Core Algorithm Overview

**Section 1: Programming/Coding**

**A. Identify the algorithm that will be used to create a program to deliver the packages and meets *all* requirements specified in the scenario.**

*I will be using a simple greedy algorithm that favors to the node with the lowest distance.*

**B. Write a core algorithm overview, using the sample given, in which you do the following:**

1. Comment using pseudocode to show the logic of the algorithm applied to this software solution.

*Step 1: Create a list of nodes from the list of addresses given.*

*Step 2: Create a list of edges based on the distances given.*

*Step 3: After loading the package onto the truck, sort the packages based on the distance from the current node at each delivery point.*

*Step 4: Choose the closest node to you and deliver all packages there.*

*Step 5: Repeat until completed.*

2. Apply programming models to the scenario.

*To model this problem, an Application exists. This Application creates a Clock and three Trucks. It then creates a Queue of Packages, a Graph of Nodes and Edges, and uses the Trucks to load the packages. These Trucks deliver the packages to the Nodes in question via the Edges. After each Truck delivers their last Package, they return to the Hub Node and determine if each package has been delivered. If not, they load their next shipment and continue delivering Packages. If all the Packages have been reported DELIVERED by the Hub Node, the Application terminates.*

3. Evaluate space-time complexity using Big O notation throughout the coding and for the entire program.

***main.py [Space Complexity / Time Complexity]***

Line 51: O(N) / O(N)

Line 56: O(1) / O(N log N)

Line 154: O(N) / O(N)

Line 169: O(1) / O(N)

Line 203: O(N) / O(N)

Line 208: O(N) / O(N)

*Total: O(N) / O(N log N)*

***utils/entities.py [Space Complexity / Time Complexity]***

Line 134: O(N) / O(N)

Line 146: O(1) / O(N log N)

*Total: O(N) / O(N log N)*

***utils/graph.py [Space Complexity / Time Complexity]***

Line 71: O(N) / O(N)

Line 76: O(N) / O(N)

Line 83: O(N) / O(1)

Line 92: O(1) / O(N^2)

Line 107: O(N) / O(N)

Line 116: O(N) / O(N)

*Total: O(N) / O(N^2)*

4. Discuss the ability of your solution to adapt to a changing market and to scalability.

*This solution can handle a solid number of packages but when we reach a high amount of packages, there will be slowdown based on how we traverse the package instances and how much data we’re sorting. The solution can adapt to a changing market by adding more nodes and edges to the graph since each node is a location and each edge represents the distance between them. However, slowdown may be expected as we load in larger datasets.*

5. Discuss the efficiency and maintainability of the software.

*The program is efficient enough to handle small to medium use cases. We successfully delivered 40 packages with confounding variables on time, under 140 miles, and with 4.5 hours to spare. If we double the number of packages going to the same place, we would still make it before the end of the day. Further testing is required to achieve efficiency with large datasets. The maintainability is admittedly fragile since the number of packages loaded into each truck is hard-coded. This can be mitigated by moving the number of packages loaded into each truck into variables and removing the resupply constraints.*

6. Discuss the self-adjusting data structures chosen and their strengths and weaknesses based on the scenario.

*The strength of the queues are how easy they are to manage via push/pop operations. The weakness of the queues come from the lack of a lookup table to quickly find a single item inside. Additionally, iterating over the queue can be tricky depending on the data structures contained inside. If the queue needs to adapt due to sorting priorities based on new information, there could be slowdown during runtime since it has to sort with a complexity of O(N log N).*

C. Write an original code to solve and to meet the requirements of lowest mileage usage and having *all* packages delivered on time.

1. Create a comment within the first line of your code that includes your first name, last name, and student ID.

2. Include comments at *each* block of code to explain the process and flow of the coding.

**D. Identify a data structure that can be used with your chosen algorithm to store the package data.**

1. Explain how your data structure includes the relationship between the data points you are storing.

*I chose a priority queue combined with a graph and greedy algorithm to determine which packages to deliver and when to deliver them. The relationship to the data points are based on how far away each package needs to go and when it needs to be there.*

**E. Develop a hash table, without using any additional libraries or classes, with an insertion function that takes the following components as input and inserts the components into the hash table:**

• package ID number

• delivery address

• delivery deadline

• delivery city

• delivery zip code

• package weight

• delivery status (e.g., delivered, in route)

**F. Develop a look-up function that takes the following components as input and returns the corresponding data elements:**

• package ID number

• delivery address

• delivery deadline

• delivery city

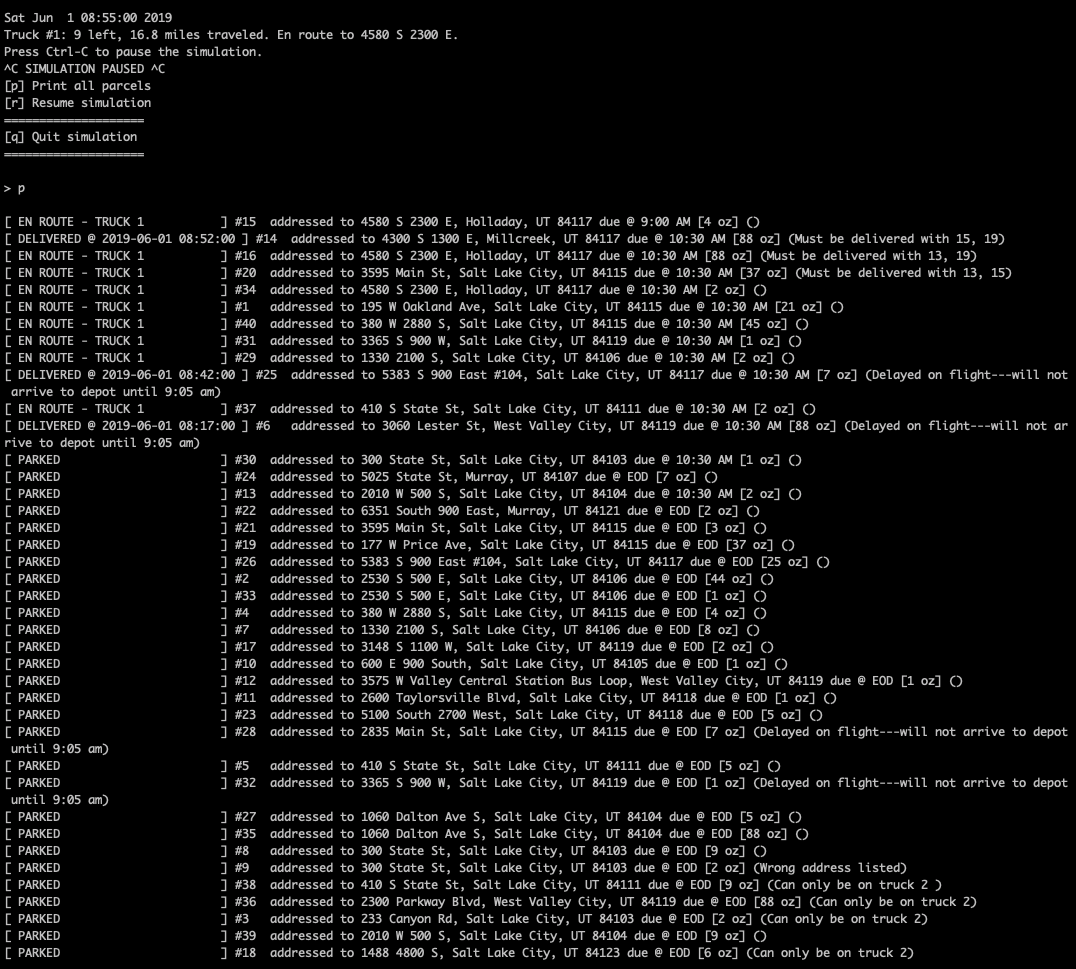
• delivery zip code

• package weight

• delivery status (e.g., delivered, in route)

**G. Provide an interface for the insert and look-up functions to view the status of any package at any time. This function should return all information about eachpackage, including delivery status.**

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



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

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



H. Run your code and provide screenshots to capture the complete execution of your code.



**Section 2: Annotations**

**I. Justify your choice of algorithm by doing the following:**

1. Describe *at least*  two strengths of the algorithm you chose.

*\* The code for a greedy algorithm is simple to implement.*

*\* Depending on the implementation and datasets provided, it can be very quick.*

2. Verify that the algorithm you chose meets *all* the criteria and requirements given in the scenario.

*Based on the following picture, I can confirm that my algorithm successfully delivers all packages on time and the total miles traveled by all trucks is 126.6 miles.*



3. Identify two other algorithms that could be used and would have met the criteria and requirements given in the scenario.

*Dijkstra’s Algorithm could be used to shorten the distance traveled even further. This would require adding weights to each part of the graph and determining the route to take before leaving the Hub.*

*Miller-Tucker-Zemlin could also be used but it would require that “there is only a single tour covering all cities” ("Travelling salesman problem", 2019). You could ignore all edges over a certain threshold and try creating a single tour that way.*

a. Describe how *each* algorithm identified in part I3 is different from the algorithm you chose to use in the solution.

*Dijkstra’s Algorithm requires more setup and consideration of heuristics than a simple greedy algorithm. The route would have to be completely calculated before embarking from the hub where as the greedy algorithm can just be compared as you go. The same can be said for Miller-Tucker-Zemlin except the edges of the graph would have to be reduced to two edges per city. This can work if predefined routes were an additional constraint to this problem.*

**J. Describe what you would do differently if you did this project again.**

*I would implement Dijkstra’s instead of a naïve greedy algorithm. This would require an additional function to determine the best route to take based on distance and another function to do proper backtracking. After the route has been calculated, I have to remove the old algorithm, sort the queue of the truck based on the new route, and deliver the packages in order.*

**K. Justify your choice of data structure by doing the following:**

1. Verify that the data structure you chose meets all the criteria and requirements given in the scenario.

*As shown below, my data structure does ensure the least number of total miles, all packages delivered on time, the hash and look-up function work, and the reporting is accurate.*

**

a. Describe the efficiency of the data structure chosen.

*A queue is O(1) for push and pop operations. While a pure queue would lack the ability to inspect the elements inside, this queue is a bit relaxed in its methodology. Iterating over said queue would be O(N). This queue contains a list of Package instances and these instances are iterated over to produce the report you see above.*

b. Explain the expected overhead when linking to the next data item.

*There is no linking involved. We always pull from the top of the queue. As a result of that, pulling from the top of the queue is always O(1), memory used is equal to the size of a single Package instance, and no bandwidth is used since there is a lack of networked operations.*

c. Describe the implications of when more package data is added to the system or other changes in scale occur.

*Sorting and traversing becomes slower as the dataset grows but the push/pop operations should remain constant. When the number of packages are increased, there will be a limit imposed by the maximum capacity of the trucks and the deadline of each package (in this case, end-of-day/1700 hours). Increasing the number of truck operating simultaneously will mitigate this issue temporarily but as the number of packages continue to rise, you’ll still run into this issue. If the number of cities increase, the distance traveled will increase which will increase the risk of not delivering packages on time. If additional instances of this system are deployed across regions, this limit stops being an issue.*

2. Identify two other data structures that can meet the same criteria and requirements given in the scenario.

*I could use a linked list and construct the package order based on the initial list.*

*I could also use a stack which would be the most analogous to my current implementation.*

a. Describe how *each* data structure identified in part K2 is different from the data structure you chose to use in the solution.

*The stack is different than my priority queue because there’s no builtin iterable property to a linked list. You have to traverse each link to obtain a subsection of said structure.*

*The difference between the stack and the queue is only being able to push and pop from the top of the structure and I would have to load the packages in reverse order.*

**L. Acknowledge sources, using in-text citations and references, for content that is quoted, paraphrased, or summarized.**

*Travelling salesman problem. (2019, July 14). Retrieved July 15, 2019, from* [*https://en.wikipedia.org/wiki/Travelling\_salesman\_problem*](https://en.wikipedia.org/wiki/Travelling_salesman_problem)

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