Jacob Sawyer

Student ID: 010397170

Submitted: 7/20/2023

A) Algorithm Identification

1 *Identified named self-adjusting algo that performs task of delivering all packages:*

- In order to algorithmically deliver packages, I chose to use the nearest neighbor algorithm. The principle of this algorithm is simple: find the minimum distance from a starting point to the next node, traverse that edge, then repeat until all nodes have been visited.
    - In my program, the function “deliver” in main.py accomplishes that.

    Deliver: function that works off an implementation of the nearest neighbor algorithm.

    - This function takes two parameters: truck and starting address. Truck is just the truck object we pass to it (truck 1 2 or 3) and the starting address is the location which the truck starts
    - We run a for loop to iterate through all packages on truck. We assign the distances from starting address to each of the different packages. We use the address_lookup function on both the start address each of the packages within the list. I talk more about this function in #3 above. We then note all of these distances in distanceList.
    - We then use an if statement. If there is more than 1 package in package list:
    - We assign a min distance variable. This just finds the lowest value in distanceList.
    - We also get the index of that minimum distance. This will be used to match up with the id of the package
    - We then look for the next package using that index. This will enable us to get the address of that package.
    - We delete the distance from distanceList and the value from package list based on that index. This enables us to not have to deal with that package/location anymore
    - We recursively call the deliver function, using the same truck parameter. We assign the startAddress to be the address of the next package.

    If there is only 1 package in list

    - We print that we are returning to hub
    - We add the distance from that last location back to the hub

This algorithm works exactly as intended and meets the requirements of nearest neighbor. **Code found in lines 259-296 of main.py.**

B1) Logical Comments

1 *Submission Explains algorithm's logic using pseudocode*
(All relevant pseudocode included in sections)

The nearest neightbor (knn) algorithm is a simple algorithm that looks at all available options and selects the next best option. It does not rely on predictions, making it very accurate.

Deliver (KNN):

In my code, we use the deliver(truck, startingAddress) function as the implementation of nearest neighbor- truck is the truck object passed (either truck 1, 2 or 3) and startingAddress is the starting address of the truck, "4001 South 700 East"

Following section is found in lines 259-271 of main.py:

Define function called deliver(pass truck object and pass starting address):

distanceList = empty list that will be populated with distances from starting address

for every id present in truck package list: [

p = the values associated with the id present (id is key in this case)

a1 = address 1 = the address_lookup function is called. (This function takes the current address (startingAddress) and finds the index from 1-27 since there are only 27 destinations for the 40 packages. More details below)

a2 = address 2 = the address_lookup function is called. (This function takes the next address (p.address) and finds the index from 1-27 since there are only 27 destinations for the 40 packages. More details below)

append every value found for the distance from the starting address to the rest of the addresses in the list. Uses findDistance function (more details below)

end for-loop]

Following section found in lines 273 – 296 (deliver function cont.) This section contains the if-else of the knn algorithm. The formula to calculate time taken was given in learning materials. It is [datetime.timedelta = hours (distance) / truck speed (18mph)]

If the length of the truck package list is greater than 1: [

minDist = minimum distance variable = the minimum value from the distanceList created in the above for loop

add the time taken to get from starting point to the new address.

minDistIndex = the index (spot in array) where the minimum distance is found. Since this list is a 1-1 length correlation with truck package list, we use this index to find the package associated with the distance

nextPackage = the new package we will look at. This is found using minDistIndex. The distance index points to the id of the end address

Next packages delivered time will be the updated time we calculated above

nextPackage will be set equal to the *address* of the new package

we delete both the index from minDistIndex and the value from truck.packages so the algorithm will not back track

end if statement]

else: (this will only be ran when there is one package left and we are going to its location)[

truck distance traveled will be equal to the distance from second to last package to the last package (using findDistance, more details below), plus the previous distance total.

package delivered time is equal to the distance / 18

end else]

address_lookup (ancillary function): This function is used to calculate the address index. The index is provided for us in addressFile.py. These indices line up with the coordinates in distanceFile.csv

following code is found in lines 221-230 of main.py:

defines address_lookup(address is passed to this function):

with addressFile.csv open, we will call it addressFileImport: [

assign each row of the csv file to a list inside a list, looks like this: [[x, x,..], [y, y,…]

end with]

for each list(call it x) inside of this list called addressList: [

if there is an address in x, return the index plus one of the address list. This is

because the address.csv file is indexed 1-27, not 0-26

end for-loop]

findDistance (ancillary function): this function is an ancillary of deliver.py to find distances between packages

code found in lines 235-248 of main.py

with distanceFile.csv file open, call it distanceFileImport: [

create a distance file list with each row of distanceFile.csv

end with]

create a distance variable. It is equal to distance list at the coordinates x + 1, y + 1

if there is no value at the coordinate: [

flip coordinates and do the same (y + 1, x + 1)

end if]

As mentioned in the code, the findDistance and address_lookup function were both recommended by professor Lusby's study guide. These functions are necessary ancillaries for deliver method to work.

B2) Development Environment

1 Submission accurately describes hardware:

Device name	DESKTOP-NSJIFEG
Processor	Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz 2.11 GHz
Installed RAM	16.0 GB (15.8 GB usable)
Device ID	76FFBABF-3A77-4615-ADF0-7D88A4C12EFB
Product ID	00330-51907-37651-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

2 and software to build python applications:

Windows:

Edition	Windows 11 Pro
Version	22H2
Installed on	12/29/2022
OS build	22621.1848
Experience	Windows Feature Experience Pack 1000.22642.1000.0

Visual Studio Code:

Visual Studio Code

Version: 1.80.0 (user setup)
Commit: 660393deaaa6d1996740ff4880f1bad43768c814
Date: 2023-07-04T15:06:02.407Z (1 wk ago)
Electron: 22.3.14
ElectronBuildId: 21893604
Chromium: 108.0.5359.215
Node.js: 16.17.1
V8: 10.8.168.25-electron.0
OS: Windows_NT x64 10.0.22621

B3) Space-Time and Big O

1 Submission accurately shows spacetime Complexity of each segment and entire program using Big O Notation.

OVERALL: O(n^2) – main.py line 15

Main.Py:

Class: complexity time/ space = O(n^2) – main.py line 16

Def loadPkgInHash(): complexity time/ space = O(n) – main.py line 22

Def loadTruck1(): complexity time/ space = O(n) – main.py line 61

Def loadTruck2(): complexity time/ space = O(n) – main.py line 115

Def loadTruck3(): complexity time/ space = O(n) – main.py line: 187

Def address_lookup(): complexity time/ space = O(n) – main.py line 219

Def findDistance(): complexity time/ space = O(n) – main.py line 233

Def deliver: (): complexity time/ space = O(n^2) – main.py line 258 → this is nearest neighbor implementation.

Def t1Deliver(): complexity time/ space = O(1) – main.py line 306

Def t2Deliver(): complexity time/ space = O(1) – main.py line 323

Def t3Deliver(): complexity time/ space = O(1) – main.py line 3340

Hash.py:

Class: O(n) – hash.py line 7

Def __init__(): O(n), space O(n)– hash.py line 11

Def insert(): time: O(n), space O(1) – hash.py line 21

Def lookup(): O(n), space O(1) – hash.py line 42

Def remove(): O(n)space O(1) – hash.py line 55

Truck.py:

Class: complexity time/ space = O(1) – truck.py line 7

Package.py:

Class: complexity time/ space = O(1) – package.py line 10

Def GetStatus(): complexity time/ space = O(1) - package.py line 25

Def GetAddress(): complexity time/ space = O(1) - package.py line 29

Def updatedPackage():complexity time/ space = O(1) – package.py line 33

Interface.py:

Class: complexity time/ space = O(n) – interface.py line 10

Def prompt: complexity time/ space = O(1) - interface.py line 13

Def loadingTruckInterface: complexity time/ space = O(1) – interface.py line 22

Def output(): complexity time/ space = O(n) - interface.py line 31

B4) Scalability and Adaptability

1 Submission explains the capability of the solution to adapt and scale logically:

Since there is very minimal hardcodes portions of this code (only package loading parameters and the number of trucks and their attributes), it is very efficient at scaling. This is not a particular strength of hashmaps (they are very good at known data set sizes). So in order to scale even better, you would at some point need to initialize a bigger size list for buckets in the hashmap.

However, since there are parameters that are handled within the nearest neighbor algorithm, this is very well suited for increasing data sizes. The only limitation would be the number of trucks and their capacities (& adjusting those is out of scope of

this code). Overall, this program should be able to handle scaling very well. None of the packages needed to be hard coded into the project.

B5) Software Efficiency and Maintainability

1 Submission explains why the software is both efficient:

This program is extremely efficient in processing inputs. This is accomplished by using a hashmap to store package data points. Since hashmaps have a very low lookup, insert, and delete time complexity (only an average of O(1), the rare worst case being O(n)), it operates very quickly. The only function in the program grows at a higher complexity rate than O(n) is our nearest neighbor method, deliver(). This function is typical to be O(n^2) for using two for loops. This implementation is called recursively though. This recursive call is the reason for the O(n^2) runtime. Overall, this program is very efficient at processing inputs. The interface and the rest of the program maintains a very low time complexity and space complexity.

The hashmap is a very intuitive way of storing packages for easy access, as opposed to a stack or a different data structure. The key-value matching is the ideal concept for a hashmap and since we don't need to worry about in-order listing (we can store the packages in any order we please). And as mentioned, hashmaps have a very quick access and deletion time.

2 and easy to maintain:

As mentioned, since the program is not hard coded in any way, it is very scalable

And maintainable. The extend you would need to modify the program is to add or subtract trucks, modify attributes, or change package parameters. The algorithm and hashmap all work very well and systematically with the csv files.

In the future, if changes needed to be made to the program, it would be simple as modifying the attributes or the for loops inside of main.py. If I were to make changes in the future, it would be to add a GUI to modify inputs as well.

This program is extremely efficient in processing inputs. Since there is no hardcoding of data, it can scale very easily and. The nearest neighbor algorithm is also a great choice for finding the optimal route since it is simple to understand but also very optimal (for a single run K-NN is only O(n^2) time complexity).

B6) Self-Adjusting Data Structures

1 Submission addresses both the strengths:

Hashmaps are *extremely* versatile and can handle a wide range of data storing needs. Since data retrieval and insertion operators are both an average O(1), there are few better methods of data storage and retrieval. On top of that, you can easily change the initial load size/ capacity with a simple number change. This can prevent resizing issues down the road when you're dealing with larger data sets.

2 and weaknesses of self-adjusting data structure/hash table

Hashmaps are a great tool, but also come with a couple of drawbacks. A big one is that as you scale data size, hashmaps may have trouble accommodating the shift in resizing. A way to mitigate this would be to increase initial load capacity. Another draw back is there's no specific order of keys or values. This is really only an issue if you need to maintain a specific order of the hashmap for some reason. Finally, hashmaps are not great in multithreaded processes. When more than 1 thread accesses a hashmap concurrently, you may get unintended results

C) Original Code

1 Code is original and runs without error or warnings:

Code is original, written entirely myself except for using hashmap in learning material. Runs with zero errors or warnings. ID at top of main.py and comments all throughout code

C1) Identification Information:

1 Identifying comment at top of code

At top of main.py you'll find my name and student ID

C2) Process and Flow Comments

1 Comments are provided within code that accurately describes the code flow

There is a line of comments at nearly every line of code

D) Data Structure

1 The submission identifies a self adjusting data structure that performs well with

algorithm in part A and can store package data

The hashmap used in this program works perfectly for storing data. The code was adopted from the learning material (WGU Page and zybooks). The code Is all located in

hash.py. I will go over each section of code here and how it works (more about it working with KNN at bottom).

Section 1. Lines 13-19 hash.py __INIT__

We defined the initialization of the hashmap here. We pass in “self” and the initial capacity.

We then create a table of self which is essentially a blank list []

For each element in range if initial capacity, we append a empty list. Since our initial capacity was only 1, we append 1 bracket set = [[]]

Section 2. Lines 24-39 hash.py INSERT

We defined the insert function here. This function also updates the hashmap. We pass in self, and a key value pair.

We then create the hash bucket, which is just the hashed key divided by the length of our array

We use this new bucket to create our bucket list. This is repeated for every key value pair we insert

To update, for each keyvalue in the bucket list:

If there exists a key in the 0^{th} index, then update the new value and then return true.

Alternatively, if we are inserting a new kv pair:

Key_value = the array of key value in the format [key, value]

We then append the key value to the bucket list and return True

Section 3: lines 43-53 hash.py LOOKUP

We define the lookup function here by passing in self and key.

We create the bucket by hashing the key and dividing it by length of the array/table

Next we create the bucket list by adding it to the index of self.table

For every key value pair in bucket list,

If there exists a key, then return the value. We don't return anything else

Section 4: lines 56 – 65 hash.py REMOVE

We define remove here. This takes the same parameters as the prior two functions (insert and lookup)

This function creates the bucket and bucket list in the same way as insert and lookup.

For each key value pair,

If the key exists,

Remove the key and key value

This data structure is perfect for working with nearest neighbor. The KNN algorithm, pulls each key's value when necessary, but the indexing (the keys) is usable for most of the algorithm. The Key is the package ID and the value is the rest of the attributes of each package.

D1) Explanation of Data Structure

1 Submission accurately explains the data structure and how it accounts for the relationship between the data points being stored.

The data structure used in this program is a hashmap. Hashmaps are perfect for

This problem because each package has a unique ID associated with the rest of the package attributes. This, combined with its simplicity and the fast big O runtime for each of its functions, makes it a great choice. On average a hashmap can perform its functions in O(1) with O(n) being the worst case.

The data points being stored are A) the keys and B) the values (attributes of package). This coincides with how hashmaps operate, especially since each package 1-40 has a unique key assigned to it.

E) Hash Table

1 Hash table has an insertion function without using additional libraries or class. Free of

errors and include all components.

The insertion function can be found in lines 24-39 of hash.py

We defined the insert function here. This function also updates the hashmap. We pass in self, and a key value pair.

We then create the hash bucket, which is just the hashed key divided by the length of our array

We use this new bucket to create our bucket list. This is repeated for every key value pair we insert

To update, for each keyvalue in the bucket list:

If there exists a key in the 0^{th} index, then update the new value and then return true.

Alternatively, if we are inserting a new kv pair:

Key_value = the array of key value in the format [key, value]

We then append the key value to the bucket list and return True

Screenshot:

```
def insert(self, key, value): #  does both insert and update
    # get the bucket list where this item will go.
    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] = value
        return True

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

F) Lookup

The lookup function in the hash.py module is derived directly from *C950 Webinar-1 – Let's go Hashing – Complete Python Code.* (This is listed in sources below with a

link). Asked Professor Ligocki for permission on using this. Including screenshot of code and screenshot of test lookup. Works exactly as outlined in rubric.

Located at lines 43-53 of hash.py. more details in section D about lookup

Test:

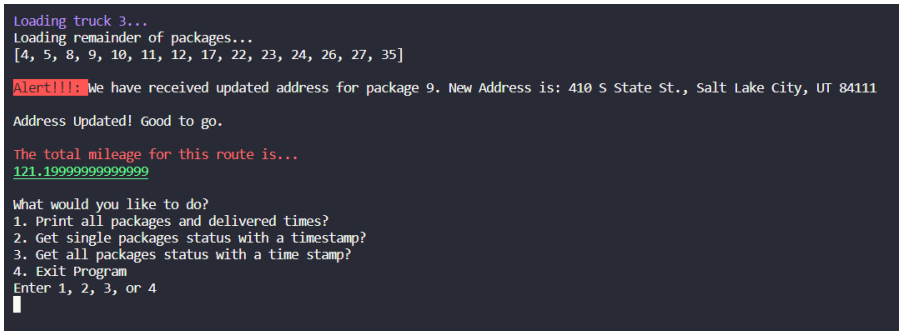
```
print(hash_table_init.lookup(15))
```

```
packageObject(ID=15, address='4580 S 2300 E', city='Holladay', state='UT', zipcode='8
4117', Deadline_time='9:00 AM', weight='4', status='')
PS C:\Users\jacob\Desktop\DSA2Program>
```

G) Interface

1 *Interface provides intuitive means of both determining the total mileage by all trucks*

Screenshot of interface:



The interface is very intuitive to use and stylized to be easier to read. As shown in the screenshot, it shows the mileage for all trucks combined, as well as each truck individually. Screenshots in this section are from option 1 in the interface (option 1 is lines 54-106 of interface.py)

Truck 3: Line 104 of interface.py

```
Status: Delivered

Truck 3 Distance after getting back to Hub: 46.5
Time returned to Hub: 13:05:00
```

Truck 2: Line 87 of interface.py

```
Status: Delivered

Truck 2 Distance after getting back to Hub: 40.4
Time returned to Hub: 11:14:40

Packages on Truck 3
```

Truck 1: Line 70 of interface.py

```
Status: Delivered

Truck 1 Distance after getting back to Hub: 34.3
Time returned to Hub: 9:54:20


Packages on Truck 2
```

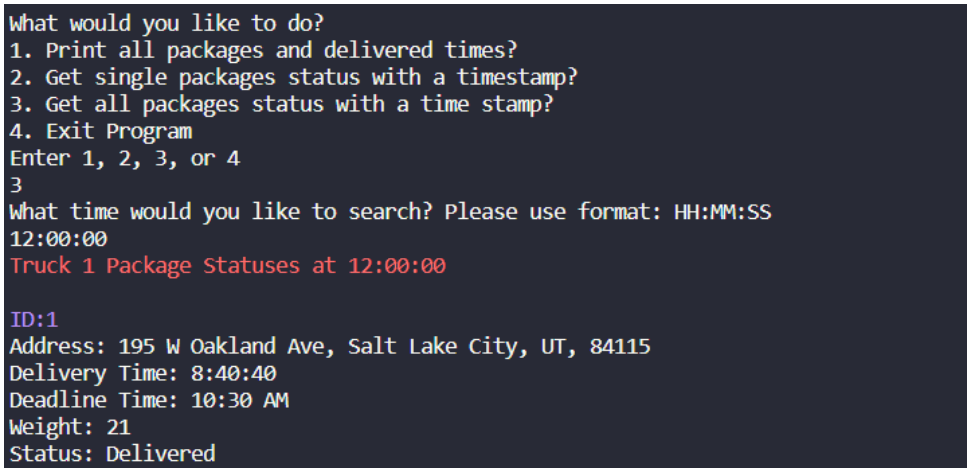
2 *and for accessing package delivery status as required and includes time of status check*

Option 2 controls single time single package (lines 111-177 of interface.py) and option 3 controls all packages single time (lines 182-282 of interface.py).

Option 2 screenshot:

```
What would you like to do?
1. Print all packages and delivered times?
2. Get single packages status with a timestamp?
3. Get all packages status with a time stamp?
4. Exit Program
Enter 1, 2, 3, or 4
2
What time would you like to search? Please use format: HH:MM:SS
11:00:00
What package would you like to search? Please enter a number 1-40.
21
Package is on truck 1
Package 21 status at 11:00:00:

ID:21
Address: 3595 Main St, Salt Lake City, UT, 84115
Delivery Time: 8:29:40
Deadline Time: EOD
Weight: 3
Status: Delivered

PS C:\Users\jacob\Desktop\DSA2Program>
```

Option 3 screenshot:



G1) First Status Check

Screenshot of all packages at a time between 08:35 and 09:25

Chose 09:00:

```
3
What time would you like to search? Please use format: HH:MM:SS
09:00:00
Truck 1 Package Statuses at 9:00:00

ID:1
Address: 195 W Oakland Ave, Salt Lake City, UT, 84115
Delivery Time: 8:40:40
Deadline Time: 10:30 AM
Weight: 21
Status: Delivered

ID:2
Address: 2530 S 500 E, Salt Lake City, UT, 84106
Delivery Time: 8:45:40
Deadline Time: EOD
Weight: 44
Status: Delivered

ID:13
Address: 2010 W 500 S, Salt Lake City, UT, 84104
Deadline Time: 10:30 AM
Weight: 2
Status: En Route

ID:14
Address: 4300 S 1300 E, Millcreek, UT, 84117
Delivery Time: 8:06:20
Deadline Time: 10:30 AM
Weight: 88
Status: Delivered

ID:15
Address: 4580 S 2300 E, Holladay, UT, 84117
Delivery Time: 8:13:00
Deadline Time: 9:00 AM
Weight: 4
Status: Delivered

ID:16
Address: 4580 S 2300 E, Holladay, UT, 84117
Delivery Time: 8:13:00
Deadline Time: 10:30 AM
Weight: 88
Status: Delivered

ID:19
Address: 177 W Price Ave, Salt Lake City, UT, 84115
Delivery Time: 8:31:20
Deadline Time: EOD
Weight: 37
Status: Delivered

ID:20
Address: 3595 Main St, Salt Lake City, UT, 84115
Delivery Time: 8:29:40
Deadline Time: 10:30 AM
Weight: 37
Status: Delivered

ID:29
Address: 1330 2100 S, Salt Lake City, UT, 84106
Delivery Time: 8:51:00
Deadline Time: 10:30 AM
Weight: 2
Status: Delivered

ID:30
Address: 300 State St, Salt Lake City, UT, 84103
Deadline Time: 10:30 AM
Weight: 1
Status: En Route
```

```
ID:31
Address: 3365 S 900 W, Salt Lake City, UT, 84119
Deadline Time: 10:30 AM
Weight: 1
Status: En Route

ID:37
Address: 410 S State St., Salt Lake City, UT, 84111
Deadline Time: 10:30 AM
Weight: 2
Status: En Route

ID:40
Address: 380 W 2880 S, Salt Lake City, UT, 84115
Delivery Time: 8:37:00
Deadline Time: 10:30 AM
Weight: 45
Status: Delivered

ID:21
Address: 3595 Main St, Salt Lake City, UT, 84115
Delivery Time: 8:29:40
Deadline Time: EOD
Weight: 3
Status: Delivered

ID:34
Address: 4580 S 2300 E, Holladay, UT, 84117
Delivery Time: 8:13:00
Deadline Time: 10:30 AM
Weight: 2
Status: Delivered

ID:39
Address: 2010 W 500 S, Salt Lake City, UT, 84104
Deadline Time: EOD
Weight: 9
Status: En Route

Truck 2 Package Statuses at 9:00:00

ID:3
Address: 233 Canyon Rd, Salt Lake City, UT, 84103
Deadline Time: EOD
Weight: 2
Status: En Route

ID:18
Address: 1488 4800 S, Salt Lake City, UT, 84123
Deadline Time: EOD
Weight: 6
Status: En Route

ID:36
Address: 2300 Parkway Blvd, West Valley City, UT, 84119
Deadline Time: EOD
Weight: 88
Status: En Route

ID:38
Address: 410 S State St., Salt Lake City, UT, 84111
Deadline Time: EOD
Weight: 9
Status: En Route

ID:6
Address: 3060 Lester St, West Valley City, UT, 84119
Deadline Time: 10:30 AM
Weight: 88
Status: En Route

ID:25
Address: 5383 South 900 East #104, Salt Lake City, UT, 84117
Deadline Time: 10:30 AM
Weight: 7
Status: En Route
```

```
ID:28
Address: 2835 Main St, Salt Lake City, UT, 84115
Deadline Time: EOD
Weight: 7
Status: En Route

ID:32
Address: 3365 S 900 W, Salt Lake City, UT, 84119
Deadline Time: EOD
Weight: 1
Status: En Route

ID:7
Address: 1330 2100 S, Salt Lake City, UT, 84106
Deadline Time: EOD
Weight: 8
Status: En Route

ID:33
Address: 2530 S 500 E, Salt Lake City, UT, 84106
Deadline Time: EOD
Weight: 1
Status: En Route

Truck 3 Package Statuses at 9:00:00

ID:4
Address: 380 W 2880 S, Salt Lake City, UT, 84115
Deadline Time: EOD
Weight: 4
Status: At hub waiting to leave

ID:5
Address: 410 S State St., Salt Lake City, UT, 84111
Deadline Time: EOD
Weight: 5
Status: At hub waiting to leave

ID:8
Address: 300 State St, Salt Lake City, UT, 84103
Deadline Time: EOD
Weight: 9
Status: At hub waiting to leave

ID:9
Address: 410 S State St., Salt Lake City, UT, 84111
Deadline Time: EOD
Weight: 2
Status: At hub waiting to leave

ID:10
Address: 600 E 900 South, Salt Lake City, UT, 84105
Deadline Time: EOD
Weight: 1
Status: At hub waiting to leave

ID:11
Address: 2600 Taylorsville Blvd, Salt Lake City, UT, 84118
Deadline Time: EOD
Weight: 1
Status: At hub waiting to leave

ID:12
Address: 3575 W Valley Central Station bus Loop, West Valley City, UT, 84119
Deadline Time: EOD
Weight: 1
Status: At hub waiting to leave
```

```
ID:17
Address: 3148 S 1100 W, Salt Lake City, UT, 84119
Deadline Time: EOD
Weight: 2
Status: At hub waiting to leave

ID:22
Address: 6351 South 900 East, Murray, UT, 84121
Deadline Time: EOD
Weight: 2
Status: At hub waiting to leave

ID:23
Address: 5100 South 2700 West, Salt Lake City, UT, 84118
Deadline Time: EOD
Weight: 5
Status: At hub waiting to leave

ID:24
Address: 5025 State St, Murray, UT, 84107
Deadline Time: EOD
Weight: 7
Status: At hub waiting to leave

ID:26
Address: 5383 South 900 East #104, Salt Lake City, UT, 84117
Deadline Time: EOD
Weight: 25
Status: At hub waiting to leave

ID:27
Address: 1060 Dalton Ave S, Salt Lake City, UT, 84104
Deadline Time: EOD
Weight: 5
Status: At hub waiting to leave

ID:35
Address: 1060 Dalton Ave S, Salt Lake City, UT, 84104
Deadline Time: EOD
Weight: 88
Status: At hub waiting to leave

PS C:\Users\jacob\Desktop\DSA2Program>
```

G2) Second Status Check

Screenshot of all packages at a time between 09:35 and 10:25

Chose 10:00:

```
What time would you like to search? Please use format: HH:MM:SS
10:00:00
Truck 1 Package Statuses at 10:00:00

ID:1
Address: 195 W Oakland Ave, Salt Lake City, UT, 84115
Delivery Time: 8:40:40
Deadline Time: 10:30 AM
Weight: 21
Status: Delivered

ID:2
Address: 2530 S 500 E, Salt Lake City, UT, 84106
Delivery Time: 8:45:40
Deadline Time: EOD
Weight: 44
Status: Delivered

ID:13
Address: 2010 W 500 S, Salt Lake City, UT, 84104
Delivery Time: 9:22:40
Deadline Time: 10:30 AM
Weight: 2
Status: Delivered

ID:14
Address: 4300 S 1300 E, Millcreek, UT, 84117
Delivery Time: 8:06:20
Deadline Time: 10:30 AM
Weight: 88
Status: Delivered

ID:15
Address: 4580 S 2300 E, Holladay, UT, 84117
Delivery Time: 8:13:00
Deadline Time: 9:00 AM
Weight: 4
Status: Delivered

ID:16
Address: 4580 S 2300 E, Holladay, UT, 84117
Delivery Time: 8:13:00
Deadline Time: 10:30 AM
Weight: 88
Status: Delivered

ID:19
Address: 177 W Price Ave, Salt Lake City, UT, 84115
Delivery Time: 8:31:20
Deadline Time: EOD
Weight: 37
Status: Delivered

ID:20
Address: 3595 Main St, Salt Lake City, UT, 84115
Delivery Time: 8:29:40
Deadline Time: 10:30 AM
Weight: 37
Status: Delivered

ID:29
Address: 1330 2100 S, Salt Lake City, UT, 84106
Delivery Time: 8:51:00
Deadline Time: 10:30 AM
Weight: 2
Status: Delivered

ID:30
Address: 300 State St, Salt Lake City, UT, 84103
Delivery Time: 9:08:40
Deadline Time: 10:30 AM
Weight: 1
Status: Delivered
```

```
ID:31
Address: 3365 S 900 W, Salt Lake City, UT, 84119
Delivery Time: 9:42:00
Deadline Time: 10:30 AM
Weight: 1
Status: Delivered

ID:37
Address: 410 S State St., Salt Lake City, UT, 84111
Delivery Time: 9:05:20
Deadline Time: 10:30 AM
Weight: 2
Status: Delivered

ID:40
Address: 380 W 2880 S, Salt Lake City, UT, 84115
Delivery Time: 8:37:00
Deadline Time: 10:30 AM
Weight: 45
Status: Delivered

ID:21
Address: 3595 Main St, Salt Lake City, UT, 84115
Delivery Time: 8:29:40
Deadline Time: EOD
Weight: 3
Status: Delivered

ID:34
Address: 4580 S 2300 E, Holladay, UT, 84117
Delivery Time: 8:13:00
Deadline Time: 10:30 AM
Weight: 2
Status: Delivered

ID:39
Address: 2010 W 500 S, Salt Lake City, UT, 84104
Delivery Time: 9:22:40
Deadline Time: EOD
Weight: 9
Status: Delivered

Truck 2 Package Statuses at 10:00:00

ID:3
Address: 233 Canyon Rd, Salt Lake City, UT, 84103
Deadline Time: EOD
Weight: 2
Status: En Route

ID:18
Address: 1488 4800 S, Salt Lake City, UT, 84123
Deadline Time: EOD
Weight: 6
Status: En Route

ID:36
Address: 2300 Parkway Blvd, West Valley City, UT, 84119
Delivery Time: 9:47:40
Deadline Time: EOD
Weight: 88
Status: Delivered

ID:38
Address: 410 S State St., Salt Lake City, UT, 84111
Deadline Time: EOD
Weight: 9
Status: En Route
```

```
ID:6
Address: 3060 Lester St, West Valley City, UT, 84119
Delivery Time: 9:42:20
Deadline Time: 10:30 AM
Weight: 88
Status: Delivered

ID:25
Address: 5383 South 900 East #104, Salt Lake City, UT, 84117
Delivery Time: 9:08:00
Deadline Time: 10:30 AM
Weight: 7
Status: Delivered

ID:28
Address: 2835 Main St, Salt Lake City, UT, 84115
Delivery Time: 9:27:40
Deadline Time: EOD
Weight: 7
Status: Delivered

ID:32
Address: 3365 S 900 W, Salt Lake City, UT, 84119
Delivery Time: 9:37:20
Deadline Time: EOD
Weight: 1
Status: Delivered

ID:7
Address: 1330 2100 S, Salt Lake City, UT, 84106
Deadline Time: EOD
Weight: 8
Status: En Route

ID:33
Address: 2530 S 500 E, Salt Lake City, UT, 84106
Delivery Time: 9:24:00
Deadline Time: EOD
Weight: 1
Status: Delivered

Truck 3 Package Statuses at 10:00:00

ID:4
Address: 380 W 2880 S, Salt Lake City, UT, 84115
Deadline Time: EOD
Weight: 4
Status: At hub waiting to leave

ID:5
Address: 410 S State St., Salt Lake City, UT, 84111
Deadline Time: EOD
Weight: 5
Status: At hub waiting to leave

ID:8
Address: 300 State St, Salt Lake City, UT, 84103
Deadline Time: EOD
Weight: 9
Status: At hub waiting to leave

ID:9
Address: 410 S State St., Salt Lake City, UT, 84111
Deadline Time: EOD
Weight: 2
Status: At hub waiting to leave

ID:10
Address: 600 E 900 South, Salt Lake City, UT, 84105
Deadline Time: EOD
Weight: 1
Status: At hub waiting to leave
```

```
ID:11
Address: 2600 Taylorsville Blvd, Salt Lake City, UT, 84118
Deadline Time: EOD
Weight: 1
Status: At hub waiting to leave

ID:12
Address: 3575 W Valley Central Station bus Loop, West Valley City, UT, 84119
Deadline Time: EOD
Weight: 1
Status: At hub waiting to leave

ID:17
Address: 3148 S 1100 W, Salt Lake City, UT, 84119
Deadline Time: EOD
Weight: 2
Status: At hub waiting to leave

ID:22
Address: 6351 South 900 East, Murray, UT, 84121
Deadline Time: EOD
Weight: 2
Status: At hub waiting to leave

ID:23
Address: 5100 South 2700 West, Salt Lake City, UT, 84118
Deadline Time: EOD
Weight: 5
Status: At hub waiting to leave

ID:24
Address: 5025 State St, Murray, UT, 84107
Deadline Time: EOD
Weight: 7
Status: At hub waiting to leave

ID:26
Address: 5383 South 900 East #104, Salt Lake City, UT, 84117
Deadline Time: EOD
Weight: 25
Status: At hub waiting to leave

ID:27
Address: 1060 Dalton Ave S, Salt Lake City, UT, 84104
Deadline Time: EOD
Weight: 5
Status: At hub waiting to leave

ID:35
Address: 1060 Dalton Ave S, Salt Lake City, UT, 84104
Deadline Time: EOD
Weight: 88
Status: At hub waiting to leave

PS C:\Users\jacob\Desktop\DSA2Program>
```

G3) Third Status Check

Screenshot of all packages at a time between 12:03 and 13:12

Chose 12:30:

```
What time would you like to search? Please use format: HH:MM:SS
12:30:00
Truck 1 Package Statuses at 12:30:00

ID:1
Address: 195 W Oakland Ave, Salt Lake City, UT, 84115
Delivery Time: 8:40:40
Deadline Time: 10:30 AM
Weight: 21
Status: Delivered

ID:2
Address: 2530 S 500 E, Salt Lake City, UT, 84106
Delivery Time: 8:45:40
Deadline Time: EOD
Weight: 44
Status: Delivered

ID:13
Address: 2010 W 500 S, Salt Lake City, UT, 84104
Delivery Time: 9:22:40
Deadline Time: 10:30 AM
Weight: 2
Status: Delivered

ID:14
Address: 4300 S 1300 E, Millcreek, UT, 84117
Delivery Time: 8:06:20
Deadline Time: 10:30 AM
Weight: 88
Status: Delivered

ID:15
Address: 4580 S 2300 E, Holladay, UT, 84117
Delivery Time: 8:13:00
Deadline Time: 9:00 AM
Weight: 4
Status: Delivered

ID:16
Address: 4580 S 2300 E, Holladay, UT, 84117
Delivery Time: 8:13:00
Deadline Time: 10:30 AM
Weight: 88
Status: Delivered

ID:19
Address: 177 W Price Ave, Salt Lake City, UT, 84115
Delivery Time: 8:31:20
Deadline Time: EOD
Weight: 37
Status: Delivered

ID:20
Address: 3595 Main St, Salt Lake City, UT, 84115
Delivery Time: 8:29:40
Deadline Time: 10:30 AM
Weight: 37
Status: Delivered

ID:29
Address: 1330 2100 S, Salt Lake City, UT, 84106
Delivery Time: 8:51:00
Deadline Time: 10:30 AM
Weight: 2
Status: Delivered

ID:30
Address: 300 State St, Salt Lake City, UT, 84103
Delivery Time: 9:08:40
Deadline Time: 10:30 AM
Weight: 1
Status: Delivered

ID:31
Address: 3365 S 900 W, Salt Lake City, UT, 84119
Delivery Time: 9:42:00
```

```
ID:31
Address: 3365 S 900 W, Salt Lake City, UT, 84119
Delivery Time: 9:42:00
Deadline Time: 10:30 AM
Weight: 1
Status: Delivered

ID:37
Address: 410 S State St., Salt Lake City, UT, 84111
Delivery Time: 9:05:20
Deadline Time: 10:30 AM
Weight: 2
Status: Delivered

ID:40
Address: 380 W 2880 S, Salt Lake City, UT, 84115
Delivery Time: 8:37:00
Deadline Time: 10:30 AM
Weight: 45
Status: Delivered

ID:21
Address: 3595 Main St, Salt Lake City, UT, 84115
Delivery Time: 8:29:40
Deadline Time: EOD
Weight: 3
Status: Delivered

ID:34
Address: 4580 S 2300 E, Holladay, UT, 84117
Delivery Time: 8:13:00
Deadline Time: 10:30 AM
Weight: 2
Status: Delivered

ID:39
Address: 2010 W 500 S, Salt Lake City, UT, 84104
Delivery Time: 9:22:40
Deadline Time: EOD
Weight: 9
Status: Delivered

Truck 2 Package Statuses at 12:30:00

ID:3
Address: 233 Canyon Rd, Salt Lake City, UT, 84103
Delivery Time: 10:49:20
Deadline Time: EOD
Weight: 2
Status: Delivered

ID:18
Address: 1488 4800 S, Salt Lake City, UT, 84123
Delivery Time: 10:01:00
Deadline Time: EOD
Weight: 6
Status: Delivered

ID:36
Address: 2300 Parkway Blvd, West Valley City, UT, 84119
Delivery Time: 9:47:40
Deadline Time: EOD
Weight: 88
Status: Delivered

ID:38
Address: 410 S State St., Salt Lake City, UT, 84111
Delivery Time: 10:46:00
Deadline Time: EOD
Weight: 9
Status: Delivered
```

```
ID:6
Address: 3060 Lester St, West Valley City, UT, 84119
Delivery Time: 9:42:20
Deadline Time: 10:30 AM
Weight: 88
Status: Delivered

ID:25
Address: 5383 South 900 East #104, Salt Lake City, UT, 84117
Delivery Time: 9:08:00
Deadline Time: 10:30 AM
Weight: 7
Status: Delivered

ID:28
Address: 2835 Main St, Salt Lake City, UT, 84115
Delivery Time: 9:27:40
Deadline Time: EOD
Weight: 7
Status: Delivered

ID:32
Address: 3365 S 900 W, Salt Lake City, UT, 84119
Delivery Time: 9:37:20
Deadline Time: EOD
Weight: 1
Status: Delivered

ID:7
Address: 1330 2100 S, Salt Lake City, UT, 84106
Delivery Time: 10:31:40
Deadline Time: EOD
Weight: 8
Status: Delivered

ID:33
Address: 2530 S 500 E, Salt Lake City, UT, 84106
Delivery Time: 9:24:00
Deadline Time: EOD
Weight: 1
Status: Delivered

Truck 3 Package Statuses at 12:30:00

ID:4
Address: 380 W 2880 S, Salt Lake City, UT, 84115
Delivery Time: 11:35:20
Deadline Time: EOD
Weight: 4
Status: Delivered

ID:5
Address: 410 S State St., Salt Lake City, UT, 84111
Delivery Time: 11:55:40
Deadline Time: EOD
Weight: 5
Status: Delivered

ID:8
Address: 300 State St, Salt Lake City, UT, 84103
Delivery Time: 11:59:00
Deadline Time: EOD
Weight: 9
Status: Delivered

ID:9
Address: 410 S State St., Salt Lake City, UT, 84111
Delivery Time: 11:55:40
Deadline Time: EOD
Weight: 2
Status: Delivered
```

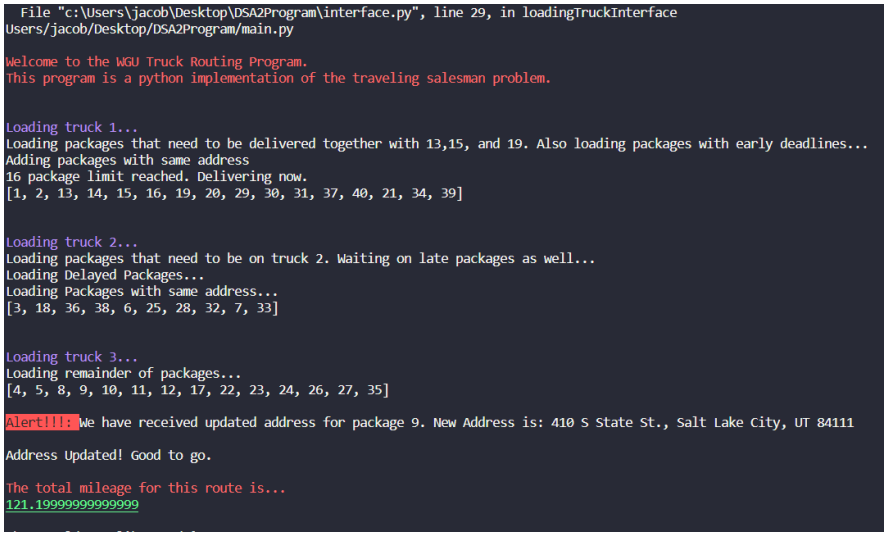
```
ID:10
Address: 600 E 900 South, Salt Lake City, UT, 84105
Delivery Time: 11:49:40
Deadline Time: EOD
Weight: 1
Status: Delivered

ID:11
Address: 2600 Taylorsville Blvd, Salt Lake City, UT, 84118
Delivery Time: 11:10:40
Deadline Time: EOD
Weight: 1
Status: Delivered

ID:12
Address: 3575 W Valley Central Station bus Loop, West Valley City, UT, 84119
Deadline Time: EOD
Weight: 1
Status: En Route

ID:17
Address: 3148 S 1100 W, Salt Lake City, UT, 84119
Delivery Time: 11:28:00
Deadline Time: EOD
Weight: 2
Status: Delivered

ID:22
Address: 6351 South 900 East, Murray, UT, 84121
Delivery Time: 10:48:00
Deadline Time: EOD
Weight: 2
Status: Delivered

ID:23
Address: 5100 South 2700 West, Salt Lake City, UT, 84118
Delivery Time: 11:12:00
Deadline Time: EOD
Weight: 5
Status: Delivered

ID:24
Address: 5025 State St, Murray, UT, 84107
Delivery Time: 10:38:00
Deadline Time: EOD
Weight: 7
Status: Delivered

ID:26
Address: 5383 South 900 East #104, Salt Lake City, UT, 84117
Delivery Time: 10:43:40
Deadline Time: EOD
Weight: 25
Status: Delivered

ID:27
Address: 1060 Dalton Ave S, Salt Lake City, UT, 84104
Delivery Time: 12:15:00
Deadline Time: EOD
Weight: 5
Status: Delivered

ID:35
Address: 1060 Dalton Ave S, Salt Lake City, UT, 84104
Delivery Time: 12:15:00
Deadline Time: EOD
Weight: 88
Status: Delivered

PS C:\Users\jacob\Desktop\DSA2Program>
```

H) Screenshots of Code Execution:

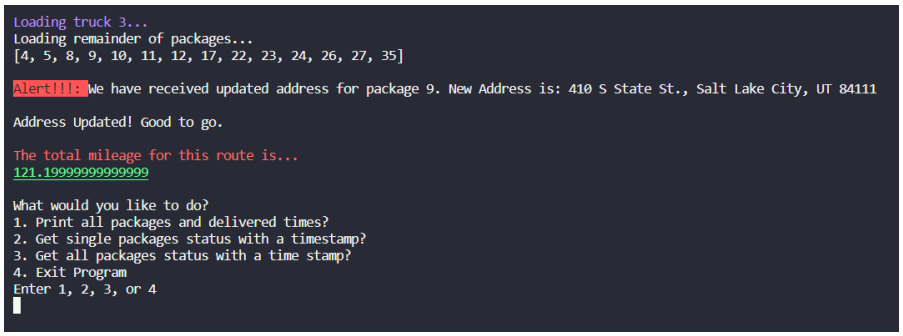
Screenshot captures complete execution of code and includes total

Delivery mileage – delivery mileage in photo 2

Part 1 (shows file run location):



Part 2 (wanted to include menu):



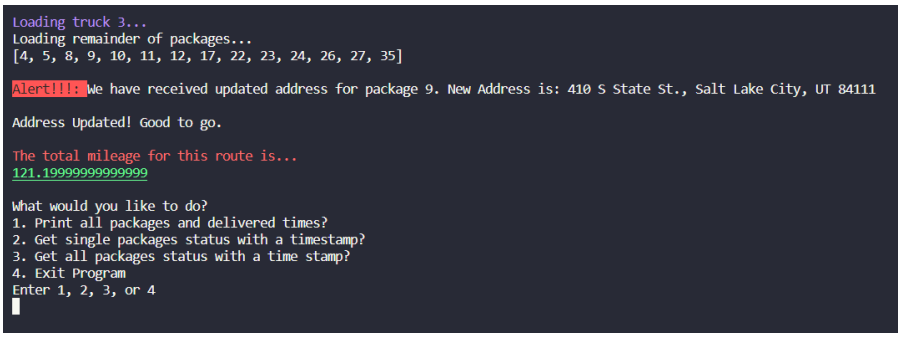
I1) Strengths of chosen algorithm

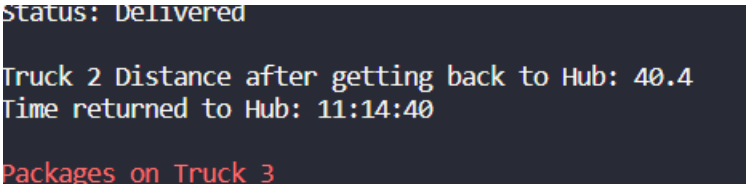
Nearest neighbor is perfect for this project and I will give two reasons why:

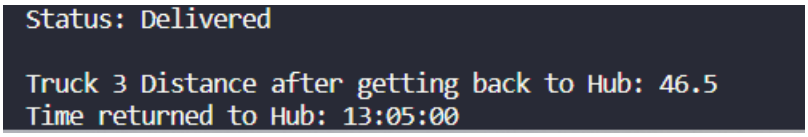
1. Nearest neighbor applies a “highest similarity” approach with zero training. It answers the same question at each iteration of its call- “which of these in the list is most similar to the current?”. For the application of this program, “most similar” translates to closest. The non-need for guessing provides an extremely high level of accuracy, since its basing its decision on *all* available data at every cycle.

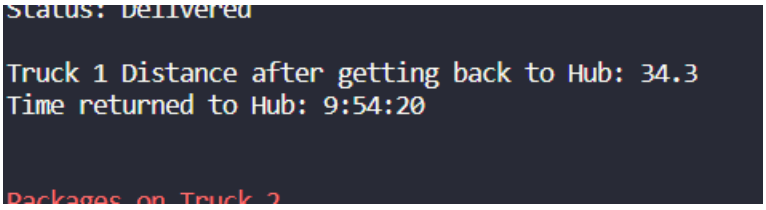
2. It is a low cost algorithm. Its not having to make any guesses or approximations about future data. It doesn't look long term (which means it may not be the *most* optimal solution), but instead it is one of the best for short term decisions, especially when the data set is relatively small (only a max of 16 per truck).

I2) Verification of chosen algorithm









As shown in the above screenshots, our trucks combined mileage is 121.99 repeated. This is not the most optimal solution but it does accomplish the task of clocking in under 140 miles. The screenshots in G1, G2, and G3 also show the on time deliveries of all packages.

I3) Other possible algorithms

find two other algorithms that would satisfy the requirements for this traveling salesman problem:

Brute Force: This algorithm would be the easiest to implement but as the dataset grows then this algorithm time and space grows disproportionately

Genetic Algorithm: This algorithm follows principles of genetics, slowly creating a best fit

algorithm through multiple processes.

I3A) Algorithm Differences

Compare and contrast the two previous mentioned algorithms, and how they compare to the selected nearest neighbor algorithm:

Brute force Algorithm: This algorithm is the least optimal but simplest to implement. It involved finding all possible solutions, by iterating through each distance until it reached the end, for every single combination. Doesn't take much to see why this is extremely inefficient for space and time complexity. When compared to nearest neighbor, it may find the most optimal solution, but it will do so in a very inefficient way. As compared to the genetic algorithm, the same can be said.

Genetic Algorithm: The standard implementation of this algorithm involves 5 steps, 1. Creating initial population 2. Calculating fitness 3. Selecting best genes 4. Crossing Over 5. Mutating to introduce variations. As compared to nearest neighbor, it works much slower. However, over time, it will trend towards an optimal solution with each iteration. The two could be combined however. In the long run, genetic algorithms should provide a much more optimal solution since it will trend towards it over time.

J) Different approach:

Identify at least 1 aspect that would be done differently if the project were attempted again, and include the details of the modifications that would be made.

If I were to do this project over again, the biggest change I would make is A. GUI and B. Include options to change the parameters within the UI. I like using either Tkinter or Pyforms so I would've liked to use one of those packages. Specifically, for the parameter changes, I would've added a plus or minus button for number of trucks. These buttons would be easy to implement since it would only impact the number of truck objects. For the other parameters, I would've included a drop down box for packages with the options "Must be loaded with", "Must be loaded on" and include another drop down box for a package ID or a truck number. For special packages there an option to drag packages into the sections for each truck.

K1) Verification of Data Structure

Verify the data structure meets all requirements. Includes total mileage, and that all Packages are delivered on time. The hash table with lookup is present and the reporting is accurate and efficient.

As shown in screenshots, all is accurate and as required. Check sections F for lookup, G1-

G3 for accurate reporting, and G for total mileage of all trucks

K1A) Efficiency

Explain how the lookup function is affected by increased number of packages

If there was a larger sized key, the lookup function might be affected at the average case.

Since we are using only package ID and the average lookup time for our hashmap

Is O(1), aka constant, we shouldn't run into any time differences for more

packages. If we run into our worst case complexity, O(n), then we will face a

a higher time for lookup. But that is hopefully the odd ball and wont happen very often.

K1B) Overhead

Explain the implications of adding more packages to the space complexity of the

Hashmap

If we were to add more packages, since our hashmap size grows linearly with the

number of packages, space would increase at O(n). Ideally, we would want to have initialized the hashmap with enough space to accommodate. The space complexity of the lookup, delete, and insert functions would remain O(1) since they don't store any value. They only act on the existing hashmap itself.

K1C) Implications

Accurately address how changes to number of trucks or cities would affect the lookup

time and space usage.

Time complexity:

Chaining hashtables maintain the same time complexity for increases in data size.

On average, we get O(1). Worst case, O(n). Adding a higher

number of Trucks or cities would maintain that complexity for

insertion, remove, and lookup .

Space Complexity:

Chaining hashtables allocate memory outside of hashmap for linked nodes

However, since we are increasing the number of cities and trucks and not packages, the space complexity would remain the same, as we are storing the same *number of* elements within the linked lists. A standard chained hashtable has a complexity of O(n), but we have to add the number of elements, m. Therefore a chained hashtable would be O(n+m). The number of elements would not increase if we were to add more trucks or cities, so there would be no change. Each package would still have 1 designated truck and 1 city.

K2) Other Data Structures

Identify two other data structures that would meet the same

requirements in the scenario

1. Binary Search tree: A good alternative since it can also store key value pairs. It could do this via storing the inorder traversal in a secondary array. It would be similar time complexity for its worst case and worse time complexity for average case
2. Stack: This is another good alternative since we could have represented the address as members of the stack. We then could have taken the distance from the popped member to the next member of the stack in line. This would have been a slower lookup time but a better space complexity.

K2A) Data Structure Differences

Contrast and compare the above mentioned data structures with the chosen

hashmap

1. BST: As mentioned, a BST can also store key value pairs. The only difference is that we would be using nodes and edges vs a linked list type structure. This would lead to an average time of O(logN) for search, insert, and delete. This would be fine except for when we begin to scale. As we scale, O(logN) grows in complexity at a much faster rate than O(1), our average case or even O(n) our worst case.
2. Stack: A major difference between a stack and the structure we chose is that a stack has a much smaller memory footprint. While this is great, the lookup function would take O(n) on average since we need to pop each element in the stack. This would be a great choice for smaller datasets.

L) Sources

Submission includes intext citations for sources that are properly coded

No sources used

M) Professional Communication

Content reflects attention to detail, is organized, and focuses on main ideas as prescribed by task or chosen by Candidate

Thanks for evaluation! -Jacob