10/07/22

**A:** The algorithm used to deliver the packages is the Nearest Neighbor algorithm.

**B1:** Packages are loaded and delivered in separate functions.

Loading:

sudo

//Go through each bucket in the hash table and separate the packages with special

//notes from those without.

for i in range (number of buckets in hash table)

for j in range (number of items in bucket i)

if bucket has special notes add it to specialPkgs list

else add it to normalPkgs list

//Bubble sort used to sort packages without special notes in ascending order by zipcode.

//This is to ensure when they are loaded that packages with addresses geographically

//close together are loaded on the same truck (after special packages are dealt with).

for i in range (number of normal packages)

for j in range (number of normal packages – 1)

currentPackage = packageTable.search(normalPkgs[j])

nextPackage = packageTable.search(normalPkgs[j + 1])

if currentPackage.zipcode > nextPackage.zipcode

swap currentPackgage and nextPackage

//Loop through the specialPkgs list first, since some special instructions might require

//the packages to be loaded on a specific truck, and there needs to be space available

//in that truck.

for i in range (number of special packages)

currentPackage = packageTable.search(specialPkgs[i])

//Here special cases can be listed for all the special notes required. This part

//of the code would have to be customized/expanded as the program was scaled

//up. An example special case is given here.

If currentPackage.specialNote == “Can only be on truck 2”

load package on truck2

//Load the normal packages now, they can be loaded sequentially as the trucks have

//space available, since they are already sorted by zip code. Small adjustments could be

//made later if a particular package would be much better off on a different truck.

for i in range (number of normal packages)

if truck1 is less than capacity

load normalPkgs[i] onto truck1

if package must be delivered by a certain time

add normalPkgs[i] to truck1.priorityPkgs

else if truck2 is less than capacity

load normalPkgs[i] onto truck2

if package must be delivered by a certain time

add normalPkgs[i] to truck2.priorityPkgs

else

load normalPkgs[i] onto truck3

if package must be delivered by a certain time

add normalPkgs[i] to truck3.priorityPkgs

Delivering:

sudo

//Loop through each truck’s deliveries until the user specified time has been reached

//using the Nearest Neighbor algorithm. Each truck will loop through the remaining

//packages on the truck and find the closest one to the current location of the truck.

//Then it will move to that location and deliver the package, removing it from the

//Truck and updating its status and delivery time.

//Truck3’s start time is truck1’s ending time since the driver will be switching over to

//truck3 after returning to the hub.

while currentTime < userTime and truck1 is not empty

//Deliver priority packages first – those that have a delivery deadline early

//in the day to make sure they’re delivered on time. The priorityPkgs list and the

//packages list are just lists of integer package ID numbers. The packages

//themselves are never duplicated and are always looked up in the hash table

//for reference or updating data.

while truck1.priorityPkgs is not empty

for i in range (truck1.priorityPkgs.length)

if distancesTable[currentAddress][previousAddress] <

previousDistance

nearestNeighbor = truck1.priorityPkgs[i]

distance = distancesTable[currentAddress]

[previousAddress]

currentTime = currentTime + distance / 18

truck1.priorityPkgs.remove(nearestNeighbor)

truck1.packages.remove(nearestNeighbor)

truck1.add\_mileage(distance)

//Deliver the normal packages if there is still enough time before the user

//requested time is reached.

for i in range (truck1.packages.length)

if distancesTable[currentAddress][previousAddress] < previousDistance

nearestNeighbor = truck1.packages[i]

distance = distancesTable[currentAddress][previoiusAddress]

currentTime = currentTime + distance / 18

truck1.packages.remove(nearestNeighbor)

truck1.add\_mileage(distance)

//Return to the hub with truck1 and add that mileage as well.

Truck1.add\_mileage(distancesTable[previousAddress][HUB\_ADDRESS]

//Repeat this process with truck3 without resetting the currentTime variable since

//it has the same driver as truck1.

…

//Repeat the process used for truck1 for truck2, thereby getting all trucks up to date for

//the user time specified.

…

//If the user requested to see the status of all packages and mileages

if pkgID == 0

print truck1 mileage

print truck1 packages with id, status, and delivery time (if applicable)

print truck2 mileage

print truck2 packages with id, status, and delivery time (if applicable)

print truck3 mileage

print truck3 packages with id, status, and delivery time (if applicable)

print total mileage for all trucks

//if the user entered a specific package id to check

else

print specific requested package with id, status, and delivery time (if applicable)

**B2:** PyCharm IDE Community Edition v2022.2.1 was used to write this program on a Windows 10 desktop computer. Python 3.10.7 was used.

**B3:** Space-Time Complexity: Each smaller segment of code has Big O notation in the comments for specific segment complexity.

Major Segments:

Address\_lookup function: O(n) a for loop was used

Load\_trucks function: O(n^2) a for loop within a for loop was used

Deliver\_packages function: O(n^3) a for loop within a while loop within a while loop

was used.

Opening each file: O(n) a for loop was used

Package hash table search: O(n) a for loop was used

Package hash table remove: O(n) a for loop was used

Entire Program: O(n + n^2 + n^3)

**B4:** Scalability. This program should easily be able to scale up to a large number of packages. The hash table is designed to accommodate many more packages than the 40 required here, and each truck’s capacity is managed through a CAPACITY variable that can easily be changed to update all the trucks capacities. Everything is designed in a way that accommodates whatever number of packages there are in the package file. The only portion that would take much manual editing to scale up would be the special instructions packages section. That would of course be customized to whatever special instructions the packages might have in a larger system. The algorithm would grow to accommodate additional packages by a factor of n^3 since the algorithm must loop through each priority package (packages with a deadline for delivery) to find the nearest neighbor each time a priority package is delivered, all within a loop for delivering the truck’s contents as a whole.

**B5:** Efficiency and Maintainability. The program is efficient because the package objects are never duplicated. They are created and sorted into the hash table (which enables quick lookup), then looked up within the program when they need to be referenced or edited. The “packages” within each Truck object are simply integer package IDs that can be used to quickly retrieve a package from the hash table if needed. It is easily maintained in that the capacity of all trucks can quickly be updated by changing the CAPACITY variable, and adding more packages to the package file is all that is required to scale up the program. The program reads whatever number of packages it is given from the package file and sorts and delivers them.

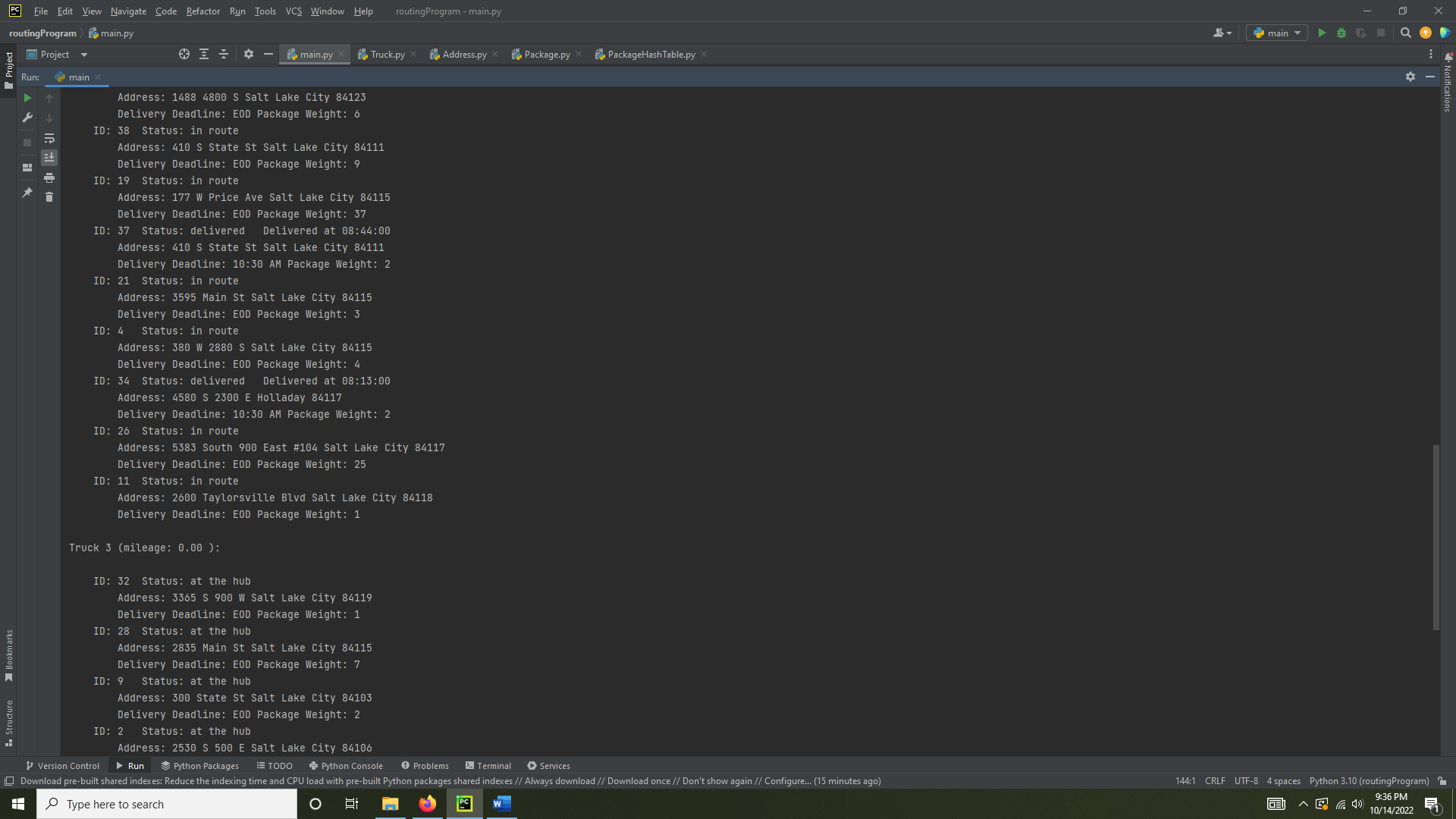
**B6:** The hash table is a good data structure to use because it makes the lookup of a package quicker since they are hashed by ID, and it provides scalability since the list of packages is broken down into many smaller lists. The disadvantage of the hash table is that it’s almost overkill on such a small scale as was used here, if this program were for a small delivery business it might be better to just use a dictionary or other data structure better suited for smaller data sets. The nearest neighbor algorithm was good for this program because it delivered the packages in the most efficient way based on zip code after considering which packages must be delivered first due to delivery deadlines. The disadvantage of it was that it did not take into account delivery deadlines on its own, and some “backtracking” of trucks was unavoidable to ensure that the packages with deadlines were delivered on time.

**D1:** Data Structure. The data structure used to store the package data for this program is a hash table. The hash table hashes the packages by (ID number) % (length of hash table). In this case the length of the hash table is 10, but it could be changed in the future to allow for more packages. The packages are sorted into ten different buckets, which can then be searched by ID to reduce overall search time.

**G1:** Screenshots showing the status of all packages at 9:00 AMA computer screen capture

Description automatically generated with medium confidenceA computer screen capture

Description automatically generated with medium confidence



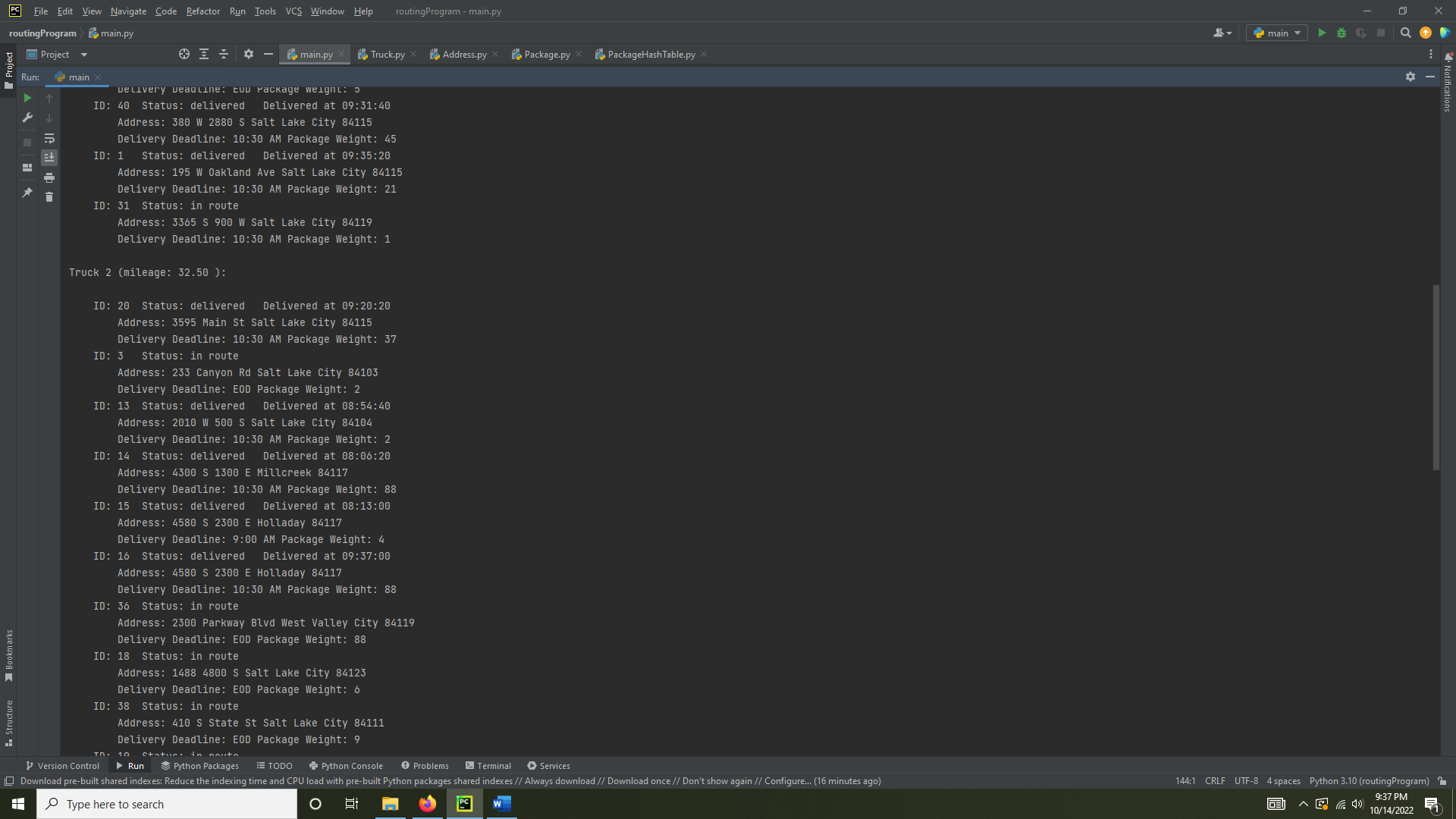
A screenshot of a computer

Description automatically generated with medium confidence

**G2:** Screenshots showing the status of all packages at 10:00 AM

A computer screen capture

Description automatically generated with medium confidence



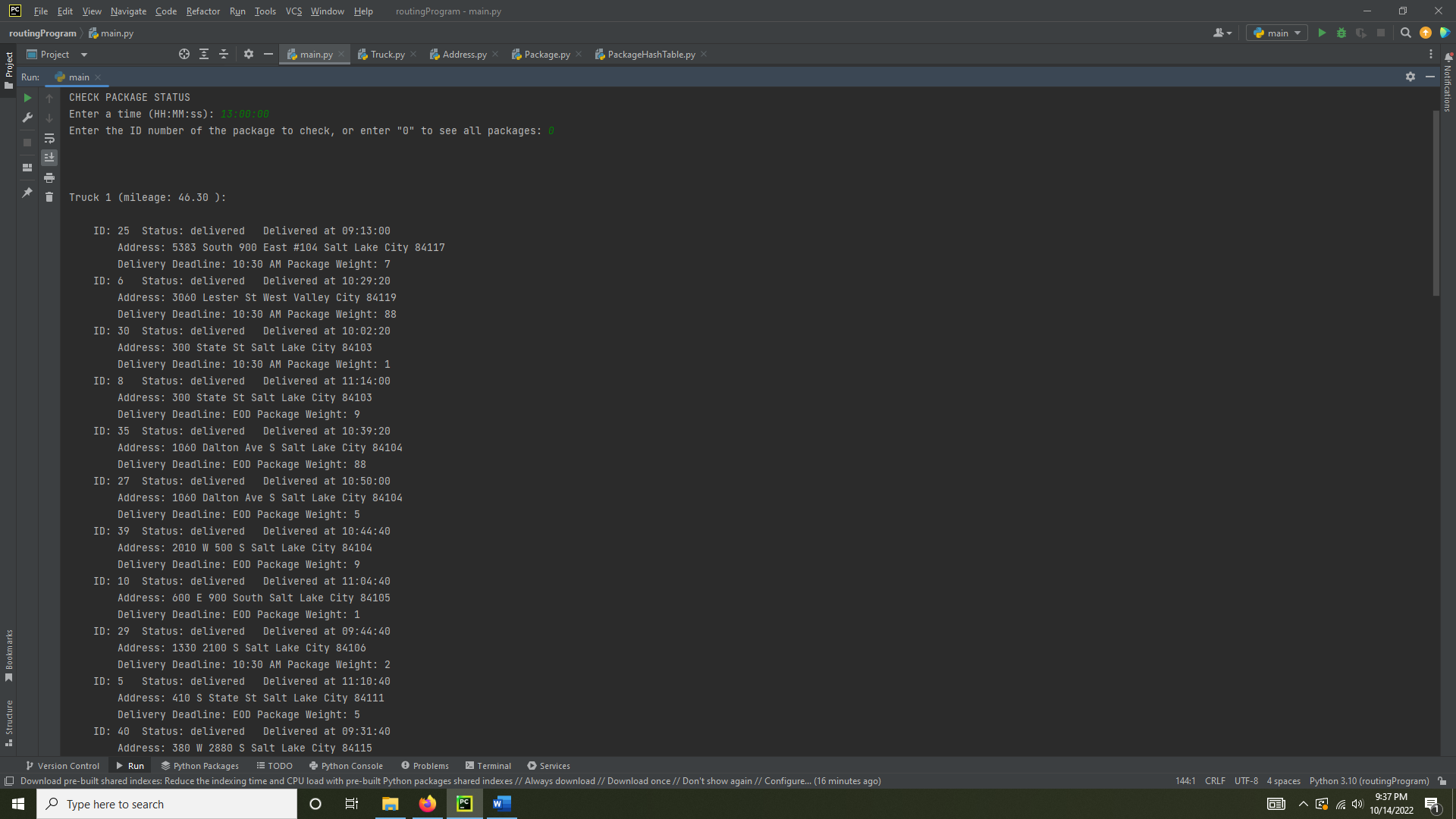
A computer screen capture

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

**G3:** Screenshots showing the status of all packages at 1:00 PM.



A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A computer screen capture

Description automatically generated with medium confidence

**H:** Screenshots showing total mileage traveled by all trucks and all package delivery times.

A computer screen capture

Description automatically generated with medium confidence

A computer screen capture

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A computer screen capture

Description automatically generated with medium confidence

**I1:** The nearest neighbor algorithm works well for any scenario with geographic locations because it’s easy to visualize with an actual map and makes logical sense to go to the nearest location next. It also is likely to find an efficient path to all locations if the locations to be visited are grouped logically by minimizing distance between each location.

**I2:** This program delivers all 40 packages in 139 miles, and all packages are delivered on time.

**I3:** Two other algorithm options would be a Greedy Algorithm or Dijkstra’s Algorithm.

**I3a:** The Greedy Algorithm would work for this scenario because it can take multiple factors into account when finding the next best location to visit as the truck moves (special notes, distances, and delivery deadlines). Dijkstra’s Algorithm would work for this scenario because it can determine the most efficient path to visit all addresses required by each truck before visiting them so you know you will get the shortest distance.

**J:** One thing I would change if I redid the project would be to use a dictionary instead of a list to store the addresses. It would make the address\_lookup function run more efficiently to look up an address by a key instead of iterating through the list.

**K1:** The hash table used has a lookup function by ID that works to retrieve the correct package. The total miles added to all trucks when the program runs is 139 miles. All packages are delivered before their respective delivery deadlines.

**K1a:** Increasing the number of packages would increase the time needed to lookup packages by a factor of O(n), since the packages are hashed into ten different buckets, it would be an increase of 10% the number of additional packages added on average.

**K1b:** Increasing the number of packages would increase the size of each bucket within the hash table. Each bucket on average would increase by 10% the number of additional packages added.

**K1c:** Additional cities and trucks being added would not affect the lookup time or space usage of the hash table.

**K2:** Two other data structures that could be used for this scenario are sets or a dictionary.

**K2a:** Using sets would be similar to using a hash table with buckets in that the packages could be grouped by some factor and accessed by a key, but you would lose the hashing aspect and the sets would all be separate entities. A dictionary would be similar to a hash table in that the packages can be accessed by a specified key, like an ID, but the search would be much less efficient than a hash table.

**L:** No outside sources were used outside of the course material.