WGU C950

Data Structures and Algorithms II

Robert Bergman

StudentID \*\*\*\*\*\*\*\*\*\*\*\*\*

Core Algorithm Overview

A. The core algorithm used in this simulation is a brute force algorithm that has no scalability. It works in this instance simply due to the small size of the data sets being evaluated. Anything over 16 nodes would take impossibly long to calculate.

The algorithm uses a recursive function to follow every possible path that can be made between the locations being evaluated one time each beginning and ending at the same point. This graph problem is known as a Hamilton Cycle which is the path as described through an undirected graph of locations and the distances between the locations as given in the distance matrix provided as part of the project.

A better approach would have certainly been the minimum spanning tree using either Prim-Jarnik or Kruskals algorithms. Prim-Jarnik generates a minimum spanning tree in O(N^3) Time with the same space, Kruskal in O(N log M). Another approach would be a genetic algorithm to find the best path approximation.

For the brute force method used the algorithm as follows:

“B.1.” G(i,S) = min(w(i, j) + g(j, {S-j}) (Lectures, 2019)

Algorithm BruteForce Dynamic Programming Hamiltonian Cycle (G):

Input: An undirected, weighted, connected graph G with n vertices and m edges

Output: The shortest path

g = dictionary with all (i, S) keys holding the minimum value for the iteration

S = Set of locations not including start location

i = start location

p = array holding all

func g(I, S):

For g(I, S) do

Get min of path w(I, j) + g(j, S-j)

Store path

Path = start +

For segment of path do

Add minimum distance edge to path

Add start to end

The recursive call to G(i, S) until base case where S is only one location or vertex. Record the minimum starting at the base case and work back through the stack until the minimum path is returned.

Iterate through the minimums to get the final path to return

For the location in the array of minimum paths, get the distance of the minimum path and add it to the final path array.

“B.2” The programming environment used to create the Python application was Jetbrains Pycharm version 2021.3.3 and python 3.9

“B.3.” Each function has the time and space complexity listed in the source code

“B.4.” The ability of the solution used for solving the package delivery problem in this project to scale is very low. While the problem at hand is solved is a difficult one, using solutions that incorporate minimum spanning tree and depth-first searches would provide a more scalable solution. With packages over 16 the brute force functional programming method starts to take noticeable amounts of time and space, 40 packages in one path took all the compute resources of my machine for longer than I was willing to let it run.

“B.5.” The software has well-defined classes and interfaces between objects making it efficient and easy to maintain. The comments in the code help to explain the logic being used to complete the project. Code segments are clarified where they might be confusing in the future.

“B.6.” The self-adjusting data structure or hash table is well suited for storing data that needs to be stored and retrieved with minimal time. Some of the challenges discovered while designing the hash table used in this project were the difference between hashing a number as an integer vs a string. The integer hash in python is just the integer being hashed whereas a string representation of a number is based on the memory location of the string. This can create a different hash for the same number during comparison if the number is a string resulting in different storage locations in the array.

C.  Write an original program to deliver all the packages, meeting all requirements, using the attached supporting documents “Salt Lake City Downtown Map,” “WGUPS Distance Table,” and the “WGUPS Package File.”

1.  Create an identifying comment within the first line of a file named “main.py” that includes your first name, last name, and student ID.

2.  Include comments in your code to explain the process and the flow of the program.

“D.1.” The data points stored in the hash table for this project were the packages. Each package had an id that was used as the key to retrieve data and store data in the hash table. Each data point was stored along with all the information needed to determine the location the package needed to be delivered to, the truck that was used to deliver the package, and the time the package was delivered. Each package has a location associated with it and an index that gets used to determine the distance from the distance matrix. The location information with the distances from the distance matrix gets used to establish the vertex and edges used in the graph needed by the algorithm to determine the shortest path.

E.  HashTable class

class HashTable:

table\_size = 31 # should not be a power of 2 or a power of 10 and should be prime

def \_\_init\_\_(self):

self.buckets = [None for i in range(self.table\_size)]

self.size = 0

def insert(self, key, item):

"""

O(n) Time-for bucket traversal O(n) space

insert item into hashtable and store the item with its key

:param key:

:param item:

:return:

"""

# identify the bucket for insertion

index = self.\_\_hash\_index\_\_(key)

# if the bucket is empty append the key/value pair

if self.buckets[index] is None:

self.buckets[index] = []

for i, kv\_pair in enumerate(self.buckets[index]):

k, v = kv\_pair

if k == key:

# if the key is in the bucket update the key/value pair with the new key/value pair

self.buckets[index][i] = (key, item)

return

self.buckets[index].append((key, item))

self.size += 1

def delete(self, key):

""" NOT IMPLEMENTED"""

pass

def search(self, key):

"""

O(n) Time O(n) Space

:param key:

:return:

"""

hash\_key = self.\_\_hash\_index\_\_(key)

bucket = self.buckets[hash\_key]

if bucket is not None:

for item in bucket:

if item[0] == key:

return item

return None

#raise KeyError

def \_\_hash\_index\_\_(self, prekey):

"""

O(1) Time O(1) Space

returns a hashed index from the passed prekey

:param prekey:

:return:

"""

# keys are finite and discrete. covert key to discrete integer

key = hash(prekey) % self.table\_size

return key #

def \_\_str\_\_(self):

hashtablestring = ""

print\_packages = []

for bucket in self.buckets:

if bucket is not None:

for item in bucket:

k, v = item

print\_packages.append(v)

print\_packages = sorted(print\_packages, key=lambda x: x.id)

for package in print\_packages:

hashtablestring += str(package) + "\n"

return hashtablestring

def \_\_repr\_\_(self):

return self.\_\_str\_\_()

@property

def keys(self):

"""

O(n) Time O(n) size

get all the keys and return them

:return:

"""

\_keys = []

for i, bucket in enumerate(self.buckets):

if self.buckets[i] is not None:

for item in self.buckets[i]:

\_keys.append(item[0])

return \_keys

# dunder functions

def \_\_getitem\_\_(self, key):

return self.search(key)[1]

def \_\_setitem\_\_(self, key, value):

self.insert(key, value)

def \_\_len\_\_(self):

"""

O(1) Time

:return:

"""

return self.size

F.  Look up function

def search(option, search\_term, env):

"""

O(n) Time O(n) Space

Get the packages that match the search term and return the keys

:param option:

:param search\_term:

:param env:

:return:

"""

option = int(option)

"""

1. Package ID

2. delivery address

3. deliver deadline

4. delivery city

5. delivery zip code

6. package weight

7. delivery status

"""

result\_keys = []

# O(n) Time O(n) Space

for key in env.packages.keys:

if option == 1:

if str(env.packages[key].id).lower() == str(search\_term).lower():

result\_keys.append(env.packages[key].id)

if option == 2:

if str(env.packages[key].delivery\_address).lower() == str(search\_term).lower():

result\_keys.append(env.packages[key].id)

if option == 3:

if str(env.packages[key].deadline).lower() == str(search\_term).lower():

result\_keys.append(env.packages[key].id)

if option == 4:

if str(env.packages[key].dest\_city).lower() == str(search\_term).lower():

result\_keys.append(env.packages[key].id)

if option == 5:

if str(env.packages[key].dest\_zipcode).lower() == str(search\_term).lower():

result\_keys.append(env.packages[key].id)

if option == 6:

if str(env.packages[key].weight).lower() == str(search\_term).lower():

result\_keys.append(env.packages[key].id)

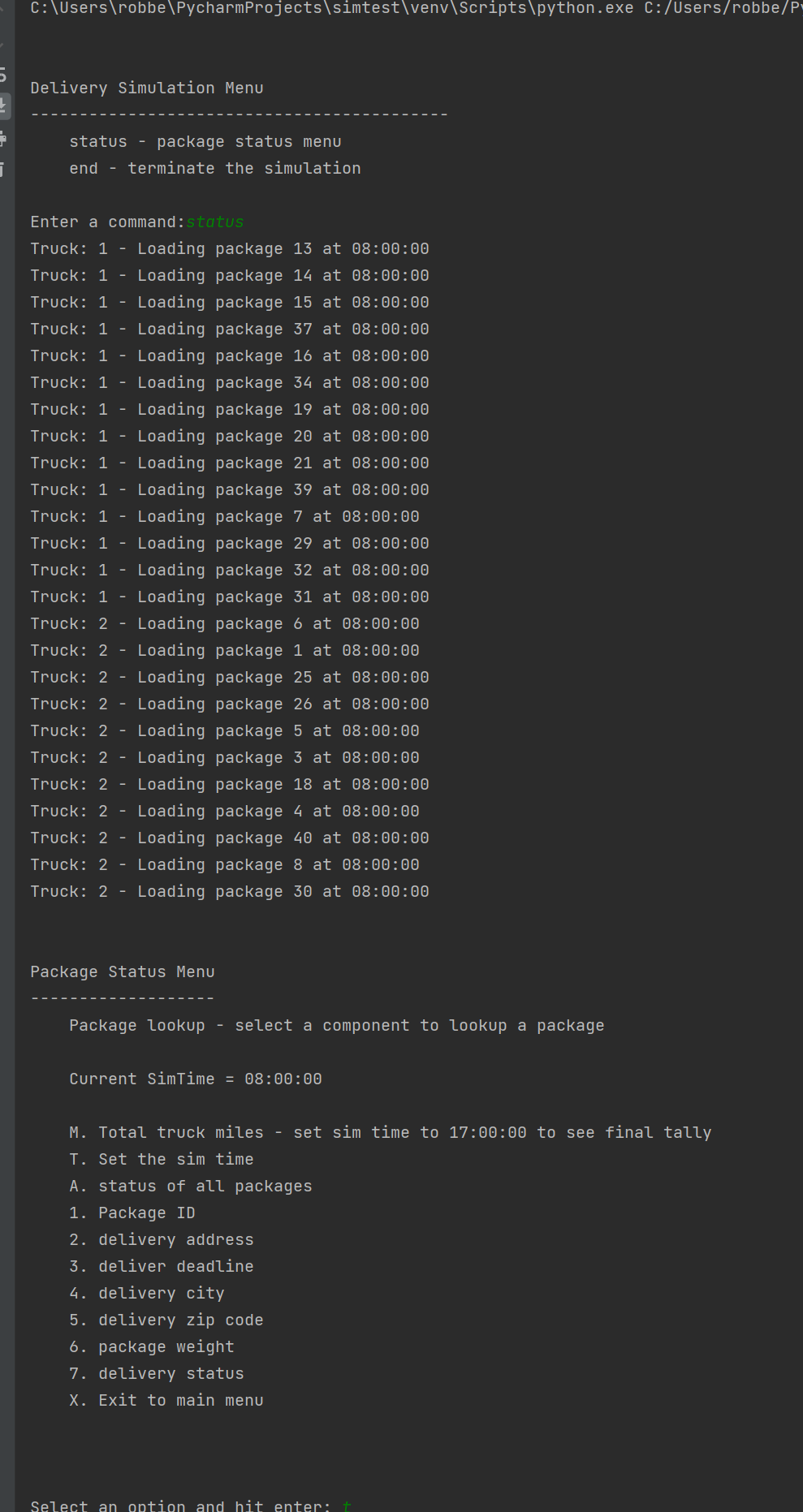
if option == 7:

if str(env.packages[key].status.name).lower() == str(search\_term).lower():

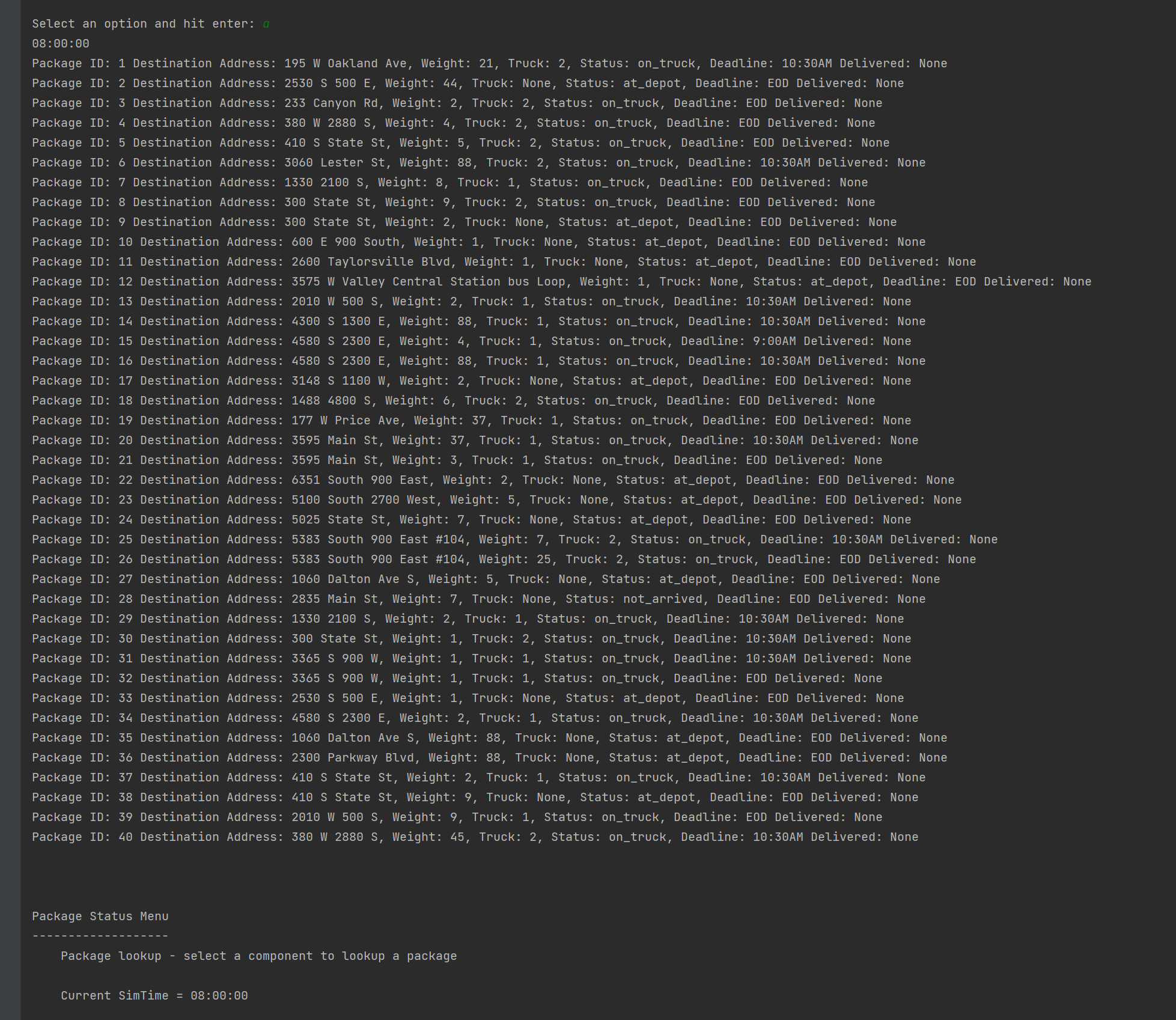
result\_keys.append(env.packages[key].id)

return result\_keys

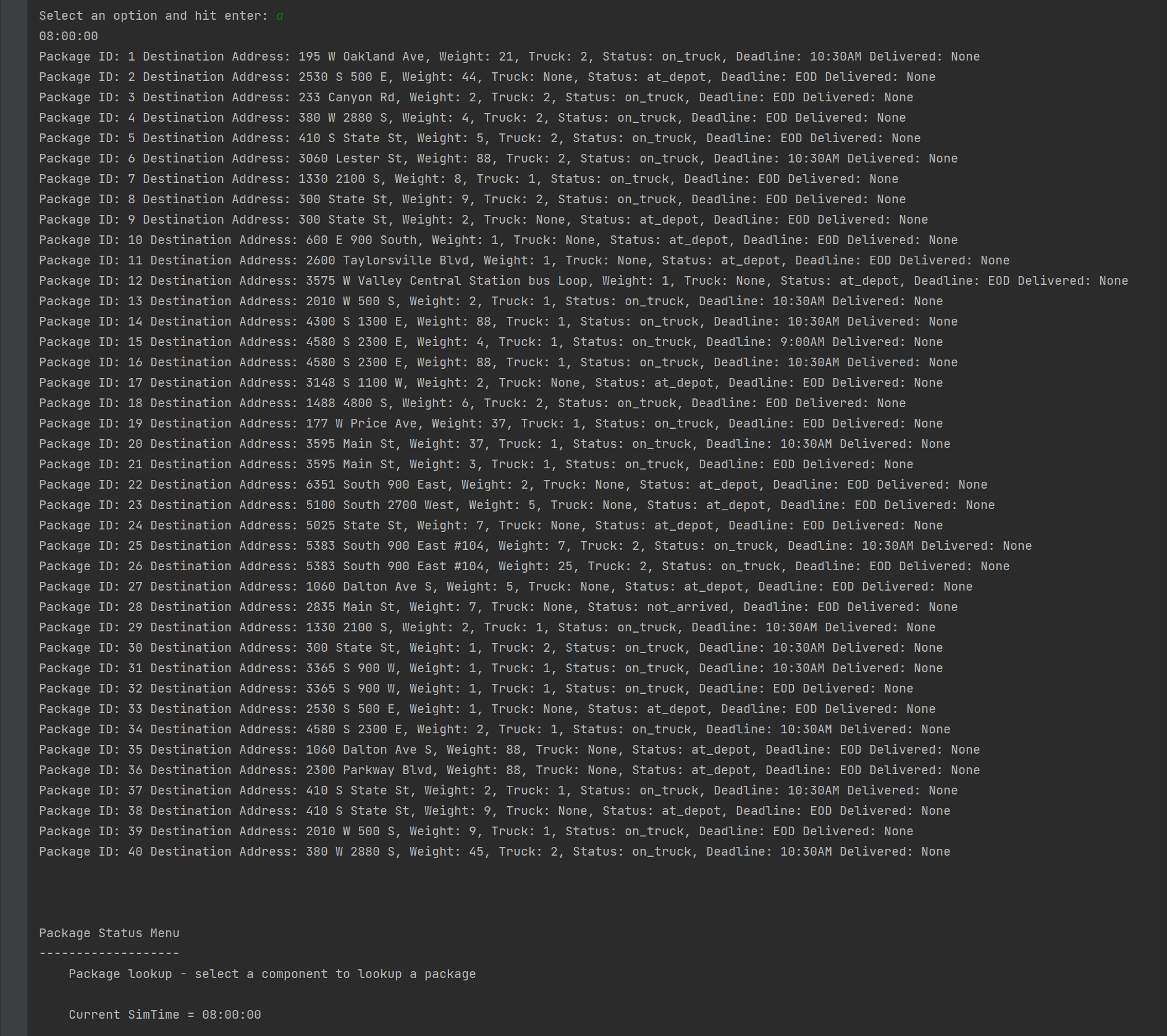
G.  Provide an interface for the user to view the status and info (as listed in part F) of any package at any time, and the total mileage traveled by all trucks. (The delivery status should report the package as at the hub, en route, or delivered. Delivery status must include the time.)



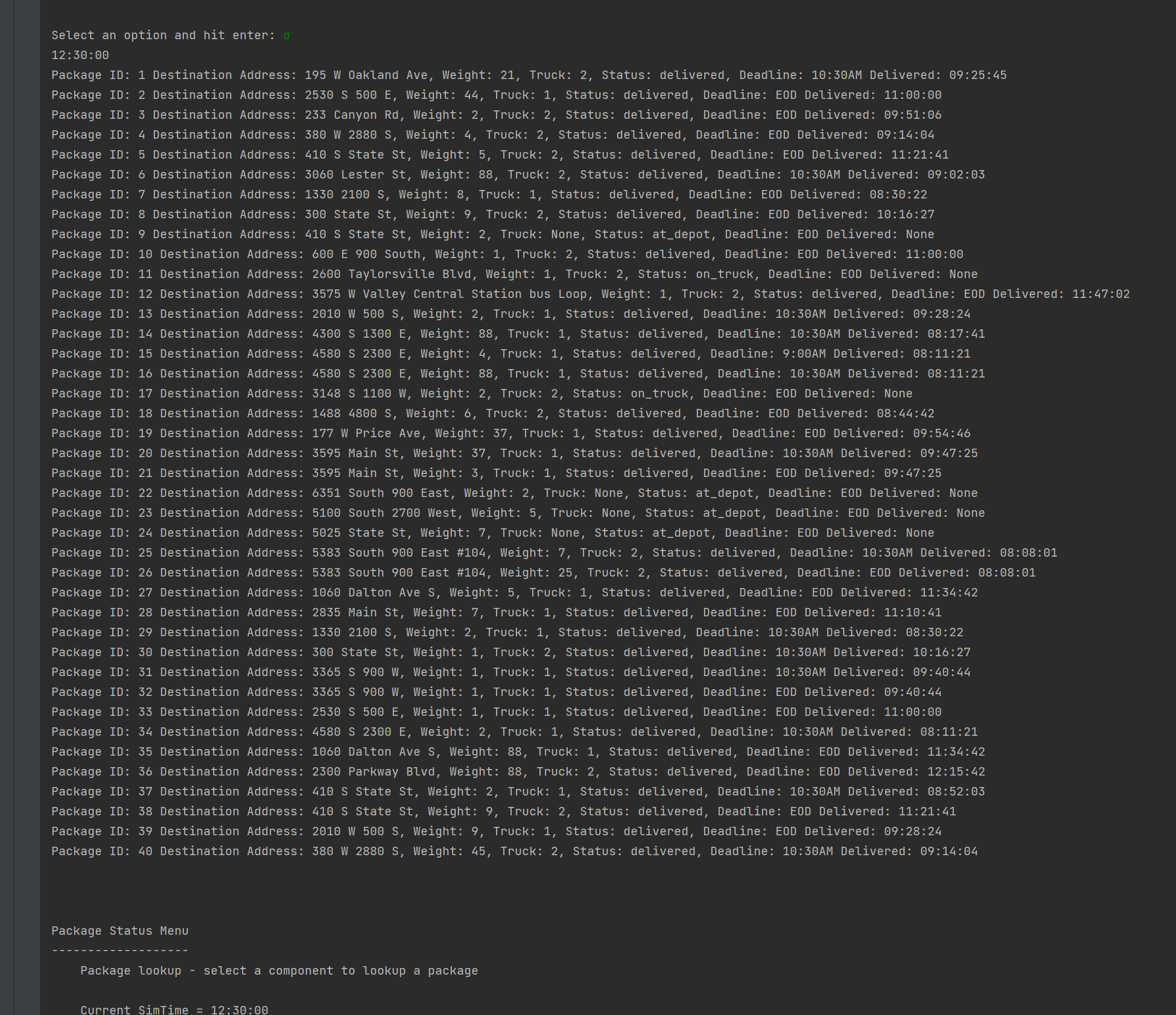
1.  Provide screenshots to show the status of all packages at a time between 8:35 a.m. and 9:25 a.m.



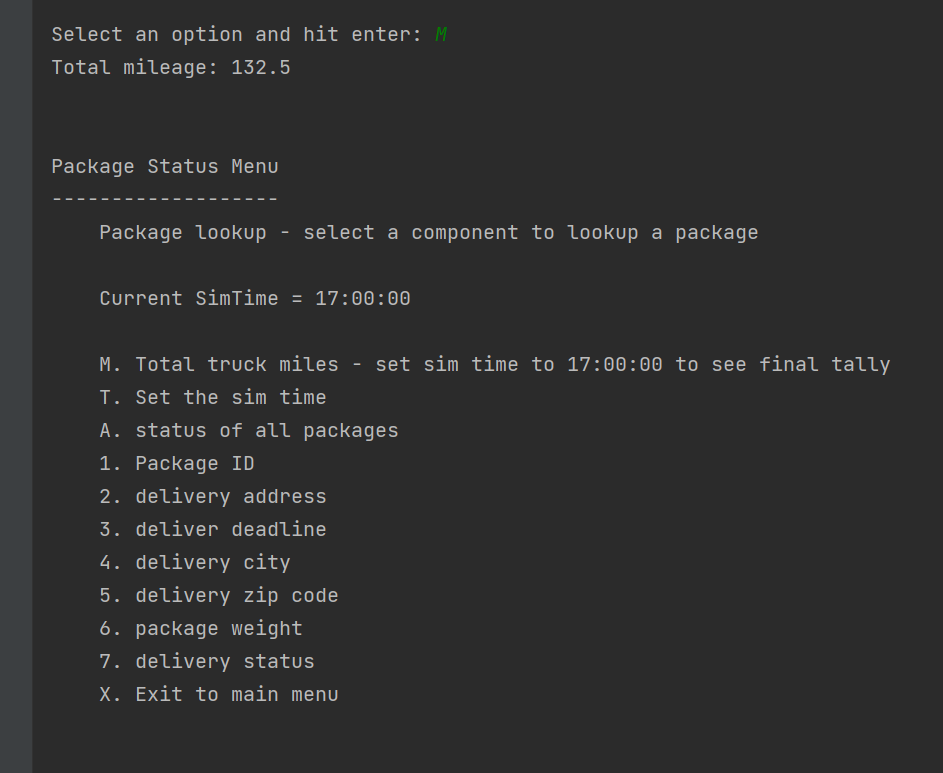
2.  Provide screenshots to show the status of all packages at a time between 9:35 a.m. and 10:25 a.m.



3.  Provide screenshots to show the status of all packages at a time between 12:03 p.m. and 1:12 p.m.



H.  Provide a screenshot or screenshots showing successful completion of the code, free from runtime errors or warnings, that includes the total mileage traveled by all trucks.



“I.1.” One of the strengths of the algorithm used for this project is that it does find the shortest path between all path locations. Since it evaluates every possible path and selects the shortest one out of all available paths, the path selected is the real shortest path as opposed to a possible shortest path. Another strength is the simple nature of the algorithm. It runs down each path and picks the shortest one through a recursive function that hits the base when only one location is left in the path to be evaluated. It then takes the minimum distance path and puts it in an array to be recalled upon completion.

2.  Verify that the algorithm used in the solution meets all requirements in the scenario.

“I.2.” Two trucks were used to deliver all the packages and all the delivery times were met at the end of the day. With a total mileage traveled at 132.5 miles

Truck 1 path 0 = 26.3

Truck 1 path 1 = 16.4

Truck 2 path 0 = 29.400000000000002

Truck 2 path 1 = 31.300000000000004

Truck 2 path 2 = 29.1

Total mileage for all trucks and paths: 132.5

3.  Identify **two** other named algorithms, different from the algorithm implemented in the solution, that would meet the requirements in the scenario.

a.  Describe how each algorithm identified in part I3 is different from the algorithm used in the solution.

“I.3” A more scalable approach could use a minimum spanning tree algorithm such as Prim-Jarnik or Kruskal's. Using the minimum spanning tree and Depth First search in combination would yield a path with repeating visits to some sites. The repeated sites are simply removed and the path returned is a close approximation of the shortest path. Another approach would be a genetic algorithm to find the best path approximation. In this case, a random series of paths are evaluated and evolved using various techniques where best paths are comingled to rapidly generate solutions using much less space on the system in terms of memory utilization. Best paths are stored and replaced as better paths are found.

“I.3.a” As described above, the genetic algorithm makes much better use of the space on the system by only storing the best path solutions where the brute force dynamic programming method used in the project has to store every possible path before releasing the memory as the result is formed.

The minimum spanning tree solution with depth-first search provides a very close approximation of the best path much quicker and uses less space. The graph used for the minimum spanning tree is only as big as the vertex and edges O(n^3) The algorithm in the project used O(2^n) space and quickly becomes unusable as the number for vertexes increases.

“J.” A lot of research went into how to create a simulation for this project. In the end, a simple look counting through the seconds gave me the resolution needed to properly evaluate locations based on time. I think a better solution would be to evaluate locations based on their original metric which was distance. Converting to time cause a change in the solution where instead of looping through minutes, seconds were needed to not skip some of the shorter edges such as .4 miles. With a minute you could travel .3 miles and at the right moment, you might completely miss a stop leaving packages on the truck. Another change would be to utilize a graphical representation of the data to help visualize the different algorithms and how the path selection process is affected depending on which was used. Finally, the hash table would have better scalability as a linked list rather than a simple array for the bucket chaining.

“K.1.a.” The time needed to lookup packages in the Hashtable is O(1+n/m) where n is the number of items being stored and m is the number of buckets to store them in.

b.  Explain how the data structure space usage is affected by changes in the number of packages to be delivered.

“K.1.b.” The data structure uses space to store the keys and the data which includes the key again. O(2n+m) space is used as the number of items increases where n is the key and m is the data.

c.  Describe how changes to the number of trucks or the number of cities would affect the look-up time and the space usage of the data structure.

“K.1.c.” The number of trucks and cities does not affect the look-up time. The location and truck references are stored for each package and the lookup times and storage space would be the same regardless of how many different trucks and locations there were.

“K.2” A stack or a queue could have been used to store the packages in the scenario.

“K.2.a.” A stack could have been used to store the packages. Rather than store the data in a hash where it could be easily referenced, the system could have used the locations as the primary data point and stored the data for each location with a stack where the packages would be pushed onto the stack as they were loaded onto the truck for the locations and then popped out of the stack upon arrival at the location. Another option would be to use a queue where each package would just be queued for the location at the depot by putting the reference to the data in the queue and the removed upon arrival at the location in the order they were put in.

# Bibliography

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2013). *Data Structures and Algorithms in Python.* John Wiley & Sons.

Lectures, J. (2019, Mar 13). *Traveling Salesman Problem using Dynamic Programming | DAA*. Retrieved from Youtube: https://www.youtube.com/watch?v=hh-uFQ-MGfw&t=987s&ab\_channel=Jenny%27slecturesCS%2FITNET%26JRF

Sedgewick, R., & Wayne, K. (2011). *Algorithms Fourth Edition.* Peason Education, Inc.

M.  Demonstrate professional communication in the content and presentation of your submission.