Vladan Djurkovic

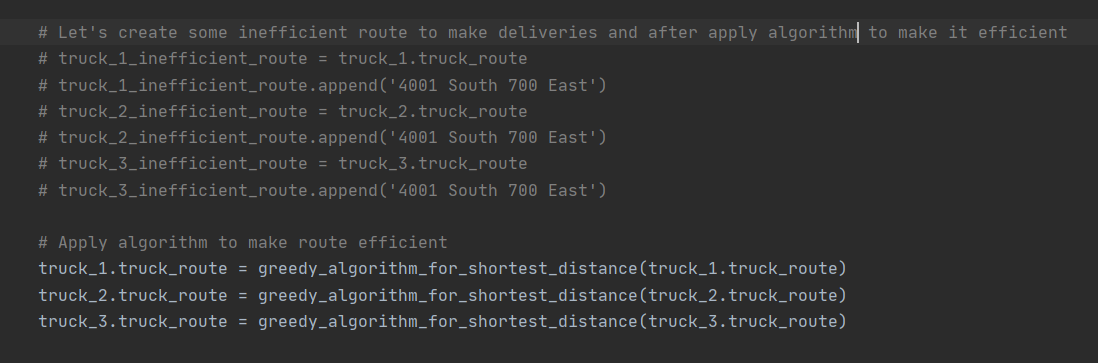
Student ID #000695823

WGU

Date: 08/14/2020

**Data Structures and Algorithms II – task C950**

For this task may choice of algorithm is a greedy algorithm because it is optimal and somewhat efficient choice to solve a traveling salesman problem by sorting list of delivery points based on the next closest location from the list. It will probably not give the best possible solution but efficient enough. I named it greedy\_algorithm\_for\_shortest\_distance which gets applied in deliveries.py file in order to make previously created inefficient route more efficient, help deliver packages and meet the requirements.



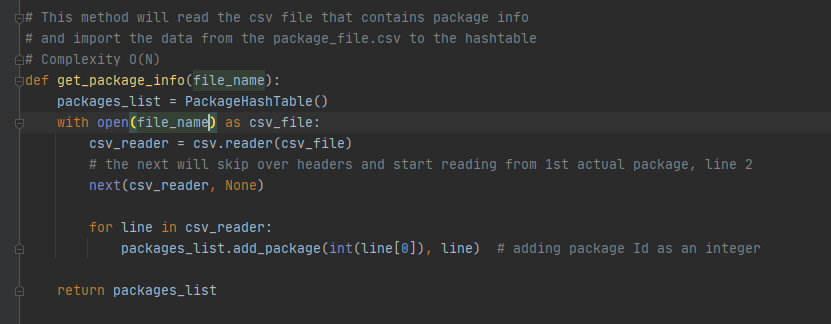
All the major section in the coding part of the project are commented on and explain logic of each method applied and here I will collaborate on how the greedy\_algorithm\_for\_shortest\_distance. Applying greedy algorithm will help to solve our problem with three delivery trucks by finding the best possible route for each truck. All three trucks will start their deliveries from the warehouse (Hub). Every package will be loaded with its attached address, and algorithm will sort given truck route in order to find the best possible route based on the next closest location from the current location. For instance, as I mentioned at the beginning the trucks will start at the HUB- 4001 South 700 East, so the starting point will be first added to a route and then algorithm will sort the route and looked for the package with attached address that is the closest to the starting point or the HUB and so on.

The algorithm function will take in a route that is passed to it. It has a starting point and it also has distance between the delivery points imported from the graph.py file. There is an initial route created as the packages are loaded onto truck. Then I created a list better\_route = [starting\_point] which will have a starting point and initially build a better route then the one already given. While loop is applied to make sure that it goes through the passed route until all the locations are removed from the list and it the algorithm returns possibly the shortest route. It starts with distance of 0 and with a starting point. For loop nested inside the while loop finds the distance between every location from the list of distances and eliminates visits to the same delivery location from the route by removing already visited location from the list. At the end, the result is returned greedy\_path list with the shortest path for truck to go and deliver packages.

Now this program runs on the local machine only, therefore there is no protocol that is used to exchange data. There is no server that is hosting the application and the local machine does not connect anywhere in order to interact with the application, exchange the data and after disconnecting from the server.

However, the current project is set up as follows: the project data files are stored on the same machine as the one that hosts the application. There are two csv files, distances\_file.csv and package\_file.csv that are stored under the project folder inside the data directory. The entire project is developed inside the PyCharm IDE using Python programing language. Name of the project is TSP\_data\_structures\_and\_algorithms\_II.

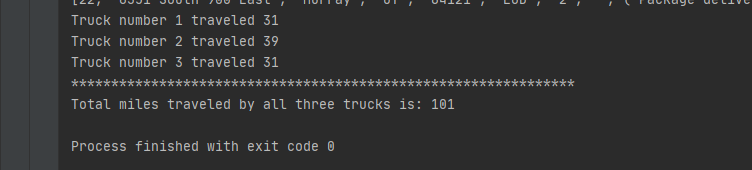
The time complexity is commented inside the project and every major block of code includes the comment where the time complexity is shown. Example shown on the picture:



In the future if there are routes with more destinations, algorithm complexity function will keep growing smoothly. The worst case that the algorithm could take is O(N^2). Algorithm can also be improved later by presorting a given list of distances if the given route grows largely bigger, which will help sorting algorithm to faster sort given route.

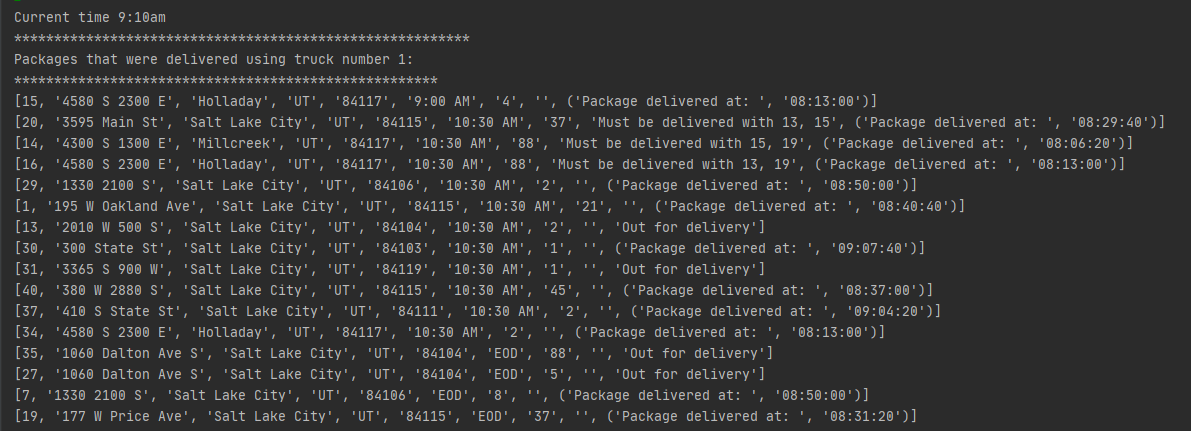
The entire program can be easily understood because its each major block contains comments on time complexity and what the given method does, which will definitely help if somebody wants to make any modifications on the program or attempt to enhance it.

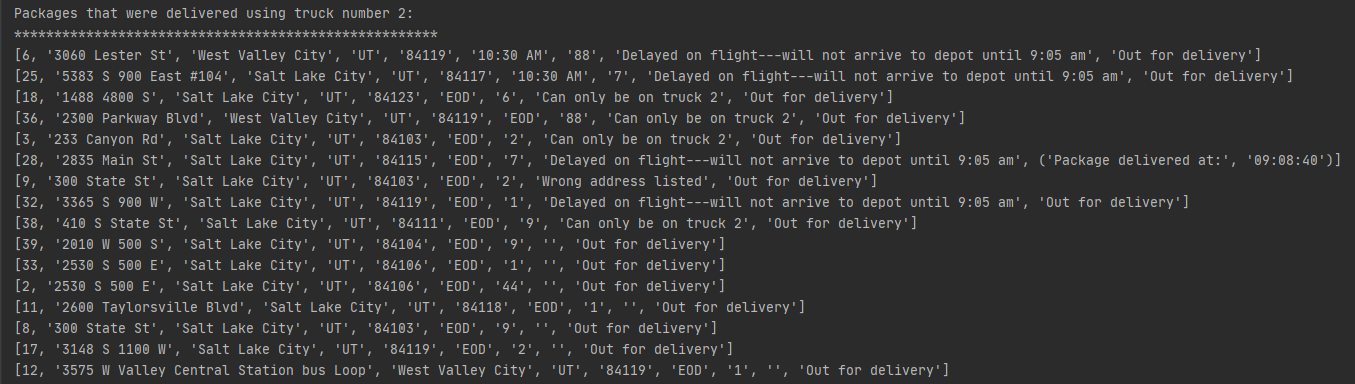
Provided solution meets the requirements because it delivers all the packages traveling less then 145mi by all three trucks together. Truck 1 traveled 31mi, truck 2 traveled 39mi and truck 3 completed deliveries traveling 31mi. Therefore, the total milage traveled by all three trucks is 101 miles.

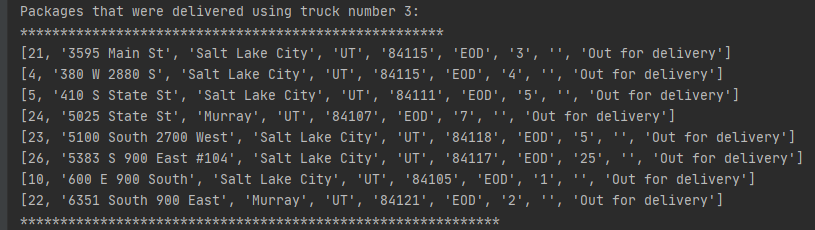


The program also clearly provides status for every package at any given time. For example, from the fallowing screenshots it can be seen status of every package at a given time.

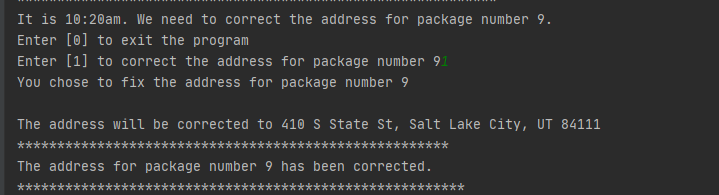
Status of every package between 8:35am and 9:25am. Given time is 9:10am.



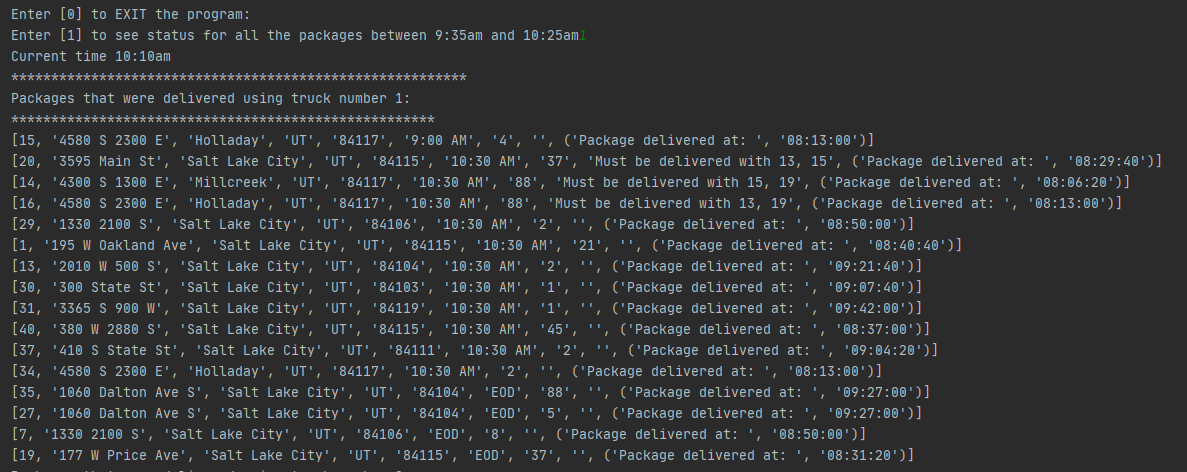


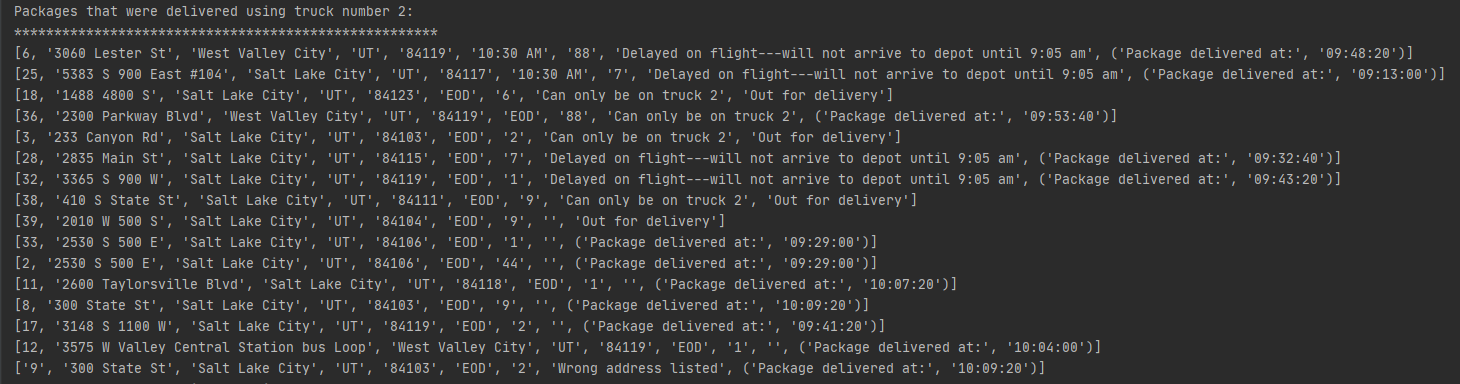


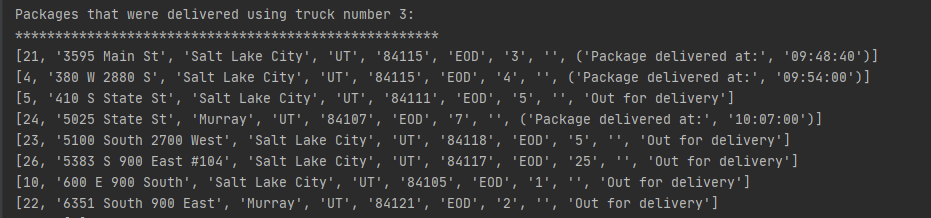
Then the program correctly fixes the address for package number 9



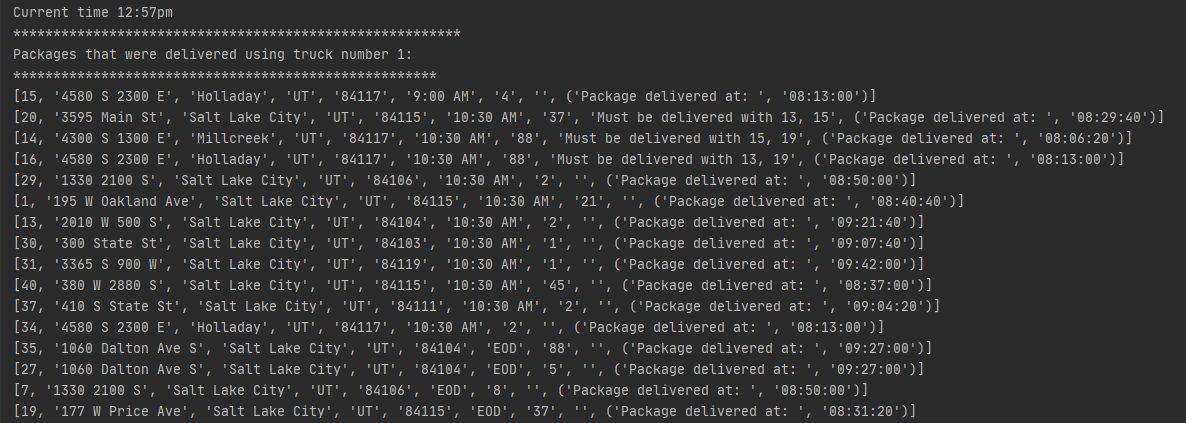
Then the program needs to show the status for all the packages between 9:35am and 10:20am. Given time is 10:10am.

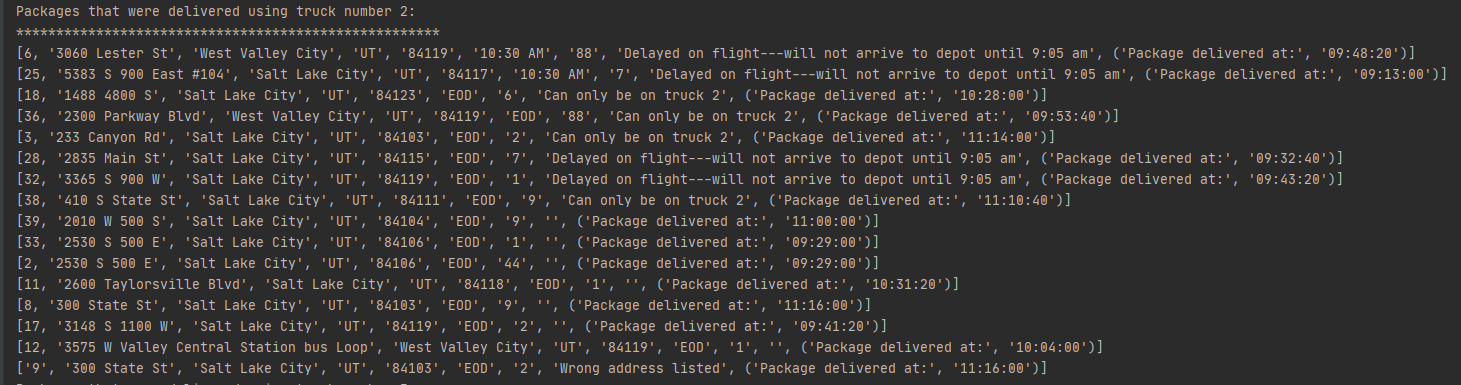


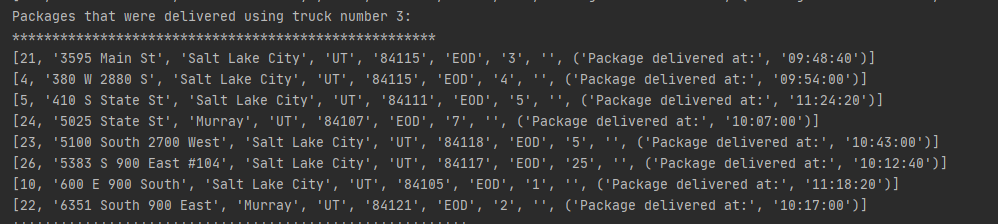




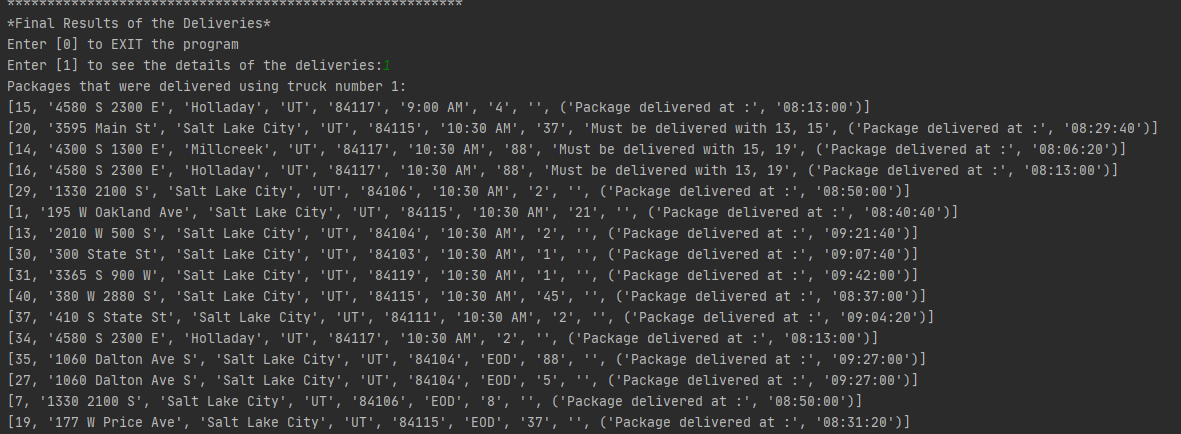
After that, the program checks the status for all the packages between 12:03pm and 1:12pm. Given time is 12:57pm

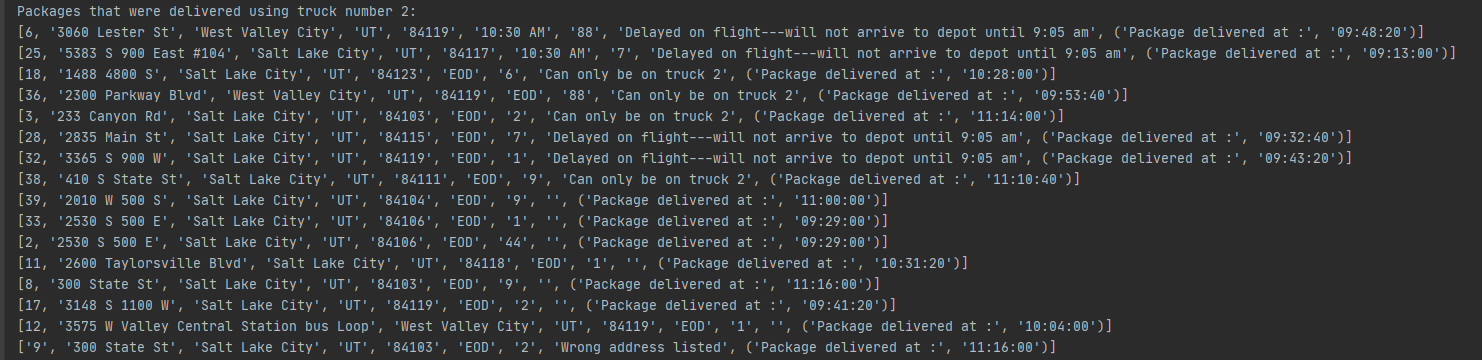


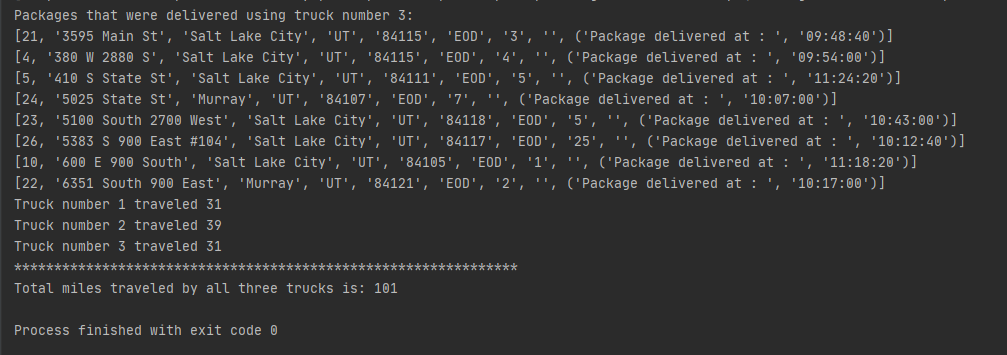




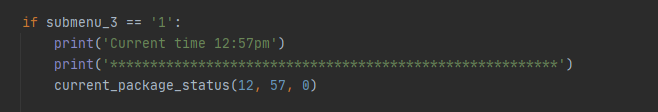
After that, the program displays result and status of all the deliveries that were done by all three trucks.



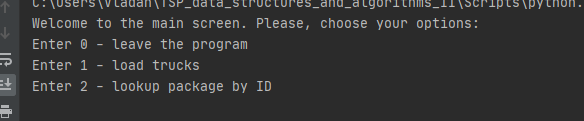


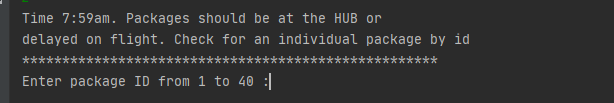


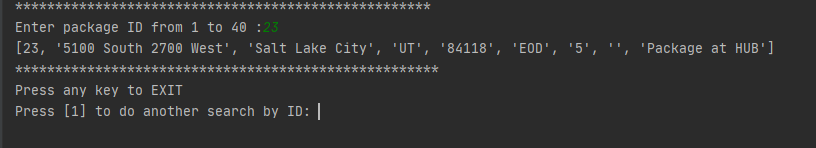
Passing different time to current\_package\_status (), inside the user\_interface.py file will display current package status for all deliveries and packages at any given time.



It also can show the current package status looked up by ID, by entering number 2 at the main screen of user interface:







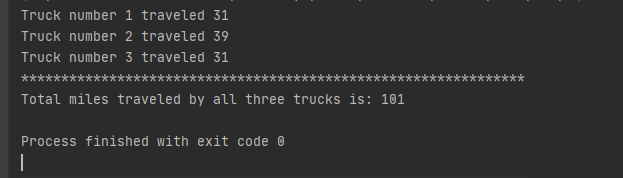
For this project I implemented a hash table called class PackageHashTable. The purpose of the hash table is to improve the speed of accessing packages that are stored inside the backets. Time complexity is O(N). Hash table has an initial capacity of 32 backets and each bucket can store a list with more than one package. For loop will loop through the hash table and add each append to initial list.

Method, def add\_package (self, key, package) inserts a new package into a hash table using a first index from package\_file.csv as a key, because the first column inside the csv file is the package Id. It will insert a package into a backet based on the package Id modulo length of the table. Based on mod taken it will place a package in the bucket. The preceding method also includes two if statements that that will do check if the package is delayed on flight. Method, def def\_package() will take a key as a parameter and first it will look if the package is in the any of the bucket and if it is it will loop through the list and find the matching key if it exists. If not, it will return None. Method, def del\_package () will do the similar thing as the previous method. The difference is that if the matching key or package Id is found it will remove it from the list.

Inside the hashtable.py file there is a method get\_package\_info () that takes a csv file as a parameter , reads the csv file package\_file.csv line by line and it will append values from the csv file to package\_list which is an instance of PackageHashTable class. Values that will be appended are package Id as an integer and the second part is the resto of the line that includes address, city, state, zip, delivery deadline, weight of a package and special notes if present. The method will return a list of stored packages.

Hashtable.py file also contains get\_package\_by\_id() that is used inside the user\_inteface.py file It takes package\_info as a parameter which is in this case package id and it will look through the hash table and display the information about the package, if matched.

Going back to the algorithm chosen for the problem, I can conclude that the greedy algorithm was the good choice because in our case it will always choose the next closest available location from the list, but it usually requires sorting of choices from the list. Taking the next best choice is a linear sort. Greedy algorithm is a very good replacement for exhaustive search algorithms. In our case it worked out well since provided a solution that delivers all forty packages and trucks were driven only 101 miles.



Some other algorithms that could have been used for our problem are for example The Nearest Insertion Algorithm and Multi-Fragment Algorithm. The Nearest Insertion algorithm works that finds the two points that are furthest apart and connects them. After that identifies the furthest location from any point from the previously connected distance of the furthest points. Incorporates the last found point into the route and deletes the previously connected furthest distance. It repeats the process until only one point remains and it is the only choice to connect to complete the tour.

A Multi Fragment algorithm takes the list of all the distances and identifies two closest points and joins them together and forms one edge. Then it finds the next two location that are closest to each other and connects them. After this step it looks for two points where joining them will not result in a closed path and each point has an edge degree of either 0 or 1. If the point has a edge degree of 2 it cannot be connected to any other points. The steps are repeated until only two points remain with an edge degree less then two. Finally, joins them and the route is completed.

Looking at these two approaches greedy algorithm that looks for a shortest distance is more efficient because it needs to look for only one edge at the time that is the closest to the previously identified one while other two algorithms previously mentioned always look for two edges.

The one thing that I would maybe modify later in my program is the hash table which I initiated with 32 buckets. If the demand increases, it could be extended to 64 so it can have more buckets and each one of them can store list with multiple packages. As a result of having more buckets, each bucket will store less packages inside which will reduce the time of looking through the bucket. Method add\_package that is defined inside of a hash table will add the package based on package id from a csv file and its complexity is O (1). Get package method that searches for an existing package from the hash table has a complexity of O(N), the delete package method has a same complexity and its purpose is to remove a package from a hash table based on package id. By adding more packages to the hash table requires more time to loop through the backet and find the correct package. By adding more and more packages, performance of the program will be affected.

Instead of a hash table used to store our package information some other possible options would be a python built in data structures dictionary and list. Dictionary could have been used because it contains of key and value par which could work for our solution by storing package id as a key of the dictionary and rest of the row for that package could have been stored as a value of the dictionary. The other option are python lists. The package data can be stored in a list by nesting a dictionary in a list where each dictionary inside the list would be a package information. Comparing these, hash table that has been used for the project is more efficient because it contains buckets where we can place packages by simply applying package id number modulo size of a hash table. That operation places a package in a particular bucket.