

Program Overview Document

A: Algorithm Selection

The core algorithm that this program uses to solve the scenario problem is the **nearest delivery neighbor algorithm** for graph traversal. It is a heuristically greedy and self-adjusting algorithm

The algorithm code and its helpers can be found in `Graph.py`.

More information about the algorithm and how the algorithm functions can be found in the following sections.

B1: Logic Comments

The following is a summary of how the core algorithm finds a solution.

First, to determine what packages to load onto a truck first, the **priority-first algorithm** looks at package deadline data and places packages into a `priority_queue[]` list based on their deadline.

```
for each package in package_table
    if package has deadline
        add package to priority_queue
```

Packages in the `priority_queue[]` are then prioritized first when loading the first truck to ensure that they meet their delivery deadlines. Once all priority packages leave the priority queue, the algorithm then queues up all remaining packages and attempts to load those packages as much as it can until the truck is full.

```
while truck is not full
    if priority_queue is not empty
        for each package in priority_queue
            load package onto truck
    else
        for each package in non_priority_queue
            load package onto truck
```

To deliver the packages, the program utilizes the **nearest delivery neighbor algorithm** to enable our truck to traverse a graph representing an area map of delivery locations to deliver all packages. Packages loaded on to the truck have their delivery locations (`p.get_address()`) tagged on the graph (using a `has_delivery` attribute set to `True`).

```
for packages in truck
    get package's delivery address
    set that delivery address vertex's has_delivery attribute to True
```

Second, our graph traversal, 'TSP', Traveling Salesperson algorithm runs. This algorithm first calls **Dijkstra's Shortest Path** algorithm to find the shortest routes from the starting location (the WGUPS hub) to all other locations on the graph. Then, the **nearest delivery neighbor algorithm** is called. This functions like a 'nearest neighbor' algorithm except it only searches for the nearest neighbor needing a delivery (a vertex with the `has_delivery` attribute). The delivery truck will then traverse the graph from nearest delivery neighbor to nearest delivery neighbor until all deliveries are made, then the truck will return to the hub on standby status awaiting the loading of new packages, repeating the process just described.

```
def nearest_has_delivery_neighbor(graph, start_location):
    first_run_dijkstra(graph, start_location)
    # first run dijkstra's to update edge weights and pred vertexes using distance to current location

    shortest_distance = infinity
    nearest_delivery_neighbor = start_location

    for each_location in all_locations_in_graph:
        if location's_predecessor_does_not_exist(each_location):
            # that means no path from nearest delivery neighbor to location exists.
        else:
            if distance_from_current_location_to_location < shortest_distance AND location_has_delivery(each_location):
                shortest_distance = distance_from_start_location_to_location(each_location)
                nearest_delivery_neighbor = each_location
```

```
return nearest delivery neighbor
```

```
def tsp(graph, first location, truck)
    first locaiton = WGUPS Hub
    first vertex visited = true
    mile tracker = 0.0

    while truck has packages to deliver
        move to new location = find nearest unvisited has_delivery neighbor algorithm
        add distance moved to mile tracker

        for each package on truck
            if package's delivery destination == new locaiton
                unload truck and drop off package

        new location visited = true

        loop back up and find the nearest unvisited has_delivery neighbor until truck has no more packages

    move back to first locaiton
```

B2: Application of Programming Models

The WGUPS application was written in Python 3.8 using the PyCharm IDE. The application provides a console based command line interface to interact with the program. Package and distance data are imported into the application via a parsing function from CSV files stored locally with the application.

Concerning a communication protocol, the application code and data used are contained on the same local machine, so no communication protocol or interaction semantics are needed.

B3: Space-Time Complexity and Big-O

See source code for triple quote comments for each code block's Big-O complexity

Example:

```
'''O(n^2)'''
def ready_route_data(g, locations):

    for loc in locations:
        for v in g.adjacency_list:
            if loc == v.label:
                v.has_delivery = True
```

B4: Adaptability

The core algorithm discussed in section A is self-adjusting. Hence, it is able to be scaled with data as needed.

The application can also import and parse data and create new package and graph objects on its own, given that CSV files are properly formatted. Adding any number of new locations to the distance table will have their location vertexes automatically created, where they can then be added to the graph's `adjacency_list` and `edge_weight` dictionaries, and thus able to be used with the nearest delivery neighbor graph traversal algorithm.

There is also no limitations to how many vertexes can be visited using the NDN algorithm however large data-sets will require a larger cost in terms of performance.

The priority-queue algorithm is able to read package data from the entire package table, regardless of the table size and queue packages according to their deadline attribute. The `priority_list[]` can have items appended and removed on its own as well.

The application may have problems scaling when it comes to dealing truck allocation. Currently truck allocation is handled manually in order to deal with certain special loading constraints. If the number of special constraints not automatically caught by the application increase, there may be issues. However a `truck.truck_id` and a `truck.status` function are already implemented and can assist with automatic truck allocation in the future with some additional code.

B5: Software Efficiency and Maintainability

Concerning overall efficiency, the application runs in ***polynomial time*** with an upper bound big-O complexity of simplified to $O(n^2)$. Currently with a small dataset performance is good and should be noticeably slow until scaled several magnitudes.

Concerning overall maintainability, the application code is neatly formatted with each function definition properly documented, thoroughly explaining the function's abilities, what parameters are taken (if any), and what is returned (if something is returned). Inline comments are also included help to explain complex code further.

Code is divided into several files to keep code organized. In some files there are also some commented out functions that can be un-commented to assist with debugging certain features if needed.

B6: Self-Adjusting Data Structures

Another self-adjusting data structure not mentioned in section B4 is the custom implemented HashTable for quickly storing and retrieving package data.

Package data is automatically parsed and inserted into the master package table hashtable, given that the package file CSV is properly formatted. New packages can be added to the end of the package file and the program will parse it into the hashtable just fine.

Because a hashtable is being used to store package data, inserting and looking up a key should be quick $O(1)$, regardless of the number of packages, that is until the number of packages > hashtable's # of empty buckets, which then it will become $O(N)$ with collision. The hashtable can take an `initial_size` parameter during initialization to change the size of the hashtable depending on the number of packages to be inserted, and whether or not you want to avoid any collisions in the hashtable. However the hashtable is ready scale or to deal with collisions and unknown keys, even though the problem scenario does not require it to be (problem states that all keys are known, and there are no collision for a package table of size 40).

See section D for more information on the hashtable

C: Original Code

The application user interface provides the user with the ability to simulate delivery for all the packages in the package table for that day.

To run a full delivery simulation, please run the application and input '2' when prompted at the main menu.

This runs the delivery simulation and outputs the status of all the packages at the end of the day. You can confirm that all requirements are met by reviewing the package table.

At the bottom is a section called "--STATS--" that confirms the number of packages delivered and the total mileage.

And as per requirements stated in section F and section G, the package table can be viewed as a snapshot during a specific time. This function can be accessed via the user interface as well.

To view the package table at a specific time, please run the application and input '3' when prompted at the main menu. Then input the desired time to view the package table at.

Using this functionality, you can verify delivery status of packages at specific points of time during the day.

C1: Identification Information

Identification information is included in a comment at the top of all code files in the project folder.

```
# Theo Perigo
# Student ID: 001083908
# C950 - Data Structures and Algorithms II
# NHP1 - Performance Assessment - WGUPS Routing Program
# 09/08/2020
# name_of_file.py
```

C2: Process and Flow Comments

Process and flow comments are include for all major blocks of code in all code files in the project folder. Function definitions are clearly written and inline comments are included throughout the code to help explain certain processes and flow.

Example:

```
def run_lookup(snapshot):
    """
    Runs the package lookup function. This function takes a user input string to specify a type of package attribute
    to look up at the previously given time. The user then enters the desired package attribute and the program will
    search the master_package_table HashTable and print the package data for all packages with the given attribute.
    If an invalid input is entered, prints a warning and calls run_lookup() again.
    :param: snapshot: ...etc
    """
```

D: Data Structure

In the program is a custom from-scratch implementation of a HashTable data structure.

See HashTable.py for code

This data structure is created using a primitive `list[]` as its base. First the list is initiated with a size of `initial_size` (defaulting to 40 for this scenario). Within each of these indexes is another `list[]` or *'bucket'* to store data in.

```
def __init__(self, initial_size=40):
    self.size = initial_size
    self.h = []
    for i in range(initial_size):
        self.h.append([])
```

D1: Explanation of Data Structure

A hashtable is a type of data structure used for fast mapping and searching for items in a list via a hashing algorithm. An item's key is hashed to find a bucket index to map its value to, and that key hash can later be used to search for the same item, all at an average of $O(1)$ time.

For this particular scenario, the HashTable will be used to store package data that has been parsed from a package CSV file. The hashtable will thus allow for quick inserting and lookup of package data for use in various parts of the application. A package logistics application such as this program will need to perform these actions many times, so having an quick data structure like a hashtable can make managing package data a lot more efficient.

The items mapped to this hashtable will be a tuple of a package id, and a package object `(p.get_package_id(), p)`. Since a package's id is unique, it can be used to uniquely identify a package as its key.

The package object contains all data for a specific package. A package object represents a real world package to be delivered. A package has a ID, destination address, city, state, zip code, a deadline, a weight, special notes, and a status. A package also has a history dictionary that takes a timestamp as a key, and a copy of the packages attributes at that time. This tracks the changing statuses of the package as it moves throughout its delivery route. All of which will be stored as a value in the hashtable.

E: Hash Table

Each item that will be stored in the hashtable is a `(tuple)` of a `key` and a `value` pair -> `(key, value)`.

In order to determine which *bucket* to put the item into, the item's `key` is hashed using the hashtable's hash formula.

```
def _hash(self, key):
    s = len(self.h)
    return key % s
```

The hashtable can use its `set()` function to insert an item to the bucket at the index given by the hash formula.

```
def set(self, key, value):
    index = self._hash(key)
    bucket = self.h[index]
    if len(bucket) == 0:
        bucket.append([key, value])
    else:
        # Collision handling
        for i in range(len(bucket)):
            if bucket[i][0] == key:
                bucket[i] = [key, value]
```

```
        return
    bucket.append([key, value])
```

If a bucket already has an item, a collision occurs. To handle this, the hashtable utilized a chaining as its method of collision resolution.

F: Look-up Function

Items in the Hashtable can be looked up by their key. Just like when inserting, an item's key is first hashed to find the bucket index to look up the item.

```
def get(self, key):
    index = self._hash(key)
    bucket = self.h[index]
    if bucket:
        if len(bucket) == 1 and bucket[0][0] == key:
            return bucket[0][1]
        else:
            # Collision handling
            for i in range(len(bucket)):
                if bucket[i][0] == key:
                    return bucket[i][1]
    return None
```

G: Interface

The application provides a console based command line interface to allow the user to interact with the program.

Demonstration on how to use the User Interface

At program start, the main menu appears and displays a list of options to the user.

```
-----===== WGUPS Command Line Interface | ver. 0.9.1 BETA -----

-----===== MAIN MENU -----

What would you like to do?
  1. Look up package(s)
  2. Run full day delivery simulation
  3. View package table status at a specified time
  4. Exit the program

Enter a number:
```

From here, the user can tell the application to perform certain actions via text input prompts.

User enters '1' to look up a package.

```
-----===== PACKAGE LOOKUP -----

Please enter a time to check delivery status for in HH:MM format (example - '8:35' or '13:12'):
```

User enters '10:20' to check that package's status at that time.

```
Running delivery simulation . . . Simulation complete

Retrieving snapshot of package table data from the specified time: [10:20]

Please choose a type of input to lookup:
  1. Package ID
  2. Delivery Address
  3. Delivery Deadline
  4. Delivery City
  5. Delivery Zip Code
  6. Package Weight
  7. Delivery Status
  8. Return to Main Menu
```

Enter a number:
Please enter a Package ID:

User enters '1', then '30' to look up Package ID 30.

And the results are displayed:

Package ID: 30
Address: 300 STATE ST
City: SALT LAKE CITY
State: UT
Zip Code: 84103
Delivery Deadline: 10:30 AM
Package Weight 1 kg
Special Notes: N/A
Status: DELIVERED by SAM at 09:00

G1: First Status Check

The application is able to show a listing of all packages and the current status of each package at 8:35 AM.

See screenshot folder for image files or run application and choose 3. View package table status at a specified time at the main menu.

ID	Address	City	State	Zip Code	Deadline	Weight	Special Notes	Status
1	195 W OAKLAND AVE	SALT LAKE CITY	UT	84115	10:30 AM	21	N/A	DELIVERED by SAM at 08:29
2	2530 S 500 E	SALT LAKE CITY	UT	84106	EOD	44	N/A	AT HUB
3	233 CANYON RD	SALT LAKE CITY	UT	84103	EOD	2	Can only be on truck 2	AT HUB
4	300 W 2000 S	SALT LAKE CITY	UT	84115	EOD	4	N/A	AT HUB
5	410 S STATE ST	SALT LAKE CITY	UT	84111	EOD	5	N/A	AT HUB
6	3060 LESTER ST	WEST VALLEY CITY	UT	84119	10:30 AM	88	Delayed on flight---will not arrive to depot until 9:05 am	AT HUB
7	1330 2100 S	SALT LAKE CITY	UT	84106	EOD	8	N/A	AT HUB
8	300 STATE ST	SALT LAKE CITY	UT	84103	EOD	9	N/A	AT HUB
9	300 STATE ST	SALT LAKE CITY	UT	84103	EOD	2	Wrong address listed	AT HUB
10	600 E 900 S	SALT LAKE CITY	UT	84105	EOD	1	N/A	AT HUB
11	2600 TAYLORSVILLE BLVD	SALT LAKE CITY	UT	84118	EOD	1	N/A	AT HUB
12	3575 W VALLEY CENTRAL STATION BUS LOOP	WEST VALLEY CITY	UT	84119	EOD	1	N/A	AT HUB
13	2010 W 500 S	SALT LAKE CITY	UT	84104	10:30 AM	2	N/A	EN-ROUTE on Truck 01
14	4300 S 1300 E	MILLCREEK	UT	84117	10:30 AM	88	Must be delivered with 15, 19	DELIVERED by SAM at 08:06
15	4500 S 2300 E	HOLLADAY	UT	84117	09:00 AM	4	N/A	EN-ROUTE on Truck 01
16	4500 S 2300 E	HOLLADAY	UT	84117	10:30 AM	88	Must be delivered with 13, 19	EN-ROUTE on Truck 01
17	3148 S 1100 W	SALT LAKE CITY	UT	84119	EOD	2	N/A	AT HUB
18	1400 4000 S	SALT LAKE CITY	UT	84123	EOD	6	Can only be on truck 2	AT HUB
19	177 W PRICE AVE	SALT LAKE CITY	UT	84115	EOD	37	N/A	DELIVERED by SAM at 08:14
20	3595 MAIN ST	SALT LAKE CITY	UT	84115	10:30 AM	37	Must be delivered with 13, 15	DELIVERED by SAM at 08:13
21	3595 MAIN ST	SALT LAKE CITY	UT	84115	EOD	3	N/A	AT HUB
22	6351 S 900 E	MURRAY	UT	84121	EOD	2	N/A	AT HUB
23	5100 S 2700 W	SALT LAKE CITY	UT	84118	EOD	5	N/A	AT HUB
24	5025 STATE ST	MURRAY	UT	84107	EOD	7	N/A	AT HUB
25	5383 S 900 E #104	SALT LAKE CITY	UT	84117	10:30 AM	7	Delayed on flight---will not arrive to depot until 9:05 am	AT HUB
26	5383 S 900 E #104	SALT LAKE CITY	UT	84117	EOD	25	N/A	AT HUB
27	1060 DALTON AVE S	SALT LAKE CITY	UT	84104	EOD	5	N/A	AT HUB
28	2835 MAIN ST	SALT LAKE CITY	UT	84115	EOD	7	Delayed on flight---will not arrive to depot until 9:05 am	AT HUB
29	1330 2100 S	SALT LAKE CITY	UT	84106	10:30 AM	2	N/A	EN-ROUTE on Truck 01
30	300 STATE ST	SALT LAKE CITY	UT	84103	10:30 AM	1	N/A	EN-ROUTE on Truck 01
31	3365 S 900 W	SALT LAKE CITY	UT	84119	10:30 AM	1	N/A	DELIVERED by SAM at 08:20
32	3365 S 900 W	SALT LAKE CITY	UT	84119	EOD	1	Delayed on flight---will not arrive to depot until 9:05 am	AT HUB
33	2530 S 500 E	SALT LAKE CITY	UT	84106	EOD	1	N/A	AT HUB
34	4500 S 2300 E	HOLLADAY	UT	84117	10:30 AM	2	N/A	EN-ROUTE on Truck 01
35	1060 DALTON AVE S	SALT LAKE CITY	UT	84104	EOD	88	N/A	AT HUB
36	2300 PARKWAY BLVD	WEST VALLEY CITY	UT	84119	EOD	88	Can only be on truck 2	AT HUB
37	410 S STATE ST	SALT LAKE CITY	UT	84111	10:30 AM	2	N/A	EN-ROUTE on Truck 01
38	410 S STATE ST	SALT LAKE CITY	UT	84111	EOD	9	Can only be on truck 2	AT HUB
39	2010 W 500 S	SALT LAKE CITY	UT	84104	EOD	9	N/A	AT HUB
40	300 W 2000 S	SALT LAKE CITY	UT	84115	10:30 AM	45	N/A	DELIVERED by SAM at 08:25

G2: Second Status Check

The application is able to show a listing of all packages and the current status of each package at 10:25 AM.

See screenshot folder for image files or run application and choose 3. View package table status at a specified time at the main menu.

Run - NHP1

Run: Main

ID	Address	City	State	Zip Code	Deadline	Weight	Special Notes	Status
1	195 W OAKLAND AVE	SALT LAKE CITY	UT	84115	10:30 AM	21	N/A	DELIVERED by SAM at 08:29
2	2530 S 500 E	SALT LAKE CITY	UT	84106	EOD	44	N/A	DELIVERED by HIGGS at 09:23
3	233 CANYON RD	SALT LAKE CITY	UT	84103	EOD	2	Can only be on truck 2	DELIVERED by HIGGS at 10:10
4	300 W 2000 S	SALT LAKE CITY	UT	84115	EOD	4	N/A	DELIVERED by HIGGS at 09:17
5	410 S STATE ST	SALT LAKE CITY	UT	84111	EOD	5	N/A	DELIVERED by HIGGS at 10:07
6	3000 LESTER ST	WEST VALLEY CITY	UT	84119	10:30 AM	88	Delayed on flight---will not arrive to depot until 9:05 am	DELIVERED by HIGGS at 09:46
7	1330 2100 S	SALT LAKE CITY	UT	84106	EOD	8	N/A	DELIVERED by HIGGS at 09:28
8	300 STATE ST	SALT LAKE CITY	UT	84103	EOD	9	N/A	DELIVERED by HIGGS at 10:12
9	410 S STATE ST	SALT LAKE CITY	UT	84111	EOD	2	Fixed Address	AT HUB
10	600 E 900 S	SALT LAKE CITY	UT	84105	EOD	1	N/A	DELIVERED by HIGGS at 10:01
11	2600 TAYLORSVILLE BLVD	SALT LAKE CITY	UT	84118	EOD	1	N/A	EN-ROUTE on Truck 02
12	3575 W VALLEY CENTRAL STATION BUS LOOP	WEST VALLEY CITY	UT	84119	EOD	1	N/A	DELIVERED by HIGGS at 09:34
13	2010 W 500 S	SALT LAKE CITY	UT	84104	10:30 AM	2	N/A	DELIVERED by SAM at 09:10
14	4300 S 1300 E	MILLCREEK	UT	84117	10:30 AM	88	Must be delivered with 15, 19	DELIVERED by SAM at 08:06
15	4500 S 2300 E	HOLLADAY	UT	84117	09:00 AM	4	N/A	DELIVERED by SAM at 08:35
16	4500 S 2300 E	HOLLADAY	UT	84117	10:30 AM	88	Must be delivered with 13, 19	DELIVERED by SAM at 08:35
17	3148 S 1100 W	SALT LAKE CITY	UT	84119	EOD	2	N/A	DELIVERED by HIGGS at 09:42
18	1488 4000 S	SALT LAKE CITY	UT	84123	EOD	6	Can only be on truck 2	EN-ROUTE on Truck 02
19	177 W PRICE AVE	SALT LAKE CITY	UT	84115	EOD	37	N/A	DELIVERED by SAM at 08:14
20	3595 MAIN ST	SALT LAKE CITY	UT	84115	10:30 AM	37	Must be delivered with 13, 15	DELIVERED by SAM at 08:13
21	3595 MAIN ST	SALT LAKE CITY	UT	84115	EOD	3	N/A	DELIVERED by HIGGS at 09:11
22	6351 S 900 E	MURRAY	UT	84121	EOD	2	N/A	DELIVERED by HIGGS at 10:24
23	5100 S 2700 W	SALT LAKE CITY	UT	84118	EOD	5	N/A	AT HUB
24	5025 STATE ST	MURRAY	UT	84107	EOD	7	N/A	AT HUB
25	5383 S 900 E #104	SALT LAKE CITY	UT	84117	10:30 AM	7	Delayed on flight---will not arrive to depot until 9:05 am	AT HUB
26	5383 S 900 E #104	SALT LAKE CITY	UT	84117	EOD	25	N/A	AT HUB
27	1060 DALTON AVE S	SALT LAKE CITY	UT	84104	EOD	5	N/A	AT HUB
28	2835 MAIN ST	SALT LAKE CITY	UT	84115	EOD	7	Delayed on flight---will not arrive to depot until 9:05 am	AT HUB
29	1330 2100 S	SALT LAKE CITY	UT	84106	10:30 AM	2	N/A	DELIVERED by SAM at 08:45
30	300 STATE ST	SALT LAKE CITY	UT	84103	10:30 AM	1	N/A	DELIVERED by SAM at 09:00
31	3365 S 900 W	SALT LAKE CITY	UT	84119	10:30 AM	1	N/A	DELIVERED by SAM at 08:20
32	3365 S 900 W	SALT LAKE CITY	UT	84119	EOD	1	Delayed on flight---will not arrive to depot until 9:05 am	AT HUB
33	2530 S 500 E	SALT LAKE CITY	UT	84106	EOD	1	N/A	AT HUB
34	4500 S 2300 E	HOLLADAY	UT	84117	10:30 AM	2	N/A	DELIVERED by SAM at 08:35
35	1060 DALTON AVE S	SALT LAKE CITY	UT	84104	EOD	88	N/A	AT HUB
36	2300 PARKWAY BLVD	WEST VALLEY CITY	UT	84119	EOD	88	Can only be on truck 2	DELIVERED by HIGGS at 09:51
37	410 S STATE ST	SALT LAKE CITY	UT	84111	10:30 AM	2	N/A	DELIVERED by SAM at 08:56
38	410 S STATE ST	SALT LAKE CITY	UT	84111	EOD	9	Can only be on truck 2	DELIVERED by HIGGS at 10:07
39	2010 W 500 S	SALT LAKE CITY	UT	84104	EOD	9	N/A	AT HUB
40	300 W 2800 S	SALT LAKE CITY	UT	84115	10:30 AM	45	N/A	DELIVERED by SAM at 08:25

G3: Third Status Check

The application is able to show a listing of all packages and the current status of each package at 1:00 PM

See screenshot folder for image files or run application and choose 3. View package table status at a specified time at the main menu.

Run - NHP1

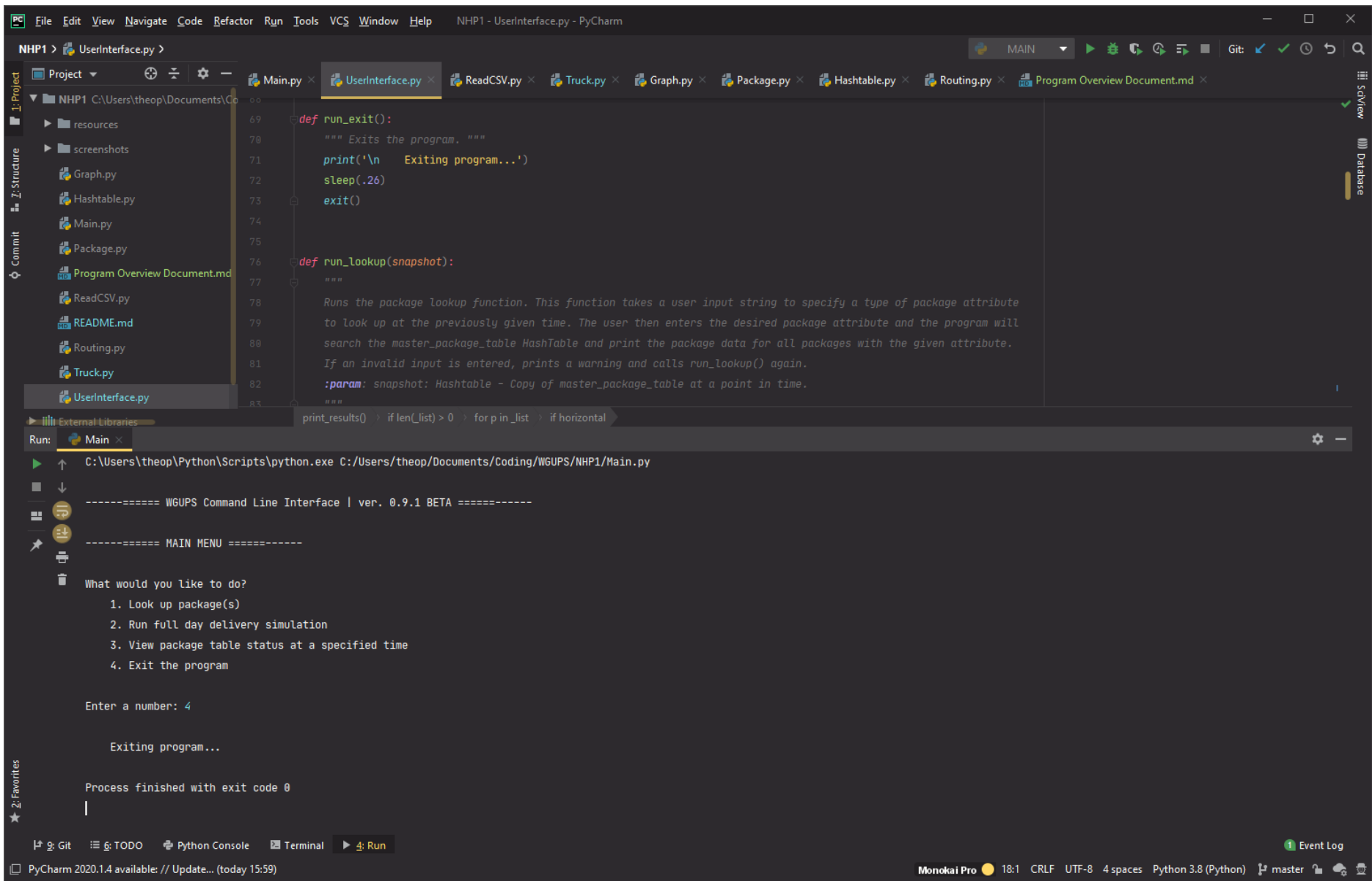
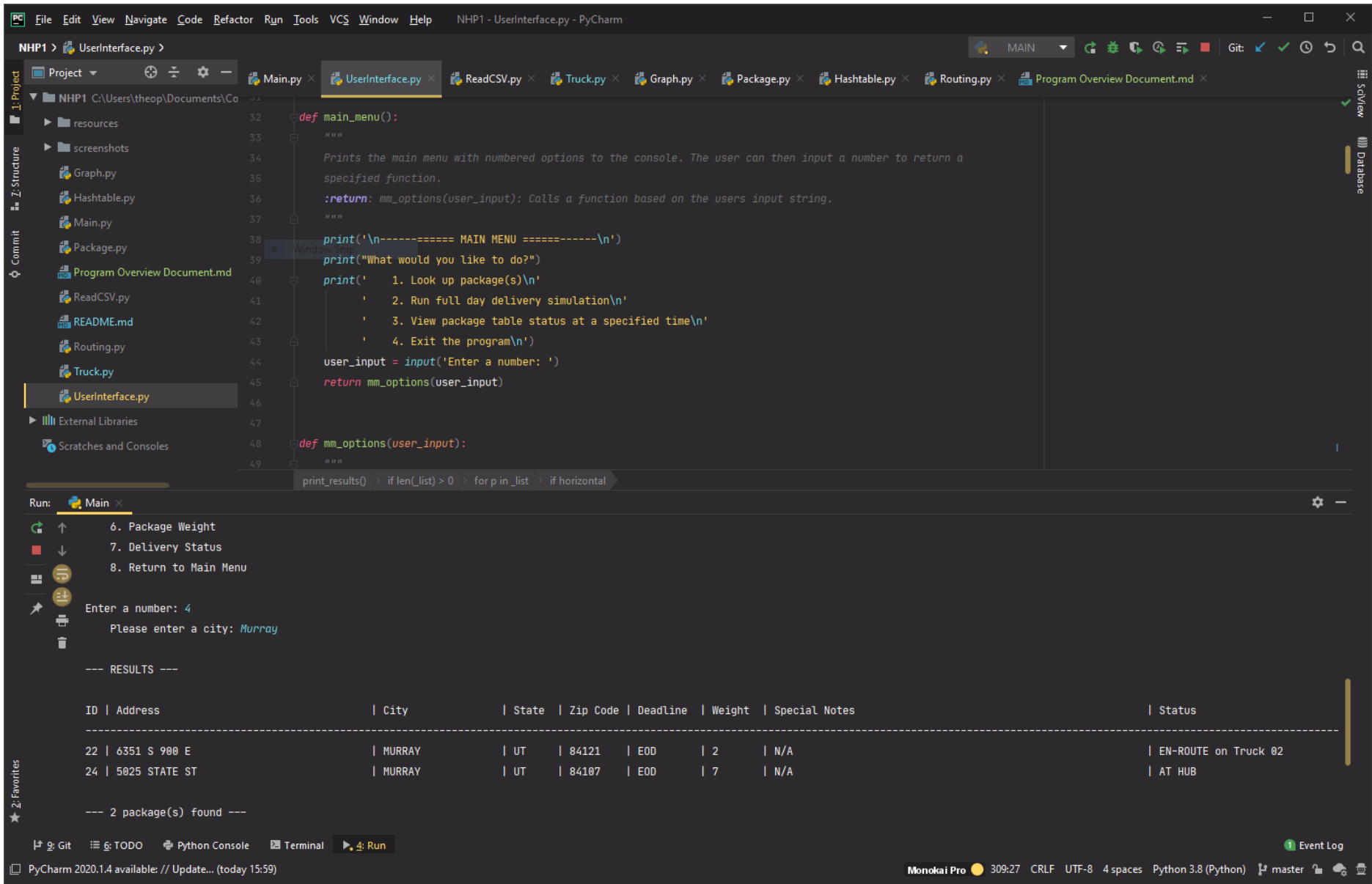
Run: Main

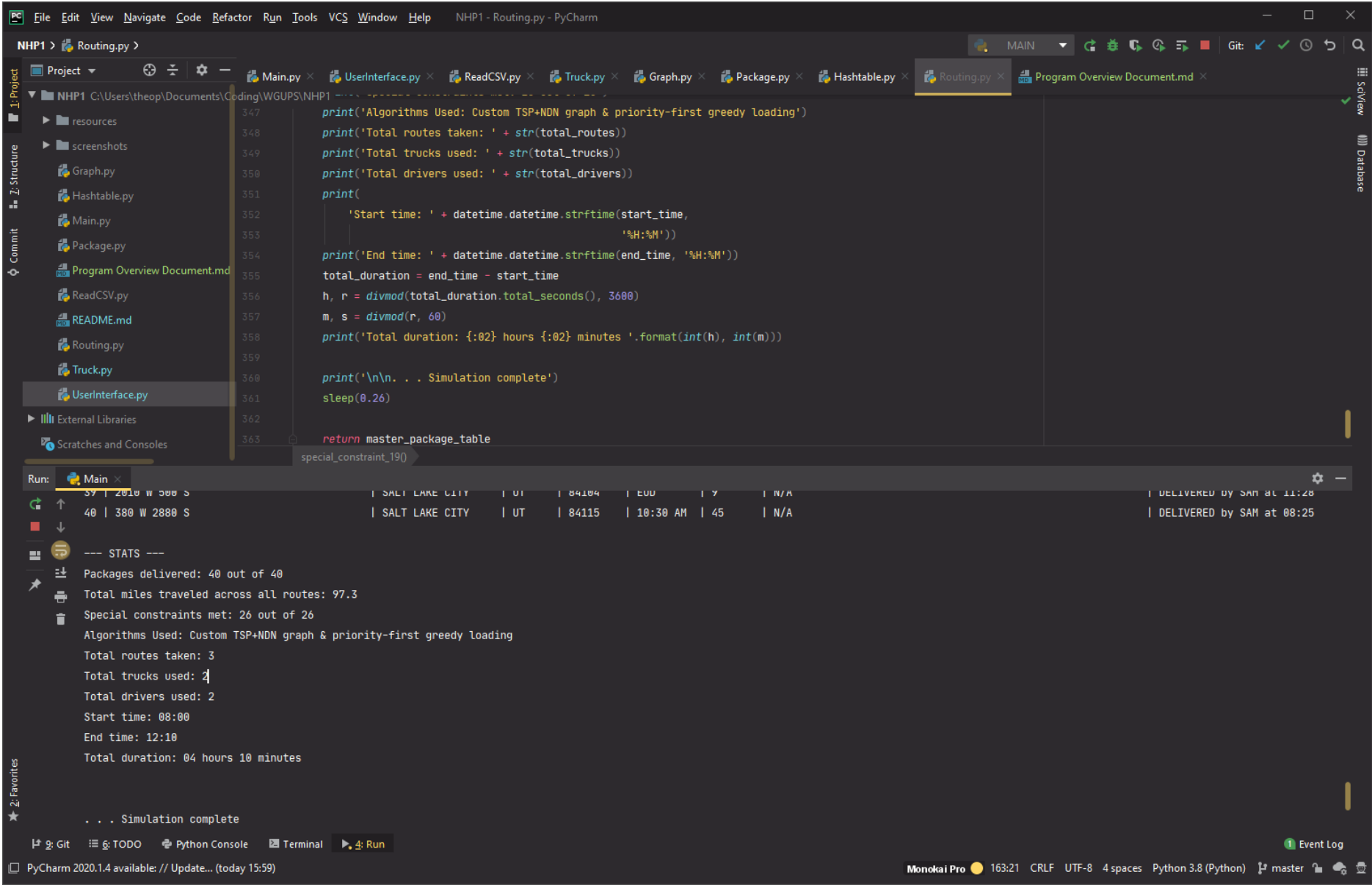
ID	Address	City	State	Zip Code	Deadline	Weight	Special Notes	Status
1	195 W OAKLAND AVE	SALT LAKE CITY	UT	84115	10:30 AM	21	N/A	DELIVERED by SAM at 08:29
2	2530 S 500 E	SALT LAKE CITY	UT	84106	EOD	44	N/A	DELIVERED by HIGGS at 09:23
3	233 CANYON RD	SALT LAKE CITY	UT	84103	EOD	2	Can only be on truck 2	DELIVERED by HIGGS at 10:10
4	300 W 2000 S	SALT LAKE CITY	UT	84115	EOD	4	N/A	DELIVERED by HIGGS at 09:17
5	410 S STATE ST	SALT LAKE CITY	UT	84111	EOD	5	N/A	DELIVERED by HIGGS at 10:07
6	3000 LESTER ST	WEST VALLEY CITY	UT	84119	10:30 AM	88	Delayed on flight---will not arrive to depot until 9:05 am	DELIVERED by HIGGS at 09:46
7	1330 2100 S	SALT LAKE CITY	UT	84106	EOD	8	N/A	DELIVERED by HIGGS at 09:28
8	300 STATE ST	SALT LAKE CITY	UT	84103	EOD	9	N/A	DELIVERED by HIGGS at 10:12
9	410 S STATE ST	SALT LAKE CITY	UT	84111	EOD	2	Fixed Address	DELIVERED by SAM at 11:18
10	600 E 900 S	SALT LAKE CITY	UT	84105	EOD	1	N/A	DELIVERED by HIGGS at 10:01
11	2600 TAYLORSVILLE BLVD	SALT LAKE CITY	UT	84118	EOD	1	N/A	DELIVERED by HIGGS at 10:40
12	3575 W VALLEY CENTRAL STATION BUS LOOP	WEST VALLEY CITY	UT	84119	EOD	1	N/A	DELIVERED by HIGGS at 09:34
13	2010 W 500 S	SALT LAKE CITY	UT	84104	10:30 AM	2	N/A	DELIVERED by SAM at 09:10
14	4300 S 1300 E	MILLCREEK	UT	84117	10:30 AM	88	Must be delivered with 15, 19	DELIVERED by SAM at 08:06
15	4500 S 2300 E	HOLLADAY	UT	84117	09:00 AM	4	N/A	DELIVERED by SAM at 08:35
16	4500 S 2300 E	HOLLADAY	UT	84117	10:30 AM	88	Must be delivered with 13, 19	DELIVERED by SAM at 08:35
17	3148 S 1100 W	SALT LAKE CITY	UT	84119	EOD	2	N/A	DELIVERED by HIGGS at 09:42
18	1488 4000 S	SALT LAKE CITY	UT	84123	EOD	6	Can only be on truck 2	DELIVERED by HIGGS at 10:37
19	177 W PRICE AVE	SALT LAKE CITY	UT	84115	EOD	37	N/A	DELIVERED by SAM at 08:14
20	3595 MAIN ST	SALT LAKE CITY	UT	84115	10:30 AM	37	Must be delivered with 13, 15	DELIVERED by SAM at 08:13
21	3595 MAIN ST	SALT LAKE CITY	UT	84115	EOD	3	N/A	DELIVERED by HIGGS at 09:11
22	6351 S 900 E	MURRAY	UT	84121	EOD	2	N/A	DELIVERED by HIGGS at 10:24
23	5100 S 2700 W	SALT LAKE CITY	UT	84118	EOD	5	N/A	DELIVERED by SAM at 11:49
24	5025 STATE ST	MURRAY	UT	84107	EOD	7	N/A	DELIVERED by SAM at 10:38
25	5383 S 900 E #104	SALT LAKE CITY	UT	84117	10:30 AM	7	Delayed on flight---will not arrive to depot until 9:05 am	DELIVERED by SAM at 10:43
26	5383 S 900 E #104	SALT LAKE CITY	UT	84117	EOD	25	N/A	DELIVERED by SAM at 10:43
27	1060 DALTON AVE S	SALT LAKE CITY	UT	84104	EOD	5	N/A	DELIVERED by SAM at 11:34
28	2835 MAIN ST	SALT LAKE CITY	UT	84115	EOD	7	Delayed on flight---will not arrive to depot until 9:05 am	DELIVERED by SAM at 10:56
29	1330 2100 S	SALT LAKE CITY	UT	84106	10:30 AM	2	N/A	DELIVERED by SAM at 08:45
30	300 STATE ST	SALT LAKE CITY	UT	84103	10:30 AM	1	N/A	DELIVERED by SAM at 09:00
31	3365 S 900 W	SALT LAKE CITY	UT	84119	10:30 AM	1	N/A	DELIVERED by SAM at 08:20
32	3365 S 900 W	SALT LAKE CITY	UT	84119	EOD	1	Delayed on flight---will not arrive to depot until 9:05 am	DELIVERED by SAM at 11:05
33	2530 S 500 E	SALT LAKE CITY	UT	84106	EOD	1	N/A	DELIVERED by SAM at 10:53
34	4500 S 2300 E	HOLLADAY	UT	84117	10:30 AM	2	N/A	DELIVERED by SAM at 08:35
35	1060 DALTON AVE S	SALT LAKE CITY	UT	84104	EOD	88	N/A	DELIVERED by SAM at 11:34
36	2300 PARKWAY BLVD	WEST VALLEY CITY	UT	84119	EOD	88	Can only be on truck 2	DELIVERED by HIGGS at 09:51
37	410 S STATE ST	SALT LAKE CITY	UT	84111	10:30 AM	2	N/A	DELIVERED by SAM at 08:56
38	410 S STATE ST	SALT LAKE CITY	UT	84111	EOD	9	Can only be on truck 2	DELIVERED by HIGGS at 10:07
39	2010 W 500 S	SALT LAKE CITY	UT	84104	EOD	9	N/A	DELIVERED by SAM at 11:28
40	300 W 2800 S	SALT LAKE CITY	UT	84115	10:30 AM	45	N/A	DELIVERED by SAM at 08:25

H: Screenshots of Code Execution

The application is able to execute successfully to completion free from runtime errors or warning.

See screenshot folder for image files





I1: Strengths of the Chosen Algorithm

One of the greatest strengths that the nearest delivery algorithm provides is a method of traversing a graph on its own. A user does not need to manually input a path, as the algorithm can determine which nodes are nearest to it via its own calculations.

This ease-of-use can be another strength for this algorithm. The algorithm only needs to know adjacent vertexes and their distance away to work.

New vertexes and distances can also be added to a graph without needing to change the algorithm at all, allowing for data scalability.

I2: Verification of Algorithm

The simulation is able to deliver **all 40 packages in 97.3 miles** while meeting all special constraints.

The user can verify the mileage and deliveries via the user interface.

To run a full delivery simulation, please run the application and input '2' when prompted at the main menu.

At the bottom is a section called "--STATS--" that confirms the number of packages delivered and the total mileage along with some other information. The package table is also displayed so each individual package status can be checked.

I3: Other Possible Algorithms

Instead of using a nearest neighbor style algorithm to find a solution, other options include:

- A brute-force search algorithm
- Branch-and-bound algorithm

These algorithms are compared and contrast in the next section (I3A)

I3A: Algorithm Differences

Both a brute-force search and a branch-and-bound algorithm would most definitely result in a lower total mileage as they aim to find exact optimal solutions to visiting all the vertexes on a graph.

A **brute-force search** would run every possible route permutation and return the one with the lowest mileage at the end. While this may be useful for cases where the number of vertexes is low, this algorithm becomes almost too process intensive at higher vertex counts due to is $O(n!)$ complexity.

A **branch-and-bound** algorithm would be a better choice over the brute-force search as its complexity is better at a $O(2^n)$, though in its worst case may end up running every possible route like a brute-force search would. This algorithm uses a state-space tree structure to find the most optimal solution to its problem. It first starts at a starting vertex, and then all the nodes that it can travel to become that node's children. Then each of those children have children that are all the nodes that they can still visit that was not the parent, or that parents parent, etc, and so on, until a tree of all routes is made. The algorithm will try find the most optimal branches and only travel down those path, potentially skipping a lot of the rest of the tree.

These two algorithms would find a more optimal solution than nearest neighbor algorithm would as the NN algorithm is a heuristic type of greedy algorithm that forgoes finding an optimal solution for better performance $O(n^2)$.

J: Different Approach

One aspect of the project I would change would be how I process special constraints and truck allocation. Currently 3 trucks are manually allocated at 3 different times to start loading and deploy on their routes. This was in order to ensure that certain special constraints could be met (wrong address, packages must be on truck 2, delayed packages with early deadlines). This is especially the case for strange constraints such as packages needing to be delivered together, but not to the same address.

I would modify the process by implementing a truck queue system that would take any available trucks on standby and fill them with available drivers on standby, to then be loaded and dispatched simultaneously. The special constrain checking would need to be automated and improved, as well as expanded to potentially catch different types of constraints that could be found in the package data excel file.

K1: Verification of Data Structure and Solution

Verification of all packages being delivered in <145 miles:

The simulation is able to deliver **all 40 packages in 97.3 miles** while meeting all special constraints.

Verification of successful package delivery:

All Packages are delivered on time according to their constraints outlined in the package notes.

The user can verify the both the mileage and deliveries via the user interface.

▮ To run a full delivery simulation, please run the application and input '2' when prompted at the main menu.

At the bottom is a section called "--STATS--" that confirms the number of packages delivered and the total mileage along with some other information. The package table is also displayed so each individual package status can be checked.

ID	Address	City	State	Zip Code	Deadline	Weight	Special Notes	Status
1	195 W OAKLAND AVE	SALT LAKE CITY	UT	84115	10:30 AM	21	N/A	DELIVERED by SAM at 08:29
2	2530 S 500 E	SALT LAKE CITY	UT	84106	EOD	44	N/A	DELIVERED by HIGGS at 09:23
3	233 CANYON RD	SALT LAKE CITY	UT	84103	EOD	2	Can only be on truck 2	DELIVERED by HIGGS at 18:10
4	300 W 2800 S	SALT LAKE CITY	UT	84115	EOD	4	N/A	DELIVERED by HIGGS at 09:17
5	410 S STATE ST	SALT LAKE CITY	UT	84111	EOD	5	N/A	DELIVERED by HIGGS at 18:07
6	3000 LESTER ST	WEST VALLEY CITY	UT	84119	10:30 AM	88	Delayed on flight---will not arrive to depot until 9:05 am	DELIVERED by HIGGS at 09:46
7	1330 2100 S	SALT LAKE CITY	UT	84106	EOD	8	N/A	DELIVERED by HIGGS at 09:28
8	300 STATE ST	SALT LAKE CITY	UT	84103	EOD	9	N/A	DELIVERED by HIGGS at 10:12
9	410 S STATE ST	SALT LAKE CITY	UT	84111	EOD	2	Fixed Address	DELIVERED by SAM at 11:18
10	600 E 900 S	SALT LAKE CITY	UT	84105	EOD	1	N/A	DELIVERED by HIGGS at 10:01
11	2600 TAYLORSVILLE BLVD	SALT LAKE CITY	UT	84118	EOD	1	N/A	DELIVERED by HIGGS at 10:40
12	3575 W VALLEY CENTRAL STATION BUS LOOP	WEST VALLEY CITY	UT	84119	EOD	1	N/A	DELIVERED by HIGGS at 09:34
13	2010 W 500 S	SALT LAKE CITY	UT	84104	10:30 AM	2	N/A	DELIVERED by SAM at 09:10
14	4300 S 1300 E	MILLCREEK	UT	84117	10:30 AM	88	Must be delivered with 15, 19	DELIVERED by SAM at 08:06
15	4500 S 2300 E	HOLLADAY	UT	84117	09:00 AM	4	N/A	DELIVERED by SAM at 08:35
16	4500 S 2300 E	HOLLADAY	UT	84117	10:30 AM	88	Must be delivered with 13, 19	DELIVERED by SAM at 08:35
17	3148 S 1100 W	SALT LAKE CITY	UT	84119	EOD	2	N/A	DELIVERED by HIGGS at 09:42
18	1488 4800 S	SALT LAKE CITY	UT	84123	EOD	6	Can only be on truck 2	DELIVERED by HIGGS at 10:37
19	177 W PRICE AVE	SALT LAKE CITY	UT	84115	EOD	37	N/A	DELIVERED by SAM at 08:14
20	3595 MAIN ST	SALT LAKE CITY	UT	84115	10:30 AM	37	Must be delivered with 13, 15	DELIVERED by SAM at 08:13
21	3595 MAIN ST	SALT LAKE CITY	UT	84115	EOD	3	N/A	DELIVERED by HIGGS at 09:11
22	6351 S 900 E	MURRAY	UT	84121	EOD	2	N/A	DELIVERED by HIGGS at 10:24
23	5100 S 2700 W	SALT LAKE CITY	UT	84118	EOD	5	N/A	DELIVERED by SAM at 11:49
24	5025 STATE ST	MURRAY	UT	84107	EOD	7	N/A	DELIVERED by SAM at 10:30
25	5383 S 900 E #104	SALT LAKE CITY	UT	84117	10:30 AM	7	Delayed on flight---will not arrive to depot until 9:05 am	DELIVERED by SAM at 10:43
26	5383 S 900 E #104	SALT LAKE CITY	UT	84117	EOD	25	N/A	DELIVERED by SAM at 10:43
27	1060 DALTON AVE S	SALT LAKE CITY	UT	84104	EOD	5	N/A	DELIVERED by SAM at 11:34
28	2835 MAIN ST	SALT LAKE CITY	UT	84115	EOD	7	Delayed on flight---will not arrive to depot until 9:05 am	DELIVERED by SAM at 10:56
29	1330 2100 S	SALT LAKE CITY	UT	84106	10:30 AM	2	N/A	DELIVERED by SAM at 08:45
30	300 STATE ST	SALT LAKE CITY	UT	84103	10:30 AM	1	N/A	DELIVERED by SAM at 09:00
31	3365 S 900 W	SALT LAKE CITY	UT	84119	10:30 AM	1	N/A	DELIVERED by SAM at 08:20
32	3365 S 900 W	SALT LAKE CITY	UT	84119	EOD	1	Delayed on flight---will not arrive to depot until 9:05 am	DELIVERED by SAM at 11:05
33	2530 S 500 E	SALT LAKE CITY	UT	84106	EOD	1	N/A	DELIVERED by SAM at 10:53
34	4500 S 2300 E	HOLLADAY	UT	84117	10:30 AM	2	N/A	DELIVERED by SAM at 08:35
35	1060 DALTON AVE S	SALT LAKE CITY	UT	84104	EOD	88	N/A	DELIVERED by SAM at 11:34
36	2300 PARKWAY BLVD	WEST VALLEY CITY	UT	84119	EOD	88	Can only be on truck 2	DELIVERED by HIGGS at 09:51
37	410 S STATE ST	SALT LAKE CITY	UT	84111	10:30 AM	2	N/A	DELIVERED by SAM at 08:56
38	410 S STATE ST	SALT LAKE CITY	UT	84111	EOD	9	Can only be on truck 2	DELIVERED by HIGGS at 10:07
39	2010 W 500 S	SALT LAKE CITY	UT	84104	EOD	9	N/A	DELIVERED by SAM at 11:20
40	300 W 2800 S	SALT LAKE CITY	UT	84115	10:30 AM	45	N/A	DELIVERED by SAM at 08:25

Project

NHP1

C:\Users\theop\Documents\Coding\WGUPS\NHP1

resources

screenshots

Graph.py

Hashtable.py

Main.py

Package.py

Program Overview Document.md

ReadCSV.py

README.md

Routing.py

Truck.py

UserInterface.py

External Libraries

Scratches and Consoles

Run: Main

39 | 2010 W 500 S | SALT LAKE CITY | UT | 84104 | EOD | 9 | N/A | DELIVERED by SAM at 11:20

40 | 300 W 2800 S | SALT LAKE CITY | UT | 84115 | 10:30 AM | 45 | N/A | DELIVERED by SAM at 08:25

--- STATS ---

Packages delivered: 40 out of 40

Total miles traveled across all routes: 97.3

Special constraints met: 26 out of 26

Algorithms Used: Custom TSP+NDN graph & priority-first greedy loading

Total routes taken: 3

Total trucks used: 4

Total drivers used: 2

Start time: 08:00

End time: 12:10

Total duration: 04 hours 10 minutes

... Simulation complete

Git | TODO | Python Console | Terminal | Run

PyCharm 2020.1.4 available: // Update... (today 15:59)

Monokai Pro 163:21 CRLF UTF-8 4 spaces Python 3.8 (Python) master

NHP1 > Routing.py

MAIN

print('Algorithms Used: Custom TSP+NDN graph & priority-first greedy loading')

print('Total routes taken: ' + str(total_routes))

print('Total trucks used: ' + str(total_trucks))

print('Total drivers used: ' + str(total_drivers))

print('Start time: ' + datetime.datetime.strftime(start_time, '%H:%M'))

print('End time: ' + datetime.datetime.strftime(end_time, '%H:%M'))

total_duration = end_time - start_time

h, r = divmod(total_duration.total_seconds(), 3600)

m, s = divmod(r, 60)

print('Total duration: {:02} hours {:02} minutes '.format(int(h), int(m)))

print('\n\n. . . Simulation complete')

sleep(0.26)

return master_package_table

Verification of a hashtable with a look up function being present:

The program implements a custom from-scratch Hashtable data structure. For more information on the Hashtable please read the following items:

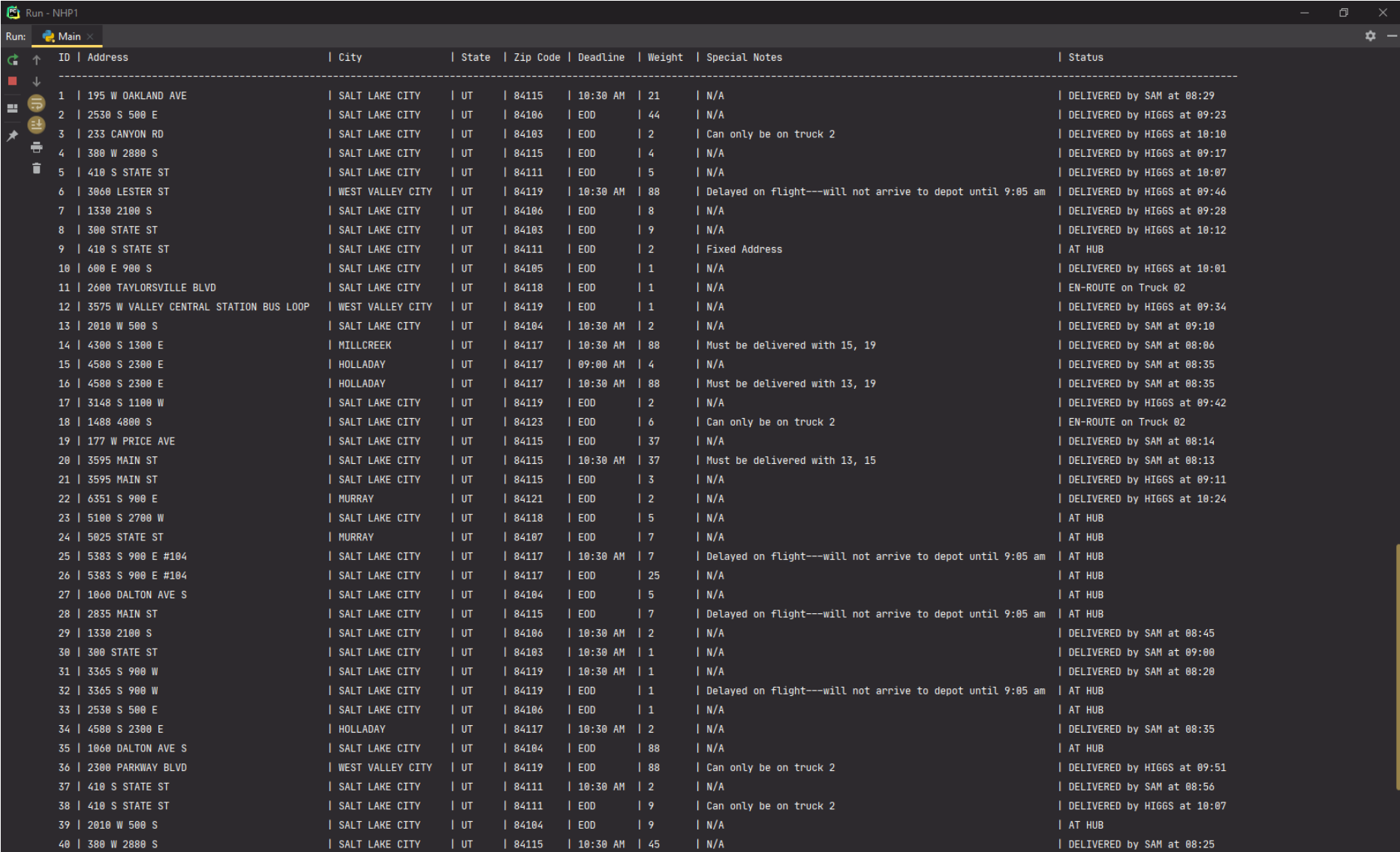
- Section D: Data Structure
- Section D: Explanation of Data Structure
- Section E: Hash Table
- Section F: Look-up Function
- Hashtable.py

Items in the Hashtable can be looked up by their key. An item's key is first hashed to find the bucket index to look up the item.

```
def get(self, key):
    index = self._hash(key)
    bucket = self.h[index]
    if bucket:
        if len(bucket) == 1 and bucket[0][0] == key:
            return bucket[0][1]
        else:
            # Collision handling
            for i in range(len(bucket)):
                if bucket[i][0] == key:
                    return bucket[i][1]
    return None
```

Verification that reporting is accurate and efficient

The application is able to show a listing of all packages and the current status of each package and all their attributes at any given time.

A screenshot of a terminal window titled "Run - NHP1" showing a table of package data. The table has columns for ID, Address, City, State, Zip Code, Deadline, Weight, Special Notes, and Status. The data is listed in 40 rows, showing various addresses, cities (SALT LAKE CITY, WEST VALLEY CITY, HOLLADAY, MURRAY), states (UT), zip codes, deadlines, weights, and status updates (e.g., "DELIVERED by SAM at 08:29", "AT HUB", "EN-ROUTE on Truck 02").

ID	Address	City	State	Zip Code	Deadline	Weight	Special Notes	Status
1	195 W OAKLAND AVE	SALT LAKE CITY	UT	84115	10:30 AM	21	N/A	DELIVERED by SAM at 08:29
2	2530 S 500 E	SALT LAKE CITY	UT	84106	EOD	44	N/A	DELIVERED by HIGGS at 09:23
3	233 CANYON RD	SALT LAKE CITY	UT	84103	EOD	2	Can only be on truck 2	DELIVERED by HIGGS at 10:10
4	300 W 2800 S	SALT LAKE CITY	UT	84115	EOD	4	N/A	DELIVERED by HIGGS at 09:17
5	410 S STATE ST	SALT LAKE CITY	UT	84111	EOD	5	N/A	DELIVERED by HIGGS at 10:07
6	3060 LESTER ST	WEST VALLEY CITY	UT	84119	10:30 AM	88	Delayed on flight---will not arrive to depot until 9:05 am	DELIVERED by HIGGS at 09:46
7	1330 2100 S	SALT LAKE CITY	UT	84106	EOD	8	N/A	DELIVERED by HIGGS at 09:28
8	300 STATE ST	SALT LAKE CITY	UT	84103	EOD	9	N/A	DELIVERED by HIGGS at 10:12
9	410 S STATE ST	SALT LAKE CITY	UT	84111	EOD	2	Fixed Address	AT HUB
10	600 E 900 S	SALT LAKE CITY	UT	84105	EOD	1	N/A	DELIVERED by HIGGS at 10:01
11	2600 TAYLORSVILLE BLVD	SALT LAKE CITY	UT	84118	EOD	1	N/A	EN-ROUTE on Truck 02
12	3575 W VALLEY CENTRAL STATION BUS LOOP	WEST VALLEY CITY	UT	84119	EOD	1	N/A	DELIVERED by HIGGS at 09:34
13	2010 W 500 S	SALT LAKE CITY	UT	84104	10:30 AM	2	N/A	DELIVERED by SAM at 09:10
14	4300 S 1300 E	MILLCREEK	UT	84117	10:30 AM	88	Must be delivered with 15, 19	DELIVERED by SAM at 08:06
15	4500 S 2300 E	HOLLADAY	UT	84117	09:00 AM	4	N/A	DELIVERED by SAM at 08:35
16	4500 S 2300 E	HOLLADAY	UT	84117	10:30 AM	88	Must be delivered with 13, 19	DELIVERED by SAM at 08:35
17	3140 S 1100 W	SALT LAKE CITY	UT	84119	EOD	2	N/A	DELIVERED by HIGGS at 09:42
18	1400 4000 S	SALT LAKE CITY	UT	84123	EOD	6	Can only be on truck 2	EN-ROUTE on Truck 02
19	177 W PRICE AVE	SALT LAKE CITY	UT	84115	EOD	37	N/A	DELIVERED by SAM at 08:14
20	3595 MAIN ST	SALT LAKE CITY	UT	84115	10:30 AM	37	Must be delivered with 13, 15	DELIVERED by SAM at 08:13
21	3595 MAIN ST	SALT LAKE CITY	UT	84115	EOD	3	N/A	DELIVERED by HIGGS at 09:11
22	6351 S 900 E	MURRAY	UT	84121	EOD	2	N/A	DELIVERED by HIGGS at 10:24
23	5100 S 2700 W	SALT LAKE CITY	UT	84118	EOD	5	N/A	AT HUB
24	5025 STATE ST	MURRAY	UT	84107	EOD	7	N/A	AT HUB
25	5383 S 900 E #104	SALT LAKE CITY	UT	84117	10:30 AM	7	Delayed on flight---will not arrive to depot until 9:05 am	AT HUB
26	5383 S 900 E #104	SALT LAKE CITY	UT	84117	EOD	25	N/A	AT HUB
27	1060 DALTON AVE S	SALT LAKE CITY	UT	84104	EOD	5	N/A	AT HUB
28	2835 MAIN ST	SALT LAKE CITY	UT	84115	EOD	7	Delayed on flight---will not arrive to depot until 9:05 am	AT HUB
29	1330 2100 S	SALT LAKE CITY	UT	84106	10:30 AM	2	N/A	DELIVERED by SAM at 08:45
30	300 STATE ST	SALT LAKE CITY	UT	84103	10:30 AM	1	N/A	DELIVERED by SAM at 09:00
31	3365 S 900 W	SALT LAKE CITY	UT	84119	10:30 AM	1	N/A	DELIVERED by SAM at 08:20
32	3365 S 900 W	SALT LAKE CITY	UT	84119	EOD	1	Delayed on flight---will not arrive to depot until 9:05 am	AT HUB
33	2530 S 500 E	SALT LAKE CITY	UT	84106	EOD	1	N/A	AT HUB
34	4500 S 2300 E	HOLLADAY	UT	84117	10:30 AM	2	N/A	DELIVERED by SAM at 08:35
35	1060 DALTON AVE S	SALT LAKE CITY	UT	84104	EOD	88	N/A	AT HUB
36	2300 PARKWAY BLVD	WEST VALLEY CITY	UT	84119	EOD	88	Can only be on truck 2	DELIVERED by HIGGS at 09:51
37	410 S STATE ST	SALT LAKE CITY	UT	84111	10:30 AM	2	N/A	DELIVERED by SAM at 08:56
38	410 S STATE ST	SALT LAKE CITY	UT	84111	EOD	9	Can only be on truck 2	DELIVERED by HIGGS at 10:07
39	2010 W 500 S	SALT LAKE CITY	UT	84104	EOD	9	N/A	AT HUB
40	300 W 2800 S	SALT LAKE CITY	UT	84115	10:30 AM	45	N/A	DELIVERED by SAM at 08:25

See section G1-G3 for all screenshots of the data output from the reporting function

K1A: Efficiency

The program implements a custom from-scratch Hashtable data structure. For more information on the Hashtable please read the following sections:

- Section D: Data Structure
- Section D: Explanation of Data Structure
- Section E: Hash Table
- Section F: Look-up Function

A hashtable is a type of data structure used for fast mapping and searching for items in a list via a hashing algorithm. An item's key is hashed to find a bucket index to map its value to, and that key hash can later be used to search for the same item, all at an average of $O(1)$ time efficiency, with a worst case of $O(n)$ if collision occurs.

For this particular scenario, the Hashtable will be used to store package data that has been parsed from a package CSV file. The hashtable will thus allow for quick inserting and lookup of package data for use in various parts of the application. A package logistics application such as this program will need to perform these actions many times, so having an quick data structure like a hashtable can make managing package data a lot more efficient.

The items mapped to this hashtable will be a tuple of a package id, and a package object (`p.get_package_id()`, `p`) . Since a package's id is unique, it can be used to uniquely identify a package as its key.

The package object contains all data for a specific package. A package object represents a real world package to be delivered. A package has a ID, destination address, city, state, zip code, a deadline, a weight, special notes, and a status. A package also has a history dictionary that takes a timestamp as a key, and a copy of the packages attributes at that time. This tracks the changing statuses of the package as it moves throughout its delivery route. All of which will be stored as a value in the hashtable.

K1B: Overhead

As stated in the previous section, the hashtable is implemented in a way that allows for a computational time complexity of $O(1)$ on average for inserting and looking-up data. If collision is present, this complexity changes to $O(n)$, however in this scenario we are able to implement a perfect hash formula resulting in zero collisions, so our time complexity stays constant.

Space complexity for the hashtable is $O(n)$ for both the average and worst cases.

The application is run from a local machine, so bandwidth and memory are of no concern and are not taken into consideration for this project.

K1C: Implications

The hashtable will be able to perform at constant time $o(1)$ as long as there are no collisions in buckets when new packages are added to the system. Collision occurs when the number of packages > number of empty buckets in the hashtable. The size of the hashtable can be easily changed during initialization using the `initial_size` parameter to prevent collision.

Adding new vertex locations and trucks do not directly affect the hashtable as they are implemented with other data structures such as graphs and lists.

K2: Other Data Structures

Instead of implementing a custom hashtable, other data structures that could potentially be used in this solutions include:

- 2D Arrays
- Dictionary
- Queues

K2A: Data Structure Differences

A **2D Array** could store package data and their attributes in a matrix style data structure, where each row would represent a package, and each column would be a package attribute. They are already fairly similar to a hashtable, with some hashtable implementations utilizing arrays for their base. The time complexity for accessing data would be $O(1)$ with known keys, similar to our hashtable. Array sizes are fixed so we would need to properly allocate size when creating the array for the first time. Hashtables don't need to worry about being sorted where with arrays, certain functions slow down when not sorted.

Dictionaries are another data structure that could be implemented in place of our hashtable. Mainly due to the fact that they are very similar in structure, using keys to access data. However, the main difference between a dictionary and a hashtable is that a dictionary does not use a hash function when mapping values to keys. Since we are simply using `package_ids` (int) types as keys, there is not much of a difference, but if we ever need to hash some non-generic value, a hashtable would be more useful.

L: Sources

This submission does not include any outside sources that were directly referenced.

M: Professional Communication

See 'this document' and source code for evidence to evaluate professional communication.