

WGU Parcel Service

Routing Program Implementation

Katrina Stradford

ID #011334472

kstrad7@wgu.edu

12/21/2023

C950 Data Structures and Algorithms II

Table of Contents

A. HASH TABLE	5
B. LOOK-UP FUNCTION	6
C. ORIGINAL CODE	7
C1. Identification	9
C2. Process And Flow Comments	10
D. INTERFACE	11
D1. FIRST STATUS CHECK	11
D2. SECOND STATUS CHECK	12
D3. THIRD STATUS CHECK	13
E. CODE EXECUTION	14
F. THE 3-OPT ALGORITHM	15
F1. Strengths of the 3-Opt Algorithm	15
F2. 3-Opt Algorithm Verification	16
F3. Other Possible Algorithms	17
F3a. Algorithm Differences	18
G. DIFFERENT APPROACH	18
H. VERIFICATION OF CHAINING HASH TABLE	19
H1. OTHER DATA STRUCTURES	20
H1A. DATA STRUCTURE DIFFERENCES	21
I. Sources	21

A. HASH TABLE

```

class PackageHash:
    """
    Class that constructs a hash table to store the package objects.
    """
    load_factor = 1.5
    num_packages = 0

    def __init__(self):
        """
        Initialize the hash table with 40 empty bucket lists. - O(1)
        """
        self.size = 40
        self.table = [[] for _ in range(self.size)]

    def get_hash(self, package_id):
        """
        Get the hash of a package using the package ID key. - O(1)
        """
        index = hash(package_id) % self.size
        bucket = self.table[index]

        return bucket

    def insert(self, package_id, package_data):
        """
        Insert or update a package using the package ID key and the package data list as the value. - O(n)
        """
        bucket = self.get_hash(package_id)
        p = [package_id, package_data]

        if bucket in self.table:
            for package in bucket:
                if package[0] == package_id:
                    package[1] = package_data
                    return True

            bucket.append(p)
            return True

        # track number of entries for rehashing at 150% capacity
        self.num_packages += 1
        load = self.num_packages / self.size
        if load > self.load_factor:
            self.resize()

    def lookup(self, package_id):
        """
        Look up a package using the package ID. - O(n)
        """
        bucket = self.get_hash(package_id)
        if bucket in self.table:
            for package in bucket:
                if package[0] == package_id:
                    return package[1]
            else:
                raise LookupError(f"There is no record of Package {package_id}")
        else:
            raise LookupError("Something went wrong with the lookup function. Try again.")

    def resize(self):
        """
        Resize the hash table to fit the package size and copy over the existing values. - O(n^2)
        """
        # Copy existing packages into a temporary list.
        temp_packages = []
        for package in self.table:
            temp_packages.append(package)

        # Create a new empty hash table and double the size
        self.__init__()
        self.size = self.size * 2
        self.table = [[] for _ in range(self.size)]

        # Insert the packages from the temporary list into the new hash table.
        for package_id, package_data in temp_packages:
            self.insert(package_id, package_data)

```

The hash table is constructed in the `package.py` file and includes the `insert()` function, which takes the package ID as the key and then inserts the `package_data` list as the value.

C950 Task-2 WGUPS Routing Program

```
def insert(self, package_id, package_data):  
    """  
    Insert or update a package using the package ID key and the package data list as the value. - O(n)  
    """  
    bucket = self.get_hash(package_id)  
    p = [package_id, package_data]  
  
    if bucket in self.table:  
        for package in bucket:  
            if package[0] == package_id:  
                package[1] = package_data  
                return True  
  
        bucket.append(p)  
        return True  
  
    # track number of entries for rehashing at 150% capacity  
    self.num_packages += 1  
    load = self.num_packages / self.size  
    if load > self.load_factor:  
        self.resize()
```

insertion function

```
dldPackages = PackageHash() # Create an instance of the package hash table.
```

```
# Create Package object for each row and insert it into the hash table instance.  
p = Package(package_id, address, city, state, zip_code, deadline, weight, notes, truck)  
dldPackages.insert(package_id, p)
```

The package_data list consists of the package ID, address, city, state, zip code, weight, deadline, notes, status, and delivery time

B. LOOK-UP FUNCTION

```
def lookup(self, package_id):  
    """  
    Look up a package using the package ID. - O(n)  
    """  
    bucket = self.get_hash(package_id)  
    if bucket in self.table:  
        for package in bucket:  
            if package[0] == package_id:  
                return package[1]  
        else:  
            raise LookupError(f"There is no record of Package {package_id}")  
    else:  
        raise LookupError("Something went wrong with the lookup function. Try again.")
```

lookup function implementation



```
main.py ×  
12  
13     for i in range(1, 6):  
14         print(package.dldPackages.lookup(i))  
15
```

calling the lookup function to return the packages with package IDs 1-5

C950 Task-2 WGUPS Routing Program

```
Stack:
  <module>, main.py:14
1, 195 W Oakland Ave, Salt Lake City, UT, 84115, 21, 1900-01-01 10:30:00 , at the hub, 1900-01-01 09:12:20
Stack:
  <module>, main.py:14
2, 2530 S 500 E, Salt Lake City, UT, 84106, 44, 1900-01-01 16:59:59 , at the hub, 1900-01-01 10:30:20
Stack:
  <module>, main.py:14
3, 233 Canyon Rd, Salt Lake City, UT, 84103, 2, 1900-01-01 16:59:59 Can only be on truck 2, at the hub, 1900-01-01 11:01:20
Stack:
  <module>, main.py:14
4, 380 W 2880 S, Salt Lake City, UT, 84115, 4, 1900-01-01 16:59:59 , at the hub, 1900-01-01 09:00:40
Stack:
  <module>, main.py:14
5, 410 S State St, Salt Lake City, UT, 84111, 5, 1900-01-01 16:59:59 , at the hub, 1900-01-01 10:58:00
```

output stack from the call to the lookup function above

C. ORIGINAL CODE

```
# Sort each package in the sort list into a load list via the sorting criteria.
for parcel in sort_list:
    if parcel.notes:

        # If the notes specify placement on a specific truck, add the package_id to the corresponding load list.
        if 'on truck' in parcel.notes:
            the_truck = parcel.notes.split()[-1]
            loads[int(the_truck)].append(parcel.package_id)
            loaded_list.append(parcel)
            continue

        # If the address will be updated later, add the package to the Truck 3 load list.
        elif 'Wrong address' in parcel.notes:
            loads[3].append(parcel.package_id)
            loaded_list.append(parcel)
            continue

        # If the package must be delivered with other packages, add all related package IDs to the list for Truck 1.
        elif 'delivered with' in parcel.notes:
            pointer = parcel.notes.find('with') + 5
            group = parcel.notes[pointer:].split(',')
            buddies.add(parcel.package_id)
            loaded_list.append(parcel)
            if parcel.package_id not in loads[1]:
                loads[1].append(parcel.package_id)
                for buddy in group:
                    buddies.add(int(buddy))
                    for p in sort_copy:
                        if int(buddy) == p.package_id and p.package_id not in loads[1]:
                            loads[1].append(p.package_id)
                            loaded_list.append(p)
```

A greedy algorithm sorts packages based on notes, delivery deadline, and location

```
# Create three truck objects.
trucks = [Truck( truck_id: 1, departure_time: '08:00:00', driver: 1),
          Truck( truck_id: 2, departure_time: '09:06:00', driver: 2),
          Truck( truck_id: 3, departure_time: '10:21:00', driver: 1)]
```

```
# Add package IDs to the corresponding truck's package list. - O(n)
for load in loads[1]:
    trucks[0].packages.append(package.dIdPackages.lookup(load))
for load in loads[2]:
    trucks[1].packages.append(package.dIdPackages.lookup(load))
for load in loads[3]:
    trucks[2].packages.append(package.dIdPackages.lookup(load))
```

Truck objects are created within a Truck class and loaded with the sorted packages.

C950 Task-2 WGUPS Routing Program

```
final_distances = [inf, inf, inf]
final_routes = [[], [], []]

# Pass trucks through the 3 Opt algorithm 100 times to find the best route and shortest overall distances. - O(n^2)
for _ in range(100):
    for i, the_truck in enumerate(truck.trucks):
        final_routes[i] = three_opt(the_truck, distances, places)
        final_distances[i] = update_route(the_truck, final_routes[i], final_distances[i], distances)

# Verify that all packages will arrive by their deadline or swap packages between trucks until they will. - O(n^2)
for i in range(len(truck.trucks)):
    for j in range(i + 1, len(truck.trucks)):
        truck1 = truck.trucks[i]
        truck2 = truck.trucks[j]

        if not on_time(truck1, distances, places) or not on_time(truck2, distances, places):
            swap = swap_packages(truck1, truck2, distances, places)
```

```
# Iterate through the route assigning three variables to neighboring locations. - O(n^3)
for i in range(1, len(the_truck.packages) - 3):
    for j in range(i + 1, len(the_truck.packages) - 2):
        for k in range(j + 1, len(the_truck.packages) - 1):

            # create a new_route by swapping the first and second variables, then swapping the third variable
            # with the location following it. - O(1)
            new_route = (locations[:i] + locations[i:j + 1][::-1] + locations[j + 1:k + 1][::-1] +
                        locations[k + 1:])

            # Calculate the distance from the WGU Hub to the first route location. - O(1)
            start_of_best_route = [start_location, best_route[0]]
            start_of_new_route = [start_location, new_route[0]]

            # Calculate the distance of the route. - O(1)
            best_starting_distance = distance.get_distance(start_of_best_route, distances)
            new_starting_distance = distance.get_distance(start_of_new_route, distances)

            if (distance.get_distance(new_route, distances) + new_starting_distance) < (
                distance.get_distance(best_route, distances) + best_starting_distance):
                best_route = new_route
                improved = True
```

The 3-opt algorithm optimizes the route, and new routes are checked for compliance with requirements before being fully accepted.

C1. Identification

```
main.py
1 # created by K. Stradford | Student #011334472
```

My student ID number is included in the first line of main.py.

C2. Process And Flow Comments

C950 Task-2 WGUPS Routing Program

```
def swap_packages(truck1, truck2, distances, places):  
    """  
    Swap two packages where at least one is not on time. Confirm swap if packages are now on time or reverse if the  
    packages are still not on time. -  $O(n^2)$   
  
    :param truck1: truck with the first package  
    :param truck2: truck with the second package  
    :param distances: imported distances between each location  
    :param places: imported locations for each delivery  
    :return: True if swap is successful, False otherwise  
    """
```

comment block explaining the swap_packages function process

```
# Prepare to swap packages if they don't have notes with special parameters. -  $O(n^2)$   
for parcel1 in truck1.packages:  
    for parcel2 in truck2.packages:  
        if not parcel1.notes and not parcel2.notes:  
            current_route1 = truck1.route.copy()  
            current_route2 = truck2.route.copy()  
  
            # Remove each package from its truck, then swap trucks and update package's truck attribute.  
            truck1.packages.remove(parcel1)  
            truck2.packages.remove(parcel2)  
            truck1.packages.append(parcel2)  
            truck2.packages.append(parcel1)  
            parcel2.truck = 1  
            parcel1.truck = 2  
  
            # Run new truck routes through 3-opt algorithm for the best route.  
            truck1.route = three_opt(truck1, distances, places)  
            truck2.route = three_opt(truck2, distances, places)  
  
            # Confirm that new routes are on time and exit.  
            if on_time(truck1, distances, places) and on_time(truck2, distances, places):  
                return True  
  
            # Revert changes to each package if new routes are not on time.
```

in-line comments explaining the flow of the swap_packages function

D. INTERFACE

```
----- WGU Postal Service -----
Daily Local Delivery Monitoring Service

CHOOSE A TIME OPTION:

1: Get Realtime Updates
2: Get Updates at a Specific Time

Enter a menu option number and press enter: 1

The current time is 08:58 AM.

1 - View All Packages
2 - Look Up a Package
3 - See Truck Summary
4 - Exit the Program

Enter Your Selection: 2
Enter a valid Package ID#: 29

-----
                PACKAGE QUERY
            query time: 08:58 AM
-----

Package ID: 29
Address: 1330 2100 S
City: Salt Lake City
Zip Code: 84106
Weight(kg): 2
Delivery Deadline: 10:30 AM
Notes:
Truck: 1
Status: delivered

Delivered At: 08:54 AM

CURRENT CUMULATIVE TRUCK MILES: 61.5

Enter 0 to return to the main menu.
Enter any other key to exit.
```

*interface demonstration using the current time to query
a package for its delivery status and the total truck
mileage*

```
----- WGU Postal Service -----
Daily Local Delivery Monitoring Service

CHOOSE A TIME OPTION:

1: Get Realtime Updates
2: Get Updates at a Specific Time

Enter a menu option number and press enter: 2

Enter the Query Time (HH:MM am/pm): 9:13 am
You entered 09:13 AM.

REPORT OPTIONS:
1 - View All Packages
2 - Look Up a Package
3 - See Truck Summary
4 - Exit the Program

Enter Your Selection: 2
Enter a valid Package ID#: 29

-----
                PACKAGE QUERY
            query time: 09:13 AM
-----

Package ID: 29
Address: 1330 2100 S
City: Salt Lake City
Zip Code: 84106
Weight(kg): 2
Delivery Deadline: 10:30 AM
Notes:
Truck: 1
Status: delivered

Delivered At: 08:54 AM

CURRENT CUMULATIVE TRUCK MILES: 77.1
```

*interface demonstration using a user-defined time to
query a package for its delivery status and the total
truck mileage*

C950 Task-2 WGUPS Routing Program

D1. First Status Check

DAILY PACKAGES SUMMARY						
query time: 08:47 AM						

NOTE: Projected future delivery times are indicated with ~ for undelivered packages.						
ID	DELIVERY ADDRESS	DEADLINE	NOTES	STATUS	TRUCK	DELIVERY DETAILS
1	195 W Oakland Ave	10:30 AM		delivered	1	08:11 AM
2	2530 S 500 E	04:59 PM		at the hub	3	~ 11:42 AM
3	233 Canyon Rd	04:59 PM	Can only be on truck 2	at the hub	2	~ 10:52 AM
4	380 W 2880 S	04:59 PM		delivered	1	08:15 AM
5	410 S State St	04:59 PM		at the hub	2	~ 10:55 AM
6	3060 Lester St	10:30 AM	Delayed on flight	at the hub	2	~ 10:12 AM
7	1330 2100 S	04:59 PM		at the hub	3	~ 11:47 AM
8	300 State St	04:59 PM		at the hub	3	~ 12:06 PM
9	410 S State St	04:59 PM	Wrong address listed	at the hub	3	~ 12:03 PM
10	600 E 900 South	04:59 PM		at the hub	3	~ 11:57 AM
11	2600 Taylorsville Blvd	04:59 PM		at the hub	3	~ 10:57 AM
12	3575 W Valley Central Stat	04:59 PM		at the hub	2	~ 10:28 AM
13	2010 W 500 S	10:30 AM		en route	1	~ 09:25 AM
14	4300 S 1300 E	10:30 AM	Must be delivered with 15, 19	delivered	1	08:30 AM
15	4580 S 2300 E	09:00 AM		delivered	1	08:37 AM
16	4580 S 2300 E	10:30 AM	Must be delivered with 13, 19	delivered	1	08:37 AM
17	3148 S 1100 W	04:59 PM		at the hub	2	~ 09:52 AM
18	1488 4800 S	04:59 PM	Can only be on truck 2	at the hub	2	~ 09:35 AM
19	177 W Price Ave	04:59 PM		en route	1	~ 09:52 AM
20	3595 Main St	10:30 AM	Must be delivered with 13, 15	en route	1	~ 09:50 AM
21	3595 Main St	04:59 PM		en route	1	~ 09:50 AM
22	6351 South 900 East	04:59 PM		at the hub	3	~ 10:33 AM
23	5100 South 2700 West	04:59 PM		at the hub	3	~ 10:58 AM
24	5025 State St	04:59 PM		at the hub	3	~ 10:43 AM
25	5383 South 900 East #104	10:30 AM	Delayed on flight	at the hub	2	~ 09:14 AM
26	5383 South 900 East #104	04:59 PM		at the hub	2	~ 09:14 AM
27	1060 Dalton Ave S	04:59 PM		at the hub	3	~ 11:21 AM
28	2835 Main St	04:59 PM	Delayed on flight	at the hub	2	~ 10:00 AM
29	1330 2100 S	10:30 AM		en route	1	~ 08:54 AM
30	300 State St	10:30 AM		en route	1	~ 09:11 AM
31	3365 S 900 W	10:30 AM		en route	1	~ 09:45 AM
32	3365 S 900 W	04:59 PM	Delayed on flight	at the hub	2	~ 09:50 AM
33	2530 S 500 E	04:59 PM		at the hub	3	~ 11:42 AM
34	4580 S 2300 E	10:30 AM		delivered	1	08:37 AM
35	1060 Dalton Ave S	04:59 PM		at the hub	3	~ 11:21 AM
36	2300 Parkway Blvd	04:59 PM	Can only be on truck 2	at the hub	2	~ 10:17 AM
37	410 S State St	10:30 AM		en route	1	~ 09:08 AM
38	410 S State St	04:59 PM	Can only be on truck 2	at the hub	2	~ 10:55 AM
39	2010 W 500 S	04:59 PM		en route	1	~ 09:25 AM
40	380 W 2880 S	10:30 AM		delivered	1	08:15 AM

summary of all package statuses at 8:47 am

D2. Second Status Check

C950 Task-2 WGUPS Routing Program

DAILY PACKAGES SUMMARY						
query time: 10:12 AM						

NOTE: Projected future delivery times are indicated with ~ for undelivered packages.						
ID	DELIVERY ADDRESS	DEADLINE	NOTES	STATUS	TRUCK	DELIVERY DETAILS
1	195 W Oakland Ave	10:30 AM		delivered	1	08:11 AM
2	2530 S 500 E	04:59 PM		at the hub	3	~ 11:42 AM
3	233 Canyon Rd	04:59 PM	Can only be on truck 2	en route	2	~ 10:52 AM
4	380 W 2880 S	04:59 PM		delivered	1	08:15 AM
5	410 S State St	04:59 PM		en route	2	~ 10:55 AM
6	3060 Lester St	10:30 AM	Delayed on flight	en route	2	~ 10:12 AM
7	1330 2100 S	04:59 PM		at the hub	3	~ 11:47 AM
8	300 State St	04:59 PM		at the hub	3	~ 12:06 PM
9	410 S State St	04:59 PM	Wrong address listed	at the hub	3	~ 12:03 PM
10	600 E 900 South	04:59 PM		at the hub	3	~ 11:57 AM
11	2600 Taylorsville Blvd	04:59 PM		at the hub	3	~ 10:57 AM
12	3575 W Valley Central Stat	04:59 PM		en route	2	~ 10:28 AM
13	2010 W 500 S	10:30 AM		delivered	1	09:25 AM
14	4300 S 1300 E	10:30 AM	Must be delivered with 15, 19	delivered	1	08:30 AM
15	4500 S 2300 E	09:00 AM		delivered	1	08:37 AM
16	4500 S 2300 E	10:30 AM	Must be delivered with 13, 19	delivered	1	08:37 AM
17	3148 S 1100 W	04:59 PM		delivered	2	09:52 AM
18	1488 4800 S	04:59 PM	Can only be on truck 2	delivered	2	09:35 AM
19	177 W Price Ave	04:59 PM		delivered	1	09:52 AM
20	3595 Main St	10:30 AM	Must be delivered with 13, 15	delivered	1	09:50 AM
21	3595 Main St	04:59 PM		delivered	1	09:50 AM
22	6351 South 900 East	04:59 PM		at the hub	3	~ 10:33 AM
23	5100 South 2700 West	04:59 PM		at the hub	3	~ 10:58 AM
24	5025 State St	04:59 PM		at the hub	3	~ 10:43 AM
25	5383 South 900 East #104	10:30 AM	Delayed on flight	delivered	2	09:14 AM
26	5383 South 900 East #104	04:59 PM		delivered	2	09:14 AM
27	1060 Dalton Ave S	04:59 PM		at the hub	3	~ 11:21 AM
28	2835 Main St	04:59 PM	Delayed on flight	delivered	2	10:00 AM
29	1330 2100 S	10:30 AM		delivered	1	08:54 AM
30	300 State St	10:30 AM		delivered	1	09:11 AM
31	3365 S 900 W	10:30 AM		delivered	1	09:45 AM
32	3365 S 900 W	04:59 PM	Delayed on flight	delivered	2	09:50 AM
33	2530 S 500 E	04:59 PM		at the hub	3	~ 11:42 AM
34	4500 S 2300 E	10:30 AM		delivered	1	08:37 AM
35	1060 Dalton Ave S	04:59 PM		at the hub	3	~ 11:21 AM
36	2300 Parkway Blvd	04:59 PM	Can only be on truck 2	en route	2	~ 10:17 AM
37	410 S State St	10:30 AM		delivered	1	09:08 AM
38	410 S State St	04:59 PM	Can only be on truck 2	en route	2	~ 10:55 AM
39	2010 W 500 S	04:59 PM		delivered	1	09:25 AM
40	380 W 2880 S	10:30 AM		delivered	1	08:15 AM

summary of all package statuses at 10:12 am

D3. Third Status Check

C950 Task-2 WGUPS Routing Program

DAILY PACKAGES SUMMARY						
query time: 12:20 PM						
NOTE: Projected future delivery times are indicated with ~ for undelivered packages.						
ID	DELIVERY ADDRESS	DEADLINE	NOTES	STATUS	TRUCK	DELIVERY DETAILS
1	195 W OakLand Ave	10:30 AM		delivered	1	08:11 AM
2	2530 S 500 E	04:59 PM		delivered	3	11:42 AM
3	233 Canyon Rd	04:59 PM	Can only be on truck 2	delivered	2	10:52 AM
4	380 W 2880 S	04:59 PM		delivered	1	08:15 AM
5	410 S State St	04:59 PM		delivered	2	10:55 AM
6	3060 Lester St	10:30 AM	Delayed on flight	delivered	2	10:12 AM
7	1330 2100 S	04:59 PM		delivered	3	11:47 AM
8	300 State St	04:59 PM		delivered	3	12:06 PM
9	410 S State St	04:59 PM	Wrong address listed	delivered	3	12:03 PM
10	600 E 900 South	04:59 PM		delivered	3	11:57 AM
11	2600 Taylorsville Blvd	04:59 PM		delivered	3	10:57 AM
12	3575 W Valley Central Stat	04:59 PM		delivered	2	10:28 AM
13	2010 W 500 S	10:30 AM		delivered	1	09:25 AM
14	4300 S 1300 E	10:30 AM	Must be delivered with 15, 19	delivered	1	08:30 AM
15	4500 S 2300 E	09:00 AM		delivered	1	08:37 AM
16	4500 S 2300 E	10:30 AM	Must be delivered with 13, 19	delivered	1	08:37 AM
17	3148 S 1100 W	04:59 PM		delivered	2	09:52 AM
18	1488 4800 S	04:59 PM	Can only be on truck 2	delivered	2	09:35 AM
19	177 W Price Ave	04:59 PM		delivered	1	09:52 AM
20	3595 Main St	10:30 AM	Must be delivered with 13, 15	delivered	1	09:50 AM
21	3595 Main St	04:59 PM		delivered	1	09:50 AM
22	6351 South 900 East	04:59 PM		delivered	3	10:33 AM
23	5100 South 2700 West	04:59 PM		delivered	3	10:58 AM
24	5025 State St	04:59 PM		delivered	3	10:43 AM
25	5383 South 900 East #104	10:30 AM	Delayed on flight	delivered	2	09:14 AM
26	5383 South 900 East #104	04:59 PM		delivered	2	09:14 AM
27	1060 Dalton Ave S	04:59 PM		delivered	3	11:21 AM
28	2835 Main St	04:59 PM	Delayed on flight	delivered	2	10:00 AM
29	1330 2100 S	10:30 AM		delivered	1	08:54 AM
30	300 State St	10:30 AM		delivered	1	09:11 AM
31	3365 S 900 W	10:30 AM		delivered	1	09:45 AM
32	3365 S 900 W	04:59 PM	Delayed on flight	delivered	2	09:50 AM
33	2530 S 500 E	04:59 PM		delivered	3	11:42 AM
34	4500 S 2300 E	10:30 AM		delivered	1	08:37 AM
35	1060 Dalton Ave S	04:59 PM		delivered	3	11:21 AM
36	2300 Parkway Blvd	04:59 PM	Can only be on truck 2	delivered	2	10:17 AM
37	410 S State St	10:30 AM		delivered	1	09:08 AM
38	410 S State St	04:59 PM	Can only be on truck 2	delivered	2	10:55 AM
39	2010 W 500 S	04:59 PM		delivered	1	09:25 AM
40	380 W 2880 S	10:30 AM		delivered	1	08:15 AM

summary of all package statuses at 12:20 pm

E. CODE EXECUTION

C950 Task-2 WGUPS *Routing Program*

```
----- DAILY TRUCKS SUMMARY -----
07:00 PM SNAPSHOT

----- TRUCK 1 -----
DAILY INFORMATION
Departure Time: 08:00 AM
Driver: 1
Today's Packages: 1, 13, 14, 15, 16, 19, 20, 21, 29, 30, 31, 34, 37, 39, 4, 40

CURRENT PROGRESS
Most Recent Location: 4001 South 700 East
Miles Traveled so Far: 35.8
ROUTE COMPLETED

----- TRUCK 2 -----
DAILY INFORMATION
Departure Time: 09:06 AM
Driver: 2
Today's Packages: 12, 17, 18, 25, 26, 28, 3, 32, 36, 38, 5, 6

CURRENT PROGRESS
Most Recent Location: 410 S State St
Miles Traveled so Far: 32.8
ROUTE COMPLETED

----- TRUCK 3 -----
DAILY INFORMATION
Departure Time: 10:21 AM
Driver: 1
Today's Packages: 10, 11, 2, 22, 23, 24, 27, 33, 35, 7, 8, 9

CURRENT PROGRESS
Most Recent Location: 300 State St
Miles Traveled so Far: 31.6
ROUTE COMPLETED

TOTAL MILES TRAVELED BY 07:00 PM: 100.2

Enter 0 to return to the main menu.
Enter any other key to exit.
0

See you next time.

Process finished with exit code 0
```

successful code run that includes the total truck mileage with no errors or warnings

F. THE 3-OPT ALGORITHM

F1. Strengths of the 3-Opt Algorithm

C950 Task-2 WGUPS Routing Program

The 3-opt algorithm has several strengths in solving traveling salesman problems:


Maintainable

The 3-opt algorithm is common, and documentation on the algorithm is plentiful. This makes it easier for newer programmers to implement the algorithm and increases the maintainability of solutions using the algorithm.

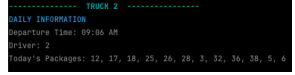



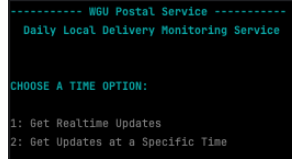
Efficient

The complexity of the 3-opt algorithm is $O(n^3)$, which qualifies as efficient. Efficiency is important when creating systems that require updates and real-time queries. This efficiency is an important consideration for the WGUPS program since the WGUPS supervisor must refresh the DLD packages and query the system daily.

F2. Verification of the 3-Opt Algorithm

REQUIREMENT	VERIFICATION STEPS	ALGORITHM IMPACT	FINAL RESULT
Travel Under 140 Miles	<ol style="list-style-type: none">1. Run the program2. Enter "2" and input a time at or after 5:00 pm3. Enter "3" to generate the truck summary4. Review the total miles for all three trucks at the end of the day at the bottom of the report <p>This data is available for review in Section E of this document.</p>	The 3-opt algorithm loops to find increasingly shorter routes, which helps minimize the overall miles traveled.	 <p>100.2 Miles Traveled</p> <p><i>Since the 3-opt algorithm always begins by randomizing the route, it will always return a new final route. While testing, the results consistently stayed under 140, ranging from the 80s to the 120s.</i></p>
Deliver All Packages By Their Deadline	<ol style="list-style-type: none">1. Run the program2. Enter "2" and input any time at or after 5:00 pm3. Enter "1" to generate the package summary4. Review the deadline and delivery details columns to ensure the package was delivered at or before the deadline. <p><i>This data is available for review in Section D3 of this document.</i></p>	The shorter routes discovered by the algorithm also allow the packages to arrive sooner since there isn't as much distance to cover while the trucks are assumed to move at an average of 18 mph. Additionally, while still in the 3-opt loop, the program shifts into additional function stacks that verify that each package will meet its deadline before fully accepting a new route.	<p>The latest delivery for any package with a 9 a.m. deadline was 8:37 a.m. This includes Package 15.</p> <p>The latest delivery for any package with a 10:30 a.m. deadline was 10:12 a.m. This includes packages 1, 6, 13, 14, 16, 20, 25, 29, 30, 31, 34, 37, and 40.</p> <p>The rest of the packages had an EOD deadline, which wasn't specified. I set the EOD to 5:00 p.m., and the last</p>

C950 Task-2 WGUPS Routing Program

			<p>package of the day was delivered at 12:06 p.m.</p> <p>There are package summary pictures in sections D1, D2, and D3.</p>
<p>Meet All Package Delivery Constraints</p>	<p>Specified Truck Constraints</p> <ol style="list-style-type: none"> 1. Run the program and use the interface to generate the truck summary at any query time. 2. Confirm that the package is listed in the truck's packages. <p>Delayed on Flight Constraints</p> <ol style="list-style-type: none"> 1. Run the program and generate the truck summary at any query time. 2. Find the package ID within the truck package lists. Check the truck's departure time and confirm that it is at or after the end of the package's delay period. <p>Wrong Address Constraints:</p> <ol style="list-style-type: none"> 1. Run the program and generate a query for the package in question. 2. Verify that the correct address is present. <p>Group Delivery Constraints:</p> <ol style="list-style-type: none"> 1. Run the program and generate the truck summary at any query time. 2. Confirm that all packages being delivered together are shown within one truck's package list. 	<p>Once sorted, packages with delivery constraints should not be swapped to different trucks to remedy late delivery times. The program maintains the integrity of the mandatory sorts by avoiding using the package swap feature on these packages. This means the 3-opt algorithm must find optimal routes while maintaining these specific packages as constants within each route.</p>	 <p>Packages 3, 18, 36, and 38 were loaded onto truck 2.</p>  <p>Package 9's address was updated to 410 S. State St. before delivery.</p>  <p>Packages 13, 14, 15, 16, 19, and 20 were delivered together on truck 1.</p>  <p>Packages 6, 25, 28, and 32 were delayed until 9:05 and left on truck 2 at 9:06.</p>
<p>Show Progress of Trucks and Packages at Assigned Points</p>	<ol style="list-style-type: none"> 1. Run the program. 2. Use the interface to see the desired progress. 	<p>The 3-opt algorithm provides the routes that are used to complete the deliveries, which are being queried through the interface.</p>	 <p>A functional user interface is available.</p>

F3. Other Possible Algorithms **HOW THEY MEET REQUIREMENTS?**

The WGUPS Routing Program must include a routing algorithm that will try to find the shortest route for package delivery. There are many known algorithms for this type of problem. Two other algorithms that I considered were the nearest neighbor algorithm and the Christofides algorithm.

Nearest Neighbor Algorithm

The nearest neighbor algorithm attempts to find the shortest path by finding the shortest path from the initial hub and traveling there, then finding the shortest path from the current location and traveling until all locations are visited.

Christofides Algorithm

The Christofides algorithm begins by creating a minimum spanning tree of the locations. This is done by connecting two locations with the shortest distance, then picking the smallest available distance that includes a connected location until all the locations are connected. Once the minimum spanning tree is complete, locations with an odd number of connections are connected to create the route. Then any repeated visits to locations are skipped.

F3A. ALGORITHM DIFFERENCES

The main difference between the nearest neighbor algorithm and the 3-opt algorithm is that the nearest neighbor algorithm is a greedy algorithm that focuses on finding the next shortest distance from each location, while the 3-opt algorithm focuses on searching for the overall shortest distance by swapping out portions of the route. The main difference between the Christofides algorithm and the 3-opt algorithm is that the Christofides algorithm focuses on constructing the route by selecting locations in a calculated order, while the 3-opt algorithm accepts a route and looks for an optimization of it. The 3-opt algorithm is a better overall fit for the routing program's requirements due to two key differences:

Optimization

The 3-opt, nearest neighbor, and Christofides algorithms are all heuristic algorithms, meaning that an approximate solution is found rather than the exact shortest route. Because of this, the routing algorithm chosen needed the ability to check for more optimal approximations after the original route was created.

The 3-opt algorithm produces a different result each time it is performed because its first step is to randomize the route it is given. This variety is not something offered by the nearest neighbor or Christofides algorithms. With the nearest neighbor algorithm, it will always start at the beginning and find the next nearest location. While there could be some flexibility for nearest neighbor solutions where there is a flexible start location, this was not the case for the WGUPS Routing Program. With the Christofides algorithm, there is a similar constancy due to the minimum spanning tree always starting with the smallest distance and finding the next smallest distance in every subsequent operation.

Flexibility

The 3-opt algorithm belongs to a group of similar pair-wise exchange algorithms that include algorithms like the 2-opt and Lin-Kernighan algorithms. Switching between these algorithms would only require changing a few lines of code and would abide by the same logic. There aren't any neighboring algorithms like this available for the nearest neighbor algorithm or Christofides algorithm that don't require an additional line of logic or a new data structure.

G. DIFFERENT APPROACH

The final WGUPS solution met all requirements, with the three trucks traveling 85-127 miles and completing the delivery of all packages between 11:45 a.m. and 2:45 p.m. If another solution was built, the main priority would be finding a way to create more consistency in the final results in these areas. A few suggestions on how this could improve are below:

Sorting Algorithm

The current solution implements a greedy algorithm to parse through the package list and access each package based on varied criteria, assigning the packages to the load list associated with the first criteria met. An alternative option would be to create an enumeration to hold the three trucks and add all three trucks to every package. Then, ineligible trucks could be removed, and all eligible trucks could remain. This would allow more flexibility in truck placement for many packages that were placed in a truck load without a mandatory need to be on that truck. This would also simplify the swapping process that occurs when a package is projected to be delivered after its promised delivery time.

Dispatch Timing

Presently, the first two trucks are dispatched at their earliest available departure times, 8:00 a.m. and 9:06 a.m. In another version of the solution it may be beneficial to first load the trucks based on pre-set address groups, then determine what time each truck should leave based on the promised delivery time of its packages.


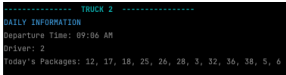
Mileage Consistency

The 3-opt algorithm and package swap function introduce variety in the results that the solution is producing, and the range that the results fall in is pretty large, at a 40-mile discrepancy. We know the algorithm is capable of finding solutions under 100 miles and in some cases as low as the mid-80s, so we could add a rejection filter that will call the 3-opt function again if the resulting route is over a certain mileage. This would look similar in implementation to the `on_time` function.


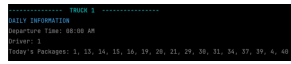
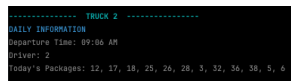
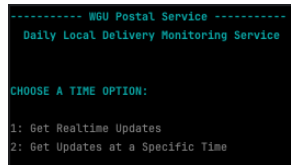
H. VERIFICATION OF THE CHAINING HASH TABLE

REQUIREMENT	VERIFICATION STEPS	HASH TABLE IMPACT	FINAL RESULT
Hash Table with Lookup and Insertion is Present	Implementation 1. Open <code>package.py</code> 2. Confirm the presence of the hash table with a lookup function and an insertion function in class <code>PackageHash</code> Functionality 1. Run the program 2. Print any of the available	The hash table is present and functional.	Implementation The hash table, lookup function, and insertion function are implemented in class <code>PackageHash</code> in the <code>package.py</code> file. This implementation is shown in sections A and B of this document.

C950 Task-2 WGUPS Routing Program

	reports from the user interface. All report information is at some point populated from the package hash table.		<p>Functionality</p> <p>The hash table, its insertion function, and its lookup function are fully functional.</p> <p>The reports that verify this are shown in sections D-E of this document.</p>
Travel Under 140 Miles	<ol style="list-style-type: none"> 1. Run the program 2. Enter "2" and input a time at or after 5:00 pm 3. Enter "3" to generate the truck summary 4. Review the total miles for all three trucks at the end of the day at the bottom of the report <p>This data is available for review in Section E of this document.</p>	The hash table stores the package details, including addresses. The addresses are matched to truck route locations once a route under 140 miles is found and confirmed.	 <p>100.2 Miles Traveled</p> <p><i>Since the 3-opt algorithm always begins by randomizing the route, it will always return a new final route. While testing, the results consistently stayed under 140, ranging from the 80s to the 120s.</i></p>
Deliver All Packages By Their Deadline	<ol style="list-style-type: none"> 1. Run the program 2. Enter "2" and input any time at or after 5:00 pm 3. Enter "1" to generate the package summary 4. Review the deadline and delivery details columns to ensure the package was delivered at or before the deadline. <p><i>This data is available for review in Section D3 of this document.</i></p>	The hash table stores each package's delivery deadline so that the on_time function can verify that all packages meet their deadline.	<p>The latest delivery for any package with a 9 a.m. deadline was 8:37 a.m. This includes Package 15.</p> <p>The latest delivery for any package with a 10:30 am deadline was 10:12 am. This includes packages 1, 6, 13, 14, 16, 20, 25, 29, 30, 31, 34, 37, and 40.</p> <p>The rest of the packages had an EOD deadline, which wasn't specified. I set the EOD to 5:00 pm, and the last package of the day was delivered at 12:06 pm.</p> <p>There are package summary pictures in sections D1, D2, and D3.</p>
Meet All Package Delivery Constraints	<p>Specified Truck Constraints</p> <ol style="list-style-type: none"> 1. Run the program and use the interface to generate the truck summary at any query time. 2. Confirm that the package 	Package delivery constraints are stored in the package hash table.	 <p>Packages 3, 18, 36, and 38 were loaded onto</p>

C950 Task-2 WGUPS Routing Program

	<p>is listed in the truck's packages.</p> <p>Delayed on Flight Constraints</p> <ol style="list-style-type: none"> 1. Run the program and generate the truck summary at any query time. 2. Find the package ID within the truck package lists. Check the truck's departure time and confirm that it is at or after the end of the package's delay period. <p>Wrong Address Constraints:</p> <ol style="list-style-type: none"> 1. Run the program and generate a query for the package in question. 2. Verify that the correct address is present. <p>Group Delivery Constraints:</p> <ol style="list-style-type: none"> 1. Run the program and generate the truck summary at any query time. 2. Confirm that all packages being delivered together are shown within one truck's package list. 		<p>truck 2.</p>  <p>Package 9's address was updated to 410 S. State St. before delivery.</p>  <p>Packages 13, 14, 15, 16, 19, and 20 were delivered together on truck 1.</p>  <p>Packages 6, 25, 28, and 32 were delayed until 9:05 and left on truck 2 at 9:06.</p>
Show Progress of Trucks and Packages at Assigned Points	<ol style="list-style-type: none"> 1. Run the program. 2. Use the interface to see the desired progress. 	<p>Progress is calculated by comparing dispatch times, query times, and delivery times. Package dispatch times and delivery times are stored in the package hash table. Package dispatch and delivery times are directly related to truck dispatch and delivery times.</p>	 <p>A functional user interface is available.</p>

H1. Other Data Structures

The data structure used to store the packages in the WGUPS routing program must be able to supply all package data using only the package ID as input. To accomplish this, a chaining hash table was used. Two other data structures that could store the packages and meet the requirements are named tuples and linked lists. Both the named tuple and linked list data structures are more efficient in terms of storage space because they do not require empty buckets to be allocated once the number of packages reaches a certain threshold.

H1A. DATA STRUCTURE DIFFERENCES

Since there are no methods associated with our packages outside the hash table's methods, a named tuple would have been a great option to store the packages quickly as objects. Named tuples allow for object creation without creating a class, and the attributes of the objects can be specified when declaring the named tuple. The biggest difference between storing the packages in a named tuple and storing them in the hash table is that named tuples are immutable, so all attributes need to be specified when the named tuple is created. Like the list of lists structure used for the hash table, the rest of the package data could be segmented into a list object, which would still allow the mutable value fields, like the package delivery time and truck, to be adjusted as needed.

The Linked List structure stores the packages in order and links each one to the next rather than hashing the package ID and shuffling the order as the packages are placed in the table. With linked lists, the packages can be stored separately from each other in memory rather than being blocked together within the hash table structure. This allows the system more space freedom while still holding the lists and packages within one searchable structure.

I. Sources

Stack Overflow. (2023, July 3). Python: class 3-opt of TSP is very slow - speeding up permutation creation.

<https://stackoverflow.com/questions/76607784/python-class-3-opt-of-tsp-is-very-slow-speeding-up-permutation-creation>

GeeksforGeeks. (2023, March 28). Load factor and rehashing.

<https://www.geeksforgeeks.org/load-factor-and-rehashing/>

Western Governors University. (2022, December 24). C950 - Webinar-2 - Getting Greedy, who moved my data - Complete Python Code. Retrieved December 21, 2023, from

<https://srm--c.vf.force.com/apex/CourseArticle?id=kA03x000000e1g4CAA#>