**Core Algorithm Overview**

**Stated Problem:**

The purpose of this project is to create an algorithm written in Python in order to aim at solving the issues that WGUPS is having. The project is met with a few constraints. There is a total of 3 trucks and 2 drivers. There is a total of 40 packages that need to be delivered on a certain day. Some of these packages need to be delivered by a certain time, some need to be delivered along with others and some may have the wrong address which will be updated at a later time. The goal of this project is to deliver all packages by their delivery deadlines while traveling the least number of miles. In order to assist WGUPS in solving this problem I’ve decided to use a greedy algorithm. The reason that I chose to go with a greedy algorithm is because it is quick, simple, and finds the optimal route from the current node.

**Algorithm Overview:**

The greedy algorithm is created in the following manner:

1. A submatrix is created for each truck using the main csv matrix provided by WGU
2. The submatrix is then transformed into a nested list that contains the distance to all nodes from the current node in the following format:

[[start\_node, end\_node, distance]]

1. The algorithm then chooses to visit the node closest to it if it is not the hub and has not been visited
2. The process is repeated until the last destination for the packages on the truck have been delivered

The time complexity for this chosen algorithm is O(n^2). In order to traverse through the optimal path created for each truck, a sorted version of rows, columns, and distances is stored in a 2d array where values are accessed using dual nested for loops. The algorithm uses a for loop, O(n), twice in order to iterate through the optimal path list (truck\_one\_minimum). The algorithm then uses nested if statements O(1) in order to find the next node with the shortest distance from the current node.

**Greedy Algorithm**

**truck\_one\_minimum = [[node\_one, node\_two, distance], [node\_one, node\_three, distance],**

**[node\_two, node\_one, distance], [node\_two, node\_three, distance]**

**[node\_three, node\_one, distance], [node\_three, node\_two, distance]]**

The for loop of the algorithm iterates through each item of the list

**For each item in truck\_one\_minimum:**

**curr\_distance = distance**

**If item\_index[i] == current\_node:**

**If item\_index[i + 1] == next node and distance <= curr\_distance:**

**curr\_distance = distance**

**truck\_one\_route\_opt.append(item)**

The nested if statements pull the smallest distance from the current node to the next node and

appends it to the truck\_one\_route\_opt that will store the shortest destinations for each package

address on the truck.

The pseudo code above shows a general overview of how the greedy algorithm for this program is used to determine the shortest route for each truck based on the packages that are currently loaded

**Time Complexity:**

**TruckLoading.py**

Line 19: for row in package\_data: Space-Complexity = O(n) Time-Complexity = O(n)

Line 77: get\_package\_hash\_map: Space-Complexity = O(1) Time-Complexity = O(1)

Line 81: truck\_one\_package\_list: Space-Complexity = O(1) Time-Complexity = O(1)

Line 85: truck\_two\_package\_list: Space-Complexity = O(1) Time-Complexity = O(1)

Line 89: truck\_three\_package\_list: Space-Complexity = O(1) Time-Complexity = O(1)

**Trucks.py**

Line 6: truck\_one: Space-Complexity = O(1) Time-Complexity = O(1)

Line 7: truck\_two: Space-Complexity = O(1) Time-Complexity = O(1)

Line 8: truck\_three: Space-Complexity = O(1) Time-Complexity = O(1)

Line 11: truck\_speed: Space-Complexity = O(1) Time-Complexity = O(1)

Line 14: truck\_one\_departure: Space-Complexity = O(1) Time-Complexity = O(1)

Line 15: truck\_two\_departure: Space-Complexity = O(1) Time-Complexity = O(1)

Line 16: truck\_three\_departure: Space-Complexity = O(1) Time-Complexity = O(1)

Line 19: truck\_one\_route\_times: Space-Complexity = O(1) Time-Complexity = O(1)

Line 22: truck\_one\_delivery\_times: Space-Complexity = O(1) Time-Complexity = O(1)

Line 25: truck\_one\_cumulative\_times: Space-Complexity = O(1) Time-Complexity = O(1)

Line 28: truck\_one\_cumulative\_times\_converted: Space-Complexity = O(1) Time-Complexity = O(1)

Line 32: truck\_two\_route\_times: Space-Complexity = O(1) Time-Complexity = O(1)

Line 35: truck\_two\_delivery\_times: Space-Complexity = O(1) Time-Complexity = O(1)

Line 38: truck\_two\_cumulative\_times: Space-Complexity = O(1) Time-Complexity = O(1)

Line 41: truck\_two\_cumulative\_times\_converted: Space-Complexity = O(1) Time-Complexity = O(1)

Line 44: truck\_three\_route\_times: Space-Complexity = O(1) Time-Complexity = O(1)

Line 47: truck\_three\_delivery\_times: Space-Complexity = O(1) Time-Complexity = O(1)

Line 50: truck\_three\_cumulative\_times: Space-Complexity = O(1) Time-Complexity = O(1)

Line 53: truck\_three\_cumulatvie\_times\_converted: Space-Complexity = O(1) Time-Complexity = O(1)

Line 57: truck\_time: Space-Complexity = O(1) Time-Complexity = O(1)

Line 63: delivery\_time: Space-Complexity = O(1) Time-Complexity = O(1)

Line 69: truck\_one\_time\_convert: Space-Complexity = O(1) Time-Complexity = O(1)

Line 74: truck\_two\_time\_convert: Space-Complexity = O(1) Time-Complexity = O(1)

Line 79: truck\_three\_time\_convert: Space-Complexity = O(1) Time-Complexity = O(1)

**TruckRouting.py**

Line 17: for i in range(len(read\_distance)):

for j in range(len(read\_distance)):

Space-Complexity = O(n^2) Time-Complexity = O(n^2)

Line 22: matrix: Space-Complexity = O(1) Time-Complexity = O(1)

Line 26: truck\_one\_package\_list: Space-Complexity = O(1) Time-Complexity = O(1)

Line 27: truck\_one\_route: Space-Complexity = O(1) Time-Complexity = O(1)

Line 29: truck\_two\_package\_list: Space-Complexity = O(1) Time-Complexity = O(1)

Line 30: truck\_two\_route Space-Complexity = O(1) Time-Complexity = O(1)

Line 32: truck\_three\_package\_list: Space-Complexity = O(1) Time-Complexity = O(1)

Line 33: truck\_three\_route: Space-Complexity = O(1) Time-Complexity = O(1)

Line 37: truck\_one\_indices: Space-Complexity = O(1) Time-Complexity = O(1)

Line 38: truck\_two\_indices: Space-Complexity = O(1) Time-Complexity = O(1)

Line 39: truck\_three\_indices: Space-Complexity = O(1) Time-Complexity = O(1)

Line 42: for row in truck\_one\_package\_list: Space-Complexity = O(n) Time-Complexity = O(n)

Line 48: for row in truck\_two\_package\_list: Space-Complexity = O(n) Time-Complexity = O(n)

Line 54: for row in truck\_three\_package\_list: Space-Complexity = O(n) Time-Complexity = O(n)

Line 62: for row in destination\_name\_data

for I in truck\_one\_route:

Space-Complexity = O(n^2) Time-Complexity = O(n^2)

Line 69: for row in destination\_name\_data

for I in truck\_two\_route:

Space-Complexity = O(n^2) Time-Complexity = O(n^2)

Line 76: for row in destination\_name\_data

for I in truck\_three\_route:

Space-Complexity = O(n^2) Time-Complexity = O(n^2)

**GreedyAlgo.py**

Line 30: truck\_one\_path\_distance: Space-Complexity = O(1) Time-Complexity = O(1)

Line 31: truck\_one\_minimum: Space-Complexity = O(1) Time-Complexity = O(1)

Line 32: truck\_one\_route\_opt: Space-Complexity = O(1) Time-Complexity = O(1)

Line 33: truck\_one\_route\_path: Space-Complexity = O(1) Time-Complexity = O(1)

Line 34: truck\_one\_total\_time: Space-Complexity = O(1) Time-Complexity = O(1)

Line 35: truck\_one\_keys: Space-Complexity = O(1) Time-Complexity = O(1)

Line 36: truck\_one\_values: Space-Complexity = O(1) Time-Complexity = O(1)

Line 37: for row in truck\_one\_indices:

for col in truck\_one\_indices:

Space-Complexity = O(n^2) Time-Complexity = O(n^2)

Line 44: for row in truck\_one\_path\_distance: Space-Complexity = O(n) Time-Complexity = O(n)

Line 52: for row in truck\_one\_minimum: Space-Complexity = O(n) Time-Complexity = O(n)

Line 86: for row in truck\_one\_route\_opt: Space-Complexity = O(n) Time-Complexity = O(n)

Line 93: for i in truck\_one\_route\_path: Space-Complexity = O(n) Time-Complexity = O(n)

Line 98: for i in truck\_one\_route\_times: Space-Complexity = O(n) Time-Complexity = O(n)

Line 101: for i in truck\_one\_delivery\_times: Space-Complexity = O(n) Time-Complexity = O(n)

Line 105: for i in truck\_one\_cumulative\_times: Space-Complexity = O(n) Time-Complexity = O(n)

Line 109: for i in truck\_one\_indices: Space-Complexity = O(n) Time-Complexity = O(n)

Line 112: for i in truck\_one\_cumulative\_times\_converted: Space-Complexity = O(n)

Time-Complexity = O(n)

Line 116: truck\_one\_dictionary: Space-Complexity = O(1) Time-Complexity = O(1)

Line 124: truck\_two\_path\_distance: Space-Complexity = O(1) Time-Complexity = O(1)

Line 125: truck\_two\_minimum: Space-Complexity = O(1) Time-Complexity = O(1)

Line 126: truck\_two\_route\_opt: Space-Complexity = O(1) Time-Complexity = O(1)

Line 127: truck\_two\_route\_path: Space-Complexity = O(1) Time-Complexity = O(1)

Line 128: truck\_two\_total\_time: Space-Complexity = O(1) Time-Complexity = O(1)

Line 129: truck\_two\_keys: Space-Complexity = O(1) Time-Complexity = O(1)

Line 130: truck\_two\_values: Space-Complexity = O(1) Time-Complexity = O(1)

Line 131: for row in truck\_two\_indices:

for col in truck\_two\_indices:

Space-Complexity = O(n^2) Time-Complexity = O(n^2)

Line 138: for row in truck\_two\_path\_distance: Space-Complexity = O(n) Time-Complexity = O(n)

Line 146: for row in truck\_two\_minimum: Space-Complexity = O(n) Time-Complexity = O(n)

Line 189: for row in truck\_two\_route\_opt: Space-Complexity = O(n) Time-Complexity = O(n)

Line 196: for i in truck\_two\_route\_path: Space-Complexity = O(n) Time-Complexity = O(n)

Line 201: for i in truck\_two\_route\_times: Space-Complexity = O(n) Time-Complexity = O(n)

Line 204: for i in truck\_two\_delivery\_times: Space-Complexity = O(n) Time-Complexity = O(n)

Line 208: for i in truck\_two \_cumulative\_times: Space-Complexity = O(n) Time-Complexity = O(n)

Line 212: for i in truck\_two \_indices: Space-Complexity = O(n) Time-Complexity = O(n)

Line 215: for i in truck\_two\_cumulative\_times\_converted: Space-Complexity = O(n)

Time-Complexity = O(n)

Line 227: truck\_three \_dictionary: Space-Complexity = O(1) Time-Complexity = O(1)

Line 228: truck\_three \_path\_distance: Space-Complexity = O(1) Time-Complexity = O(1)

Line 229: truck\_three \_minimum: Space-Complexity = O(1) Time-Complexity = O(1)

Line 230: truck\_three \_route\_opt: Space-Complexity = O(1) Time-Complexity = O(1)

Line 231: truck\_three \_route\_path: Space-Complexity = O(1) Time-Complexity = O(1)

Line 231: truck\_three \_total\_time: Space-Complexity = O(1) Time-Complexity = O(1)

Line 232: truck\_three \_keys: Space-Complexity = O(1) Time-Complexity = O(1)

Line 233: truck\_three \_values: Space-Complexity = O(1) Time-Complexity = O(1)

Line 234: for row in truck\_three\_indices:

for col in truck\_three \_indices:

Space-Complexity = O(n^2) Time-Complexity = O(n^2)

Line 241: for row in truck\_three\_path\_distance: Space-Complexity = O(n) Time-Complexity = O(n)

Line 249: for row in truck\_three\_minimum: Space-Complexity = O(n) Time-Complexity = O(n)

Line 290: for row in truck\_three\_route\_opt: Space-Complexity = O(n) Time-Complexity = O(n)

Line 297: for i in truck\_three\_route\_path: Space-Complexity = O(n) Time-Complexity = O(n)

Line 302: for i in truck\_three\_route\_times: Space-Complexity = O(n) Time-Complexity = O(n)

Line 305: for i in truck\_three \_delivery\_times: Space-Complexity = O(n) Time-Complexity = O(n)

Line 309: for i in truck\_three\_cumulative\_times: Space-Complexity = O(n) Time-Complexity = O(n)

Line 313: for i in truck\_three\_indices: Space-Complexity = O(n) Time-Complexity = O(n)

Line 316: for i in truck\_three\_cumulative\_times\_converted: Space-Complexity = O(n)

Time-Complexity = O(n)

Line 320: truck\_three\_dictionary: Space-Complexity = O(1) Time-Complexity = O(1)

Line 323: total\_route\_distance: Space-Complexity = O(1) Time-Complexity = O(1)

**HashTable.py**

Line 3: \_\_init\_\_: Space-Complexity = O(1) Time-Complexity = O(1)

Line 10 \_get\_hash: Space-Complexity = O(1) Time-Complexity = O(1)

Line 17 insert: Space-Complexity = O(n) Time-Complexity = O(n)

Line 35 update: Space-Complexity = O(1) Time-Complexity = O(1)

Line 47 get: Space-Complexity = O(n) Time-Complexity = O(n)

Line 57 delete: Space-Complexity = O(n) Time-Complexity = O(n)

Line 68 print: Space-Complexity = O(n) Time-Complexity = O(n)

**Main.py**

Line 15: user\_input == ‘lookup’: Space-Complexity = O(1) Time-Complexity = O(1)

Line 24: if user\_input: Space-Complexity = O(1) Time-Complexity = O(1)

Line 25: lookup\_input: Space-Complexity = O(1) Time-Complexity = O(1)

Line 26: time\_input: Space-Complexity = O(1) Time-Complexity = O(1)

Line 27: package\_status: Space-Complexity = O(1) Time-Complexity = O(1)

Line 28: package\_truck: Space-Complexity = O(1) Time-Complexity = O(1)

Line 29: if lookup\_input in truck\_one\_dictionary[lookup\_input]:

Space-Complexity = O(1) Time-Complexity = O(1)

Line 39: elif lookup\_input in truck\_two\_dictionary: Space-Complexity = O(1) Time-Complexity = O(1)

Line 48: elif lookup\_input in truck\_three\_dictionary: Space-Complexity = O(1) Time-Complexity = O(1)

Line 60: print: Space-Complexity = O(n) Time-Complexity = O(n)

Line 81: elif user\_input == ‘all’: Space-Complexity = O(1) Time-Complexity = O(1)

Line 82: time\_input: Space-Complexity = O(1) Time-Complexity = O(1)

Line 83: package\_status: Space-Complexity = O(1) Time-Complexity = O(1)

Line 84: package\_truck: Space-Complexity = O(1) Time-Complexity = O(1)

Line 85: for item in truck\_one\_dictionary: Space-Complexity = O(n) Time-Complexity = O(n)

Line 94: print: Space-Complexity = O(n) Time-Complexity = O(n)

Line 111: for item in truck\_two\_dictionary: Space-Complexity = O(n) Time-Complexity = O(n)

Line 120: print: Space-Complexity = O(n) Time-Complexity = O(n)

Line 136: for item in truck\_three\_dictionary: Space-Complexity = O(n) Time-Complexity = O(n)

Line 45: print: Space-Complexity = O(n) Time-Complexity = O(n)

Total Space-Complexity = O(n^2)

Total Time-Complexity = O(n^2)

**Overhead:**

Overall, the program provides a solution to the WGUPS routing problem in quadratic O(n^2) time. Since this is a console application, variables such as firewalls, protocols, and network speeds don’t need to be taken into consideration. Modern computers shouldn’t run into any problems regarding computational time and memory management when running this program.

**Explanation of Data Structure and Its Efficiency:**

The packages within the program are stored using a combination of a hash table and a list. In TruckLoading.py starting at line 19 I used a for loop to assign the value of each column in a row to a variable. In line 34 of TruckLoading.py I store the previously iterated variables into a list which then gets inserted as a value into the hash table. The reason I went about using the hash table in this manner is so that I could access packages using the package ID as a key and a value-index pair when calling the get, update, and delete functions to reference the attributes of packages (ID, address, city, etc.). For example, if I were to call get\_package\_hash\_map().get(‘key’, value) it would return the entire list of attributes for the package that’s being called. However, instead of having to get, update or delete the entire list when making changes I now only have to add an index to the value, (‘key’, value[index]) in order to make changes to a specific attribute for the desired package.

In addition to the hash table, I used a series of 2D lists to normalize the data from the original 26x26 matrix to the Nx3 matrix created for each truck using the Greedy Algorithm. We start with the normalization of the 26x26 matrix in the line 37 of GreedyAlgo.py file. The double for loop uses the destinations for truck one’s route as references and creates a NxN matrix based on the packages that are stored on the truck. In this case of truck one it is a 10x10 matrix. Once this submatrix has been created, the program then iterates through the submatrix and create a new Nx3 matrix so that the algorithm can calculate the shortest distance from the current node to the next node. This Nx3 matrix is stored in a 2D list in the following format [[start\_node], [end\_node], [distance]].

**Possible Data Structures:**

Data structures that would satisfy the requirements of the project are the graph data structure and queues. The graph data structure could be used to create a graph for each submatrix of the trucks. Then, using Dijkstra’s algorithm, the optimal route for each graph could be determined with the implementation of a priority queue.

**Possible Algorithms and Their Differences:**

Algorithms that would satisfy the requirements of the project include Dijkstra’s Algorithm and Floyd-Warshall’s algorithm. Dijkstra’s algorithm would apply to this project since were are given a single vertex from where our path begins. The algorithm works by initializing all vertices in a graph to infinity. The algorithm then traverses all possible paths for the graph while storing and updating distance values from the source as new vertices are discovered. Floyd-Warshall’s algorithm is used to compute the shortest path between all pairs of nodes. In fact, this is the algorithm that I had originally chosen. I have an old computer and wondered why it would take the program upwards of 5 minutes just to run. I then realized that computational complexity is O(n^3) which is what was causing the latency in my program, so I decided against it.

**Different Approach:**

If I had to do the project over again, I would Implement Dijkstra’s algorithm and create a Graph class.

Since each already truck creates its own submatrix based on the packages that are loaded, the trucks will have their own graphs which would call to Dijkstra’s algorithm in order to find the optimal path for the current route.

**Adaptability and Scalability:**

Due to the way that the packages are loaded onto the HashMap the program has the ability to adapt and scale with the total number of packages that need to be delivered. If the total number of destinations were to increase, the program will adjust by creating a new route for each truck based on the packages that are loaded.

**Efficiency and Maintainability:**

The program is efficient in that it chooses the shortest distance to the next destination available at the time of the truck’s current location. While this may not be the shortest overall path it is most certainly the quickest. Since the csv files are loaded into the program it will be fairly simple to maintain. Not many values are hard-coded into the program so if one wanted to change the destinations and distances within the csv files the program would adjust the routes, paths, times, distances, etc.

**Strengths and Weaknesses:**

One of the strengths of this program is the abundant use of lists. Lists are simple, adaptable, and are quick to traverse. The most obvious weakness that I can think of within my program in terms of scalability is the truck package limit. There are currently only three trucks that can each hold 16 packages. This limits the current program’s scalability with csv files to only 48 packages. Another weakness is the assumption that all package IDs are unique. If one were to input duplicate packages IDs the program wouldn’t be able to handle collisions.