the hub
ID: 8, Address: 300 State St, Salt Lake City, UT, 84103, Weight: 9 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:23:20, | Status: At the hub
ID: 9, Address: 300 State St, Salt Lake City, UT, 84103, Weight: 2 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:23:20, | Status: At the hub
ID: 10, Address: 600 E 900 South, Salt Lake City, UT, 84105, Weight: 1 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:14:00, | Status: At the hub
ID: 11, Address: 2600 Taylorsville Blvd, Salt Lake City, UT, 84118, Weight: 1 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 12:12:00, | Status: At the hub
ID: 12, Address: 3575 W Valley Central Station bus Loop, West Valley City, UT, 84119, Weight: 1 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 12:39:00, | Status: At the hub
ID: 17, Address: 3148 S 1100 W, Salt Lake City, UT, 84119, Weight: 2 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:54:40, | Status: At the hub
ID: 22, Address: 6351 South 900 East, Murray, UT, 84121, Weight: 2 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 10:39:20, | Status: At the hub
ID: 23, Address: 5100 South 2700 West, Salt Lake City, UT, 84118, Weight: 5 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 12:10:40, | Status: At the hub
ID: 24, Address: 5025 State St, Murray, UT, 84107, Weight: 7 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 10:29:00, | Status: At the hub
ID: 27, Address: 1060 Dalton Ave S, Salt Lake City, UT, 84104, Weight: 5 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:39:20, | Status: At the hub
ID: 33, Address: 2530 S 500 E, Salt Lake City, UT, 84106, Weight: 1 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 10:59:20, | Status: At the hub
ID: 35, Address: 1060 Dalton Ave S, Salt Lake City, UT, 84104, Weight: 88 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:39:20, | Status: At the hub

Truck 3:

ID: 3, Address: 233 Canyon Rd, Salt Lake City, UT, 84103, Weight: 2 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 11:17:40, | Status: At the hub
ID: 6, Address: 3060 Lester St, West Valley City, UT, 84119, Weight: 88 lbs, Departure Time: 9:06:00, Deadline: 10:30 AM, Delivery Time: 9:23:20, | Status: At the hub
ID: 18, Address: 1488 4800 S, Salt Lake City, UT, 84123, Weight: 6 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 10:39:00, | Status: At the hub
ID: 25, Address: 5383 South 900 East #104, Salt Lake City, UT, 84117, Weight: 7 lbs, Departure Time: 9:06:00, Deadline: 10:30 AM, Delivery Time: 9:47:20, | Status: At the hub
ID: 26, Address: 5383 South 900 East #104, Salt Lake City, UT, 84117, Weight: 25 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 9:47:20, | Status: At the hub
ID: 28, Address: 2835 Main St, Salt Lake City, UT, 84115, Weight: 7 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 10:04:40, | Status: At the hub
ID: 31, Address: 3365 S 900 W, Salt Lake City, UT, 84119, Weight: 1 lbs, Departure Time: 9:06:00, Deadline: 10:30 AM, Delivery Time: 10:14:20, | Status: At the hub
ID: 32, Address: 3365 S 900 W, Salt Lake City, UT, 84119, Weight: 1 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 10:14:20, | Status: At the hub
ID: 36, Address: 2300 Parkway Blvd, West Valley City, UT, 84119, Weight: 88 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 10:25:40, | Status: At the hub
ID: 38, Address: 410 S State St, Salt Lake City, UT, 84111, Weight: 9 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 11:14:20, | Status: At the hub

1. View the status of a single package at a specific time.
2. View the status of all packages at a specific time.
3. Exit.

Choose an option:
```


PART D2. Second Status Check

Requirement: A screenshot to show the status of *all* packages loaded onto *each* truck at a time between 9:35 a.m. and 10:25 a.m.

Chosen Time: 09:45 AM

```
Welcome to Western Governors University Parcel Service
The total mileage traveled on this route was 105.4 miles.

1. View the status of a single package at a specific time.
2. View the status of all packages at a specific time.
3. Exit.

Choose an option: 2

Enter a time to check package status. Format is HH:MM 24-Hour: 9:45

Package status at 09:45:00:

Truck 1:

ID: 1, Address: 195 W Oakland Ave, Salt Lake City, UT, 84115, Weight: 21 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:40:40, | Status: Delivered
ID: 4, Address: 380 W 2880 S, Salt Lake City, UT, 84115, Weight: 4 lbs, Departure Time: 8:00:00, Deadline: EOD, Delivery Time: 8:37:00, | Status: Delivered
ID: 13, Address: 2010 W 500 S, Salt Lake City, UT, 84104, Weight: 2 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 9:21:40, | Status: Delivered
ID: 14, Address: 4300 S 1300 E, Millcreek, UT, 84117, Weight: 88 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:06:20, | Status: Delivered
ID: 15, Address: 4580 S 2300 E, Holladay, UT, 84117, Weight: 4 lbs, Departure Time: 8:00:00, Deadline: 9:00 AM, Delivery Time: 8:13:00, | Status: Delivered
ID: 16, Address: 4580 S 2300 E, Holladay, UT, 84117, Weight: 88 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:13:00, | Status: Delivered
ID: 19, Address: 177 W Price Ave, Salt Lake City, UT, 84115, Weight: 37 lbs, Departure Time: 8:00:00, Deadline: EOD, Delivery Time: 8:31:20, | Status: Delivered
ID: 20, Address: 3595 Main St, Salt Lake City, UT, 84115, Weight: 37 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:29:40, | Status: Delivered
ID: 21, Address: 3595 Main St, Salt Lake City, UT, 84115, Weight: 3 lbs, Departure Time: 8:00:00, Deadline: EOD, Delivery Time: 8:29:40, | Status: Delivered
ID: 29, Address: 1330 2100 S, Salt Lake City, UT, 84106, Weight: 2 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:50:00, | Status: Delivered
ID: 30, Address: 300 State St, Salt Lake City, UT, 84103, Weight: 1 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 9:07:40, | Status: Delivered
ID: 34, Address: 4580 S 2300 E, Holladay, UT, 84117, Weight: 2 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:13:00, | Status: Delivered
ID: 37, Address: 410 S State St, Salt Lake City, UT, 84111, Weight: 2 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 9:04:20, | Status: Delivered
ID: 39, Address: 2010 W 500 S, Salt Lake City, UT, 84104, Weight: 9 lbs, Departure Time: 8:00:00, Deadline: EOD, Delivery Time: 9:21:40, | Status: Delivered
ID: 40, Address: 380 W 2880 S, Salt Lake City, UT, 84115, Weight: 45 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:37:00, | Status: Delivered

Truck 2:

ID: 2, Address: 2530 S 500 E, Salt Lake City, UT, 84106, Weight: 44 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 10:59:20, | Status: En route
ID: 5, Address: 410 S State St, Salt Lake City, UT, 84111, Weight: 5 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:20:00, | Status: En route
ID: 7, Address: 1330 2100 S, Salt Lake City, UT, 84106, Weight: 8 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:04:40, | Status: En route
ID: 8, Address: 300 State St, Salt Lake City, UT, 84103, Weight: 9 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:23:20, | Status: En route
ID: 9, Address: 300 State St, Salt Lake City, UT, 84103, Weight: 2 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:23:20, | Status: En route
ID: 10, Address: 600 E 900 South, Salt Lake City, UT, 84105, Weight: 1 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:14:00, | Status: En route
ID: 11, Address: 2600 Taylorsville Blvd, Salt Lake City, UT, 84118, Weight: 1 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 12:12:00, | Status: En route
ID: 12, Address: 3575 W Valley Central Station bus Loop, West Valley City, UT, 84119, Weight: 1 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 12:39:00, | Status: En route
ID: 17, Address: 3148 S 1100 W, Salt Lake City, UT, 84119, Weight: 2 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:54:40, | Status: En route
ID: 22, Address: 6351 South 900 East, Murray, UT, 84121, Weight: 2 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 10:39:20, | Status: En route
ID: 23, Address: 5100 South 2700 West, Salt Lake City, UT, 84118, Weight: 5 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 12:10:40, | Status: En route
ID: 24, Address: 5025 State St, Murray, UT, 84107, Weight: 7 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 10:29:00, | Status: En route
ID: 27, Address: 1060 Dalton Ave S, Salt Lake City, UT, 84104, Weight: 5 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:39:20, | Status: En route
ID: 33, Address: 2530 S 500 E, Salt Lake City, UT, 84106, Weight: 1 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 10:59:20, | Status: En route
ID: 35, Address: 1060 Dalton Ave S, Salt Lake City, UT, 84104, Weight: 88 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:39:20, | Status: En route

Truck 3:

ID: 3, Address: 233 Canyon Rd, Salt Lake City, UT, 84103, Weight: 2 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 11:17:40, | Status: En route
ID: 6, Address: 3060 Lester St, West Valley City, UT, 84119, Weight: 88 lbs, Departure Time: 9:06:00, Deadline: 10:30 AM, Delivery Time: 9:23:20, | Status: Delivered
ID: 18, Address: 1488 4800 S, Salt Lake City, UT, 84123, Weight: 6 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 10:39:00, | Status: En route
ID: 25, Address: 5383 South 900 East #104, Salt Lake City, UT, 84117, Weight: 7 lbs, Departure Time: 9:06:00, Deadline: 10:30 AM, Delivery Time: 9:47:20, | Status: En route
ID: 26, Address: 5383 South 900 East #104, Salt Lake City, UT, 84117, Weight: 25 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 9:47:20, | Status: En route
ID: 28, Address: 2835 Main St, Salt Lake City, UT, 84115, Weight: 7 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 10:04:40, | Status: En route
ID: 31, Address: 3365 S 900 W, Salt Lake City, UT, 84119, Weight: 1 lbs, Departure Time: 9:06:00, Deadline: 10:30 AM, Delivery Time: 10:14:20, | Status: En route
ID: 32, Address: 3365 S 900 W, Salt Lake City, UT, 84119, Weight: 1 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 10:14:20, | Status: En route
ID: 36, Address: 2300 Parkway Blvd, West Valley City, UT, 84119, Weight: 88 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 10:25:40, | Status: En route
ID: 38, Address: 410 S State St, Salt Lake City, UT, 84111, Weight: 9 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 11:14:20, | Status: En route

1. View the status of a single package at a specific time.
2. View the status of all packages at a specific time.
3. Exit.

Choose an option: |
```

PART D3. Third Status Check

Requirement: A screenshot to show the status of *all* packages loaded onto *each* truck at a time between 12:03 p.m. and 1:12 p.m.

Chosen Time: 01:05 PM (13:05 24-hr time)

1. View the status of a single package at a specific time.
2. View the status of all packages at a specific time.
3. Exit.

Choose an option: 2

Enter a time to check package status. Format is HH:MM 24-Hour: 13:05

Package status at 13:05:00:

Truck 1:

ID: 1, Address: 195 W Oakland Ave, Salt Lake City, UT, 84115, Weight: 21 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:40:40, | Status: Delivered
ID: 4, Address: 380 W 2880 S, Salt Lake City, UT, 84115, Weight: 4 lbs, Departure Time: 8:00:00, Deadline: EOD, Delivery Time: 8:37:00, | Status: Delivered
ID: 13, Address: 2810 W 500 S, Salt Lake City, UT, 84104, Weight: 2 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 9:21:40, | Status: Delivered
ID: 14, Address: 4300 S 1300 E, Millcreek, UT, 84117, Weight: 88 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:06:20, | Status: Delivered
ID: 15, Address: 4580 S 2300 E, Holladay, UT, 84117, Weight: 4 lbs, Departure Time: 8:00:00, Deadline: 9:00 AM, Delivery Time: 8:13:00, | Status: Delivered
ID: 16, Address: 4580 S 2300 E, Holladay, UT, 84117, Weight: 88 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:13:00, | Status: Delivered
ID: 19, Address: 177 W Price Ave, Salt Lake City, UT, 84115, Weight: 37 lbs, Departure Time: 8:00:00, Deadline: EOD, Delivery Time: 8:31:20, | Status: Delivered
ID: 20, Address: 3595 Main St, Salt Lake City, UT, 84115, Weight: 37 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:29:40, | Status: Delivered
ID: 21, Address: 3595 Main St, Salt Lake City, UT, 84115, Weight: 3 lbs, Departure Time: 8:00:00, Deadline: EOD, Delivery Time: 8:29:40, | Status: Delivered
ID: 29, Address: 1330 2100 S, Salt Lake City, UT, 84106, Weight: 2 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:50:00, | Status: Delivered
ID: 30, Address: 300 State St, Salt Lake City, UT, 84103, Weight: 1 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 9:07:40, | Status: Delivered
ID: 34, Address: 4580 S 2300 E, Holladay, UT, 84117, Weight: 2 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:13:00, | Status: Delivered
ID: 37, Address: 410 S State St, Salt Lake City, UT, 84111, Weight: 2 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 9:04:20, | Status: Delivered
ID: 39, Address: 2910 W 500 S, Salt Lake City, UT, 84104, Weight: 9 lbs, Departure Time: 8:00:00, Deadline: EOD, Delivery Time: 9:21:40, | Status: Delivered
ID: 40, Address: 380 W 2880 S, Salt Lake City, UT, 84115, Weight: 45 lbs, Departure Time: 8:00:00, Deadline: 10:30 AM, Delivery Time: 8:37:00, | Status: Delivered

Truck 2:

ID: 2, Address: 2530 S 500 E, Salt Lake City, UT, 84106, Weight: 44 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 10:59:20, | Status: Delivered
ID: 5, Address: 410 S State St, Salt Lake City, UT, 84111, Weight: 5 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:20:00, | Status: Delivered
ID: 7, Address: 1330 2100 S, Salt Lake City, UT, 84106, Weight: 8 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:04:40, | Status: Delivered
ID: 8, Address: 300 State St, Salt Lake City, UT, 84103, Weight: 9 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:23:20, | Status: Delivered
ID: 9, Address: 410 S State St, Salt Lake City, UT, 84111, Weight: 2 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:23:20, | Status: Delivered
ID: 10, Address: 600 E 900 South, Salt Lake City, UT, 84105, Weight: 1 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:14:00, | Status: Delivered
ID: 11, Address: 2600 Taylorsville Blvd, Salt Lake City, UT, 84118, Weight: 1 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 12:12:00, | Status: Delivered
ID: 12, Address: 3575 W Valley Central Station bus Loop, West Valley City, UT, 84119, Weight: 1 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 12:39:00, | Status: Delivered
ID: 17, Address: 3148 S 1100 W, Salt Lake City, UT, 84119, Weight: 2 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:54:40, | Status: Delivered
ID: 22, Address: 6351 South 900 East, Murray, UT, 84121, Weight: 2 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 10:39:20, | Status: Delivered
ID: 23, Address: 5100 South 2700 West, Salt Lake City, UT, 84118, Weight: 5 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 12:10:40, | Status: Delivered
ID: 24, Address: 5025 State St, Murray, UT, 84107, Weight: 7 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 10:29:00, | Status: Delivered
ID: 27, Address: 1040 Dalton Ave S, Salt Lake City, UT, 84104, Weight: 5 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:39:20, | Status: Delivered
ID: 33, Address: 2530 S 500 E, Salt Lake City, UT, 84106, Weight: 1 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 10:59:20, | Status: Delivered
ID: 35, Address: 1040 Dalton Ave S, Salt Lake City, UT, 84104, Weight: 88 lbs, Departure Time: 9:21:40, Deadline: EOD, Delivery Time: 11:39:20, | Status: Delivered

Truck 3:

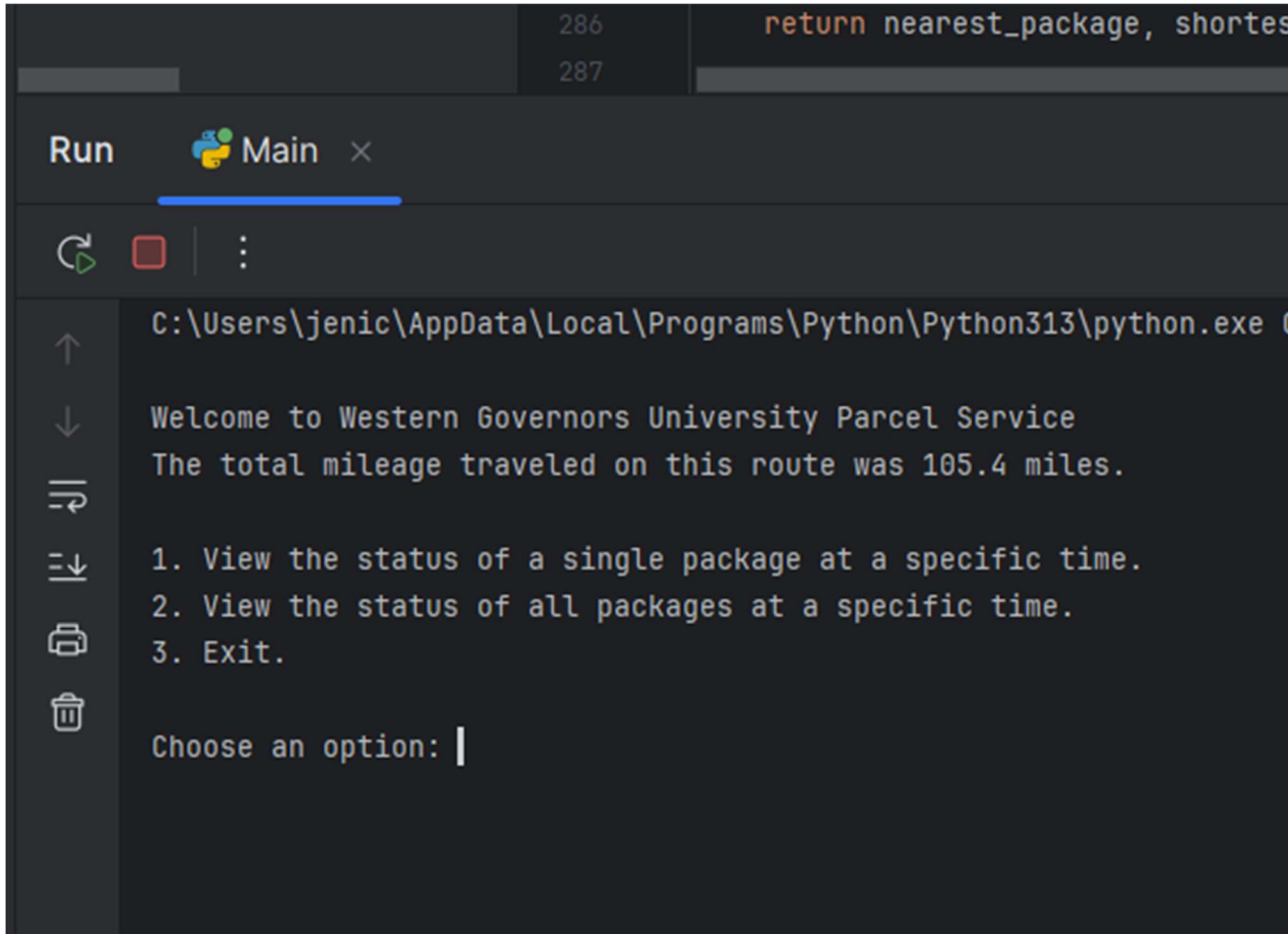
ID: 3, Address: 233 Canyon Rd, Salt Lake City, UT, 84103, Weight: 2 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 11:17:40, | Status: Delivered
ID: 6, Address: 3040 Lester St, West Valley City, UT, 84119, Weight: 88 lbs, Departure Time: 9:06:00, Deadline: 10:30 AM, Delivery Time: 9:23:20, | Status: Delivered
ID: 18, Address: 1488 4800 S, Salt Lake City, UT, 84123, Weight: 6 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 10:39:00, | Status: Delivered
ID: 25, Address: 5383 South 900 East #104, Salt Lake City, UT, 84117, Weight: 7 lbs, Departure Time: 9:06:00, Deadline: 10:30 AM, Delivery Time: 9:47:20, | Status: Delivered
ID: 26, Address: 5383 South 900 East #104, Salt Lake City, UT, 84117, Weight: 25 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 9:47:20, | Status: Delivered
ID: 28, Address: 2835 Main St, Salt Lake City, UT, 84115, Weight: 7 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 10:04:40, | Status: Delivered
ID: 31, Address: 3365 S 900 W, Salt Lake City, UT, 84119, Weight: 1 lbs, Departure Time: 9:06:00, Deadline: 10:30 AM, Delivery Time: 10:14:20, | Status: Delivered
ID: 32, Address: 3365 S 900 W, Salt Lake City, UT, 84119, Weight: 1 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 10:14:20, | Status: Delivered
ID: 36, Address: 2300 Parkway Blvd, West Valley City, UT, 84119, Weight: 88 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 10:25:40, | Status: Delivered
ID: 38, Address: 410 S State St, Salt Lake City, UT, 84111, Weight: 9 lbs, Departure Time: 9:06:00, Deadline: EOD, Delivery Time: 11:14:20, | Status: Delivered

1. View the status of a single package at a specific time.
2. View the status of all packages at a specific time.
3. Exit.

Choose an option: |

PART E. Screenshot of Code Execution

A screenshot is shown below that shows that the application ran successfully without errors in PyCharm. The total mileage traveled by all trucks is 105.4 miles.



The screenshot displays the PyCharm Run console for a Python application. The console window is titled "Run" and "Main". The output shows the execution path, a welcome message, the total mileage, a menu of options, and a prompt for user input.

```
C:\Users\jenic\AppData\Local\Programs\Python\Python313\python.exe C:\Users\jenic\AppData\Local\Programs\Python\Python313\python.exe  
Welcome to Western Governors University Parcel Service  
The total mileage traveled on this route was 105.4 miles.  
1. View the status of a single package at a specific time.  
2. View the status of all packages at a specific time.  
3. Exit.  
Choose an option: |
```

PART F1. Strengths of the Chosen Algorithm

The Nearest Neighbor algorithm is an excellent fit for this project because of its ability to make dynamic decisions efficiently. For instance, when package 9's address changes at 10:20 AM, the algorithm seamlessly incorporates this update without requiring a complete recalculation of the route. Its step-by-step approach ensures it adapts effortlessly to common logistical challenges, such as package delays, address modifications, and urgent delivery priorities.

Another key strength of Nearest Neighbor (NN) is its simplicity. It is easy to implement, understand, debug, and maintain, even for novice programmers. Unlike more complex methods like

Dijkstra's algorithm, NN relies on basic programming constructs such as loops and conditional checks. This simplicity not only lowers the barrier to implementation but also makes the algorithm intuitive, as it mirrors the natural decision-making process of a delivery driver selecting the next closest location to visit.

PART F2. Verification of Algorithm

The Nearest Neighbor algorithm successfully meets all the requirements outlined in the scenario. All packages were delivered before their respective deadlines, demonstrating the algorithm's effectiveness in handling time-sensitive deliveries. The total distance traveled by all trucks was 105.4 miles, well within the maximum limit of 140 miles. Additionally, the algorithm seamlessly updated the address for package 9 at 10:20 AM. Despite their delayed arrival at the hub at 9:05 AM, packages 6 and 25 were delivered on time, meeting their 10:30 AM deadline.

PART F3. Other Possible Algorithms

Two other algorithms that would also meet all the requirements outlined in the scenario are Dijkstra's Algorithm and the Genetic Algorithm.

PART F3A. Algorithm Differences

Dijkstra's Algorithm finds the shortest route from the starting point to all destination points by evaluating the minimum distance at each step. It starts by assigning a distance of 0 to the starting location and marking all other locations with an infinite distance. The algorithm then examines the nearest unvisited location, updates the shortest possible distances to its neighboring locations, and continues this process. Once all locations are visited, it reconstructs the optimal path by following the shortest distances it has calculated.

Dijkstra's Algorithm (DA) calculates the shortest overall path from the starting point to all other points in the network, whereas the Nearest Neighbor (NN) method decides the next stop based solely on the current location, without considering the entire route. As a result, DA is generally better at minimizing the total distance traveled compared to NN. However, DA requires recalculating the entire

route in real-time when updates occur, such as address changes or flight delays at the hub. Additionally, Dijkstra's Algorithm is more computationally intensive than Nearest Neighbor because it evaluates all possible routes to find the optimal one (Source: itsadityash for Geeks for Geeks, 2024).

The Genetic Algorithm (GA) solves problems by mimicking evolution. In this method, multiple potential truck routes are generated and evaluated using a fitness function, which measures the total distance traveled by all trucks. Shorter routes result in higher fitness scores. Over time, the best routes are "bred" to create new, improved routes (called offspring). This process is repeated over many generations, gradually yielding more optimal solutions.

GA evaluates a wide range of potential routes simultaneously, while NN builds a route incrementally, one step at a time. GA is more likely to identify the most efficient route because it refines and improves multiple solutions over several generations. However, this comes at a higher computational cost, as GA processes a population of routes and performs numerous iterations involving crossover, mutation, and fitness evaluation. GA is especially well-suited for large, complex datasets when sufficient computational resources are available. In contrast, NN is simpler, faster, and easier to implement, making it a practical choice for smaller or less complex problems, though it may result in less efficient routes. (Source: chitrangkumishra for Geeks for Geeks, 2024).

PART G. Different Approach

If I were to do this project again, I would load the trucks from the CSV file instead of loading them manually. I consider all requirements when coding this section, including hub arrival delays, delivery deadlines, and other specific package instructions. This approach would make the application more efficient and scalable. If additional packages were added to the package CSV file, the application would automatically load the trucks, saving time and reducing the need for manual updates.

PART H. Verification of Data Structure

A hash table was used in this project because it meets all the requirements outlined in the scenario. It is used to store and manage the packages, and it provides an efficient way to look up packages based on their unique ID number. This is accomplished through the `search()` method in the `ChainingHashTable` class. When the package data is loaded from the CSV file, each `Package` object

is stored in the hash table, with the package's ID as the key and the Package object itself as the value. The Package object contains important details, including the address, city, state, zip code, deadline, weight, notes, status, departure time, and delivery time.

PART H1. Other Data Structures

Besides a hash table, two other data structures that would meet the requirements for this project are a Binary Search Tree and a Priority Queue.

PART H1A. Data Structure Differences

A Binary Search Tree (BST) is a tree-like hierarchical data structure where each node holds a key and has at most 2 children. In this structure, the key of the left child node is less than the key of its parent node, and the key of the right child node is greater. This arrangement results in an ordered system that supports efficient searching, insertion, and deletion operations.

A hash table offers an average time complexity of $O(1)$ for search, insertion, and deletion, which generally results in faster performance compared to a binary search tree (BST). A balanced BST, on the other hand, has an average time complexity of $O(\log N)$ for these operations. In the worst case, if the BST becomes unbalanced, the time complexity can degrade to $O(N)$. Additionally, a BST typically uses less memory than a hash table, as it only requires space for the nodes and links between them. In contrast, a hash table needs extra space to handle collisions, which can increase memory usage, especially when the table is sparsely populated or has a high load factor. (Source: Programiz, 2024).

A Priority Queue (PQ) is a data structure where each item has a priority value. Items with higher priority are processed before those with lower priority. For this project, packages with earlier and shorter deadlines would be delivered first. The queue would sort the packages so that the most urgent ones come first.

In a priority queue, the time complexity for insertion and removal is $O(\log N)$, which is generally slower than a hash table, which has an average time complexity of $O(1)$ for the same operations (for search, insertion, and deletion). A Priority Queue is not optimized for searching for a package by ID. Items in a priority queue are organized based on their priority, not their IDs, unlike in a hash table. As

a result, searching for a package by ID would require scanning through all the elements, leading to a worst-case time complexity of $O(N)$. (Source: Ghosh, 2024).

PART I. Sources

Zybook C950: Data Structures and Algorithms II

Figure 7.8.2: Hash table using chaining

Authors: Roman Lysecky and Frank Vahid

Date: June 2018

Date Accessed: 11/29/2024

Webpage: <https://learn.zybooks.com/zybook/WGUC950Template2023>

Traveling Salsman Problem using Genetic Algorithm

Author: chitrankmishra for Geeks for Geeks

Date: 04/30/2024

Date Accessed: 12/1/2024

Webpage: <https://www.geeksforgeeks.org/traveling-salesman-problem-using-genetic-algorithm/>

What is Dijkstra's Algorithm? | Introduction to Dijkstra's Shortest Path Algorithm

Author: itsadityash for Geeks for Geeks

Date: 05/09/2024

Date Accessed: 12/1/2024

Webpage: <https://www.geeksforgeeks.org/introduction-to-dijkstras-shortest-path-algorithm/>

Binary Search Tree (BST)

Author: Programiz

Date Accessed: 12/1/2024

Webpage: <https://www.programiz.com/dsa/binary-search-tree>

What is a Priority Queue Data Structure? Implementation, Type & Many More

Author: Amit Kumar Ghosh for Scholarhat

Date Accessed: 12/1/2024

Webpage: <https://www.scholarhat.com/tutorial/datastructures/priority-queue-in-data-structures>