C950 Data Structures and Algorithms II

Basudev Rijal,

Student ID: 001365006,

[brijal1@wgu.edu](mailto:brijal1@wgu.edu)

# **Algorithm Identification**

The core algorithm ‘WGUPS - Packet Tracking Console’ uses a greedy approach to determine an efficient route and delivery distribution for Daily Local Deliveries (DLD). It solves the puzzle in three stages. During the first stage, the ‘Packet\_filtering’ system is loaded on 3 different trucks bases on the similarity of packets, special requirements, and given assumptions and restrictions. In the second stage, the ‘Get\_fastest\_route’ system finds the fastest route for each of the trucks to deliver assigned packages. And on the last stage, each packet's movements are tracked using the ‘Time\_tracking’ system’. The time records are further used to review the reports.

# **B1. Logic Comments**

**Packet\_filtering system**

This system filters all the 40 packets in 3 different tracks based on the requirements of each packet and the similarity on them. There are three main-filers and additional sub-filters inside them to assign the packets to the truck in the current solution manually.

**Pseudocode:**

# Condition 1: IF Deadline != ‘EOD’

1.1: IF Sp. Note = 'Delayed on flight…….’ 🡪 assign to truck 2

1.2: ELIF Street address = '3365 S 900 W' ’ 🡪 assign to truck 2

1.3: ELSE 🡪 assign to truck 1

# Condition 2: IF Deadline = ‘EOD’ AND Sp. Note != ‘NONE’

2.1: IF Sp. Note = Wrong address listed ’ 🡪 assign to truck 3

2.2: ELIF Zip = '84103 ’ 🡪 assign to truck 3

2.3: ELSE 🡪 assign to truck 2

# Condition 3: IF Deadline = ‘EOD’ AND Sp. Note != ‘NONE’

3.1: IF Street address = IN ['177 W Price Ave ’ , '2010 W 500 S' , '1330 2100 S' , '3148 S 1100 W' ,

'3575 W Valley Central Station bus Loop'] 🡪 assign to truck 1

3.2: ELSE

3.2.1 IF Zip = IN ['84103 ’ , '84111 ’ , '84117 ’ , '84119 ’ , '84103 ‘]

3.2.1.1 IF Street address = '300 State St’ 🡪 assign to truck 3

3.2.1.2 ELSE 🡪 assign to truck 2

3.2.2 ELSE 🡪 assign to truck 3

**STAGE2:** **Get\_fastest\_route system**

After loading, The Get\_fastest\_route system determines the shortest possible route for each truck. In the beginning, its sets the hub address as a current address. the system uses a greedy approach to compare the distance from the current address to all other remaining addresses of packages to find the next stop for the truck. The current address changes as the truck move each time to find the next closest stop again. In the end, the system calculates the sum of the total distance traveled by each truck.

**Pseudocode:**

Current\_address = initialize the current address hub address

Total\_distance\_tr1= 0

while there are packets on truck’s list to deliver:

delivery\_order = sort remaining packages by distance from current address to delivery address

next\_closest\_stop = obtain the next closest stop from the delivery\_order

distance = distance from Current\_location to next\_closest\_stop

Total\_distance\_tr1 = Total\_distance + distance

When the last packet on the truck 1’s assignment list delivered

return

Total\_distance\_tr1

….

….

return

Total\_distance = Total\_distance\_tr1+ Total\_distance\_tr2 + Total\_distance\_tr3

**STAGE3: Time\_tracking system**

The time tracking system assigns the initial movement time for each truck. Each time the truck moves, it calculates the time for the distance traveled. when the truck delivers a packet, it calculates the delivery time for that packet and stores it on the packet’s delivery status record.

**Pseudocode:**

Truck1\_init\_movement\_time: 08:00:00

Current\_location = hub

When Current\_location = packet x’s location

#Retrieve the Total\_distance\_tr1 from STAGE 2 for that delivery

Delivery status = Total\_distance\_tr1/18 (convert to the timestamp HH:MM: SS)

Store it to the packet’s Delivery status record

When the last packet on the truck 1’s assignment list delivered

Total\_time\_tr1 = Total\_distance\_tr1/18 (convert to the timestamp HH:MM: SS)

…

…

Total\_time = Total\_time\_tr1 + Total\_time\_tr2 + Total\_time\_tr3

# B2. Development Environment

The ‘WGUPS - Packet Tracking Console’ is written in Python 3.9.0 using PyCharm 2021.1.3 (Professional Edition) IDE. The PyCharm uses Command Line Interface (CUI) to execute and generate the reports. The source files (.py files) are stored on a localhost machine. The program is designed to run on any standalone computer with the following minimum specifications.

• Any CPU (Intel i5/ i7/ AMD Ryzen 7)

• 4 GB RAM, 10 GB HDD Free Space.

# B3. Space-Time and Big-O

BIG O calculation

**hash\_table.py**

| **method** | **Time-Space Complexity** |
| --- | --- |
| \_\_init\_\_ | O(n) |
| bucket\_list | O(1) |
| insert | O(1) |
| update | O(n) |
| remove | O(n) |
| search | O(n) |
| **Total** | 2\* O(1) + 4\*O(n) = **O(n)** |

**csv\_reader.py**

| **method, line** | **Time-Space Complexity** |
| --- | --- |
| line 68: hash\_table. insert | O(n) |
| get\_first\_tr\_pk\_list | O(1) |
| get\_second\_tr\_pk\_list | O(1) |
| get\_third\_tr\_pk\_list | O(1) |
| get\_hash\_table | O(1) |
| **Total** | 4\* O(1) + 1\*O(n) = **O(n)** |

**distances.py**

| **method** | **Time-Space Complexity** |
| --- | --- |
| get\_address | O(n) |
| get\_distance | O(1) |
| get\_current\_distance | O(1) |
| get\_time | O(n) |
| find\_fastest\_route | O(n^2) |
| get\_first\_tr\_sort\_index | O(1) |
| get\_first\_tr\_sort\_list | O(1) |
| get\_second\_tr\_sort\_index | O(1) |
| get\_second\_tr\_sort\_list | O(1) |
| get\_third\_tr\_sort\_index | O(1) |
| get\_third\_tr\_sort\_list | O(1) |
| **Total** | 8\* O(1) + 2\*O(n) + O(n^2) = **O(n^2)** |

**packages.py**

| **method, line** | **Time-Space Complexity** |
| --- | --- |
| Line 22, 57,94: insert delivery start time | 3\*O(n) |
| Line 26: Compare first truck addresses to full address list | 3\*O(n^2) |
| Line 40:Calculate distance for each packet | 3\*O(n) |
| total\_distance\_all\_tr | 0(1) |
| **Total** | 3\*O(n) + 3\*O(n^2) + 3\*O(n) + 0(1) = **O(n^2)** |

**main.py**

| **method** | **Time-Space Complexity** |
| --- | --- |
| print\_status | O(1) |
| print\_pk\_detail | O(1) |
| Line 36: class main | 2\*O(n) |
| **Total** | 2\* O(1) + 2\*O(n) = **O(n)** |

The total space-time complexity of the entire is :

= (hash\_table.py + csv\_reader.py+ distances.py+ packages.py+ main.py)

= O(n)+ O(n)+ O(n^2)+ O(n^2)+ O(n)

= 3 \*O(n)+2 \*O(n^2)

**= O(n^2)**

# B4. Scalability and Adaptability

The main concern when designing this program was adaptability. The chaining hash table can grow dynamically to accept new packages as needed. The package, address, and distance data can be loaded from different CSV files.

We can add as many trucks as possible and packets easily through the csv\_reader file. We can scale up hash table size easily to minimize search time if the packet size grows comparatively.

# B5. Software Efficiency and Maintainability

The core algorithm developed for ‘WGUPS - Packet Tracking Console’ was implemented using a clean, object-oriented architecture designed to achieve efficiency and maintainability. All of the major components (class, objects, and methods) are well-commented to ensure readability. The new developer willing to work on this codebase should be able to ascertain the purpose of each block of codes and their role in the big picture.

While taking steps to increase adaptability it also increased maintainability as a byproduct. Each method has clear and specific instructions assigned and is well commented for readability. The well-commented coding will contribute heavily toward future maintainability.

# B6. Self-Adjusting Data Structures

The chaining hash table ensures that we never run out of space and there will be no hash collision. As we scale up the entries, all that happens is that the linked lists just keep on growing. Insertion and deletion are extremely simple to implement—indeed.

As we know chaining hash is simply an array of linked lists. The problem here is that the linked lists grow longer and longer and longer. It will make grow the worst-case space-time complexity at the rate of O(n) for n items in the linked list. This can be eliminated by increasing the size of the hash as the packet numbers grow.

# D. Data Structure

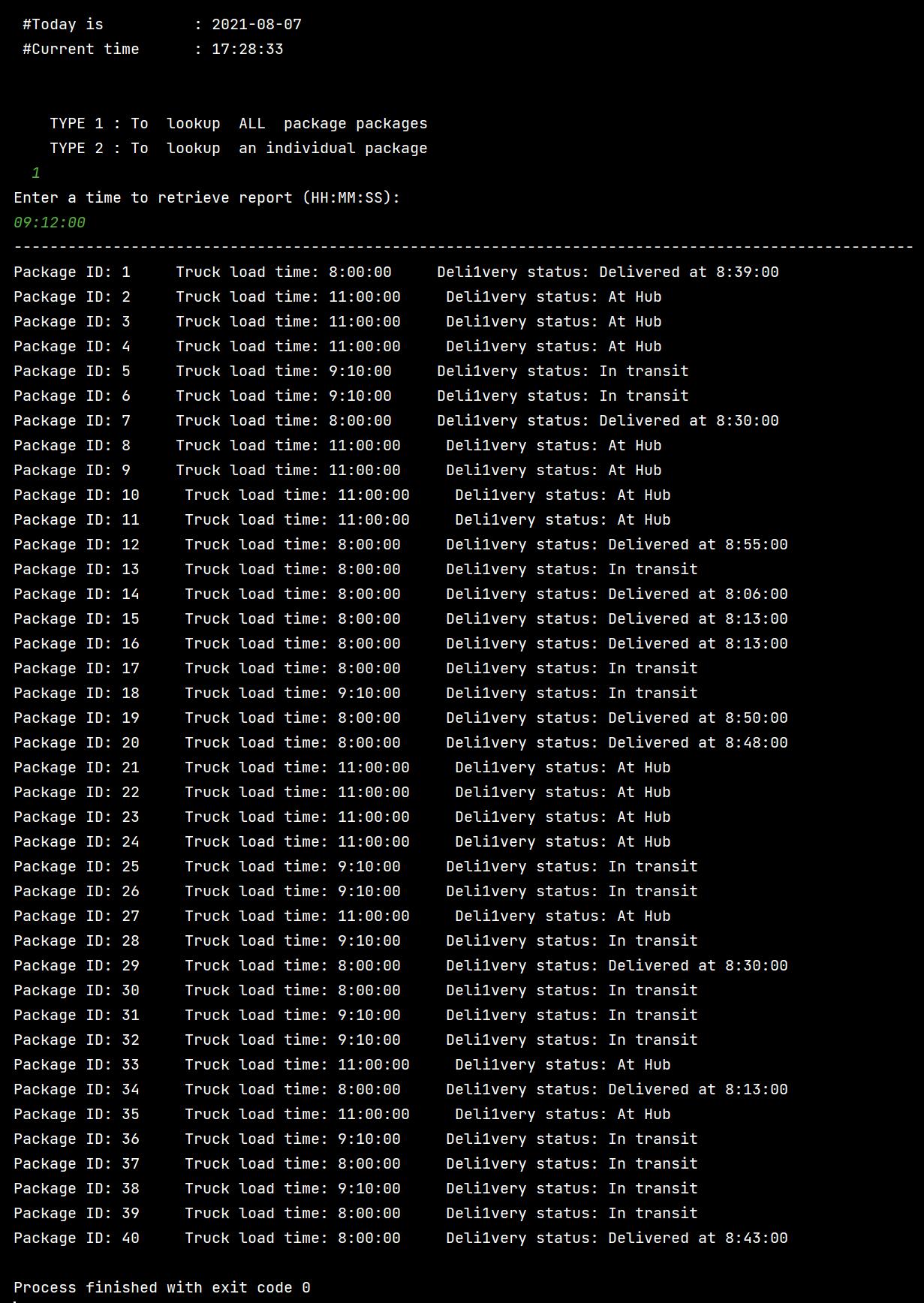
We used the chaining hash table with linear probing for searching as a self-adjusting data structure for the current solution.

# D1. Explanation of Data Structure

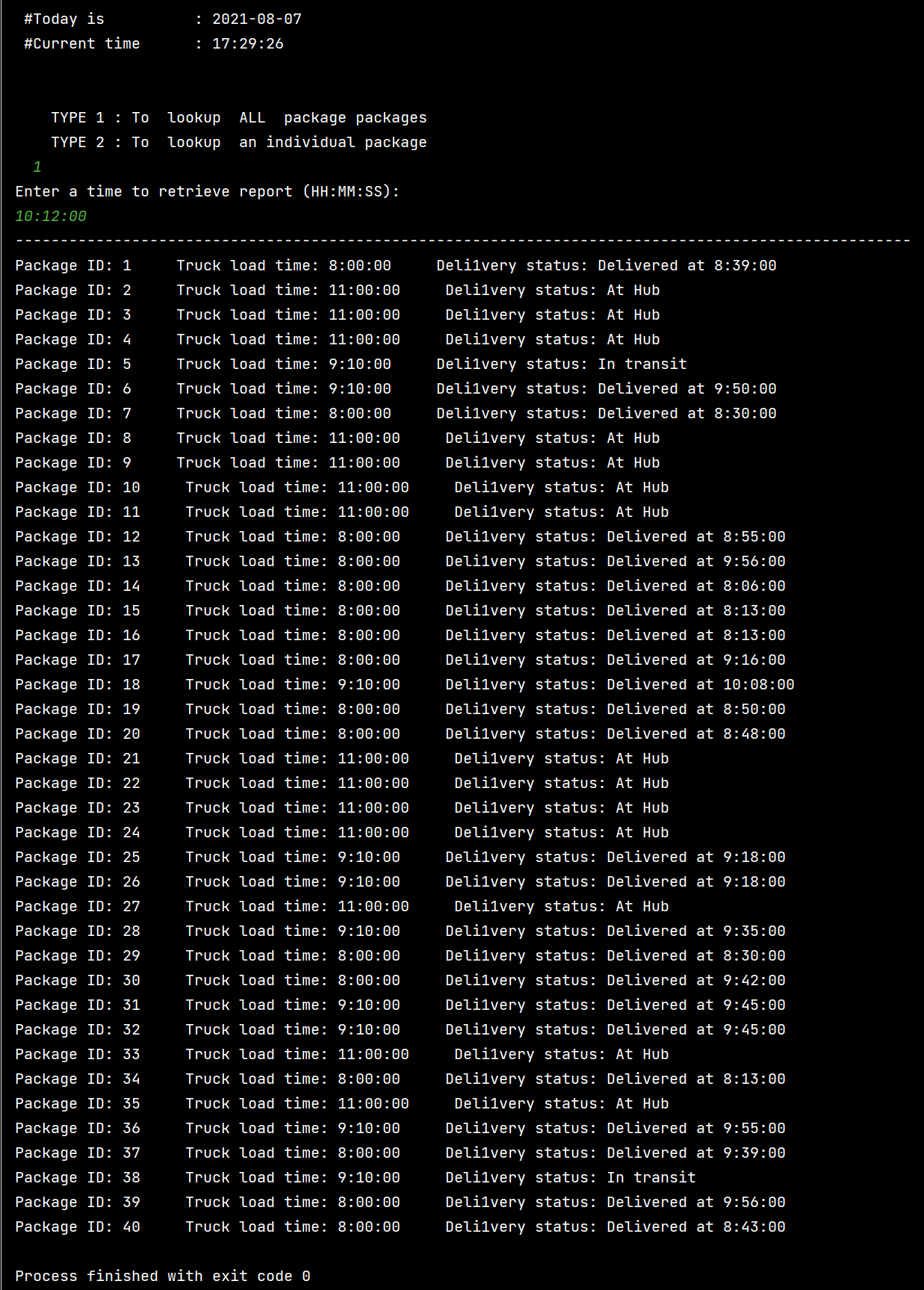
The main data structure used is a chaining hash table. It can perform the insertion and deletion as fast as direct hashing. While the searching is comparatively slow with direct hashing, it ensures no hash collision and storage efficiency. It is an array of linked lists, it can expand automatically as needed. The “Get\_fastest\_route” algorithm uses the unique package id as a key for the hash table. The hash table has four basic functions: insert, update, delete and search. In the first stage of the algorithm, the insert inserts all 40 packets in the hash table by packet id as a key. The packet is used in stages 2 and 3 for searching and sorting those packets further.

The only downside of chaining hashing is slow searching, which can be eliminated by increasing the size of the hash as the packet numbers grow.

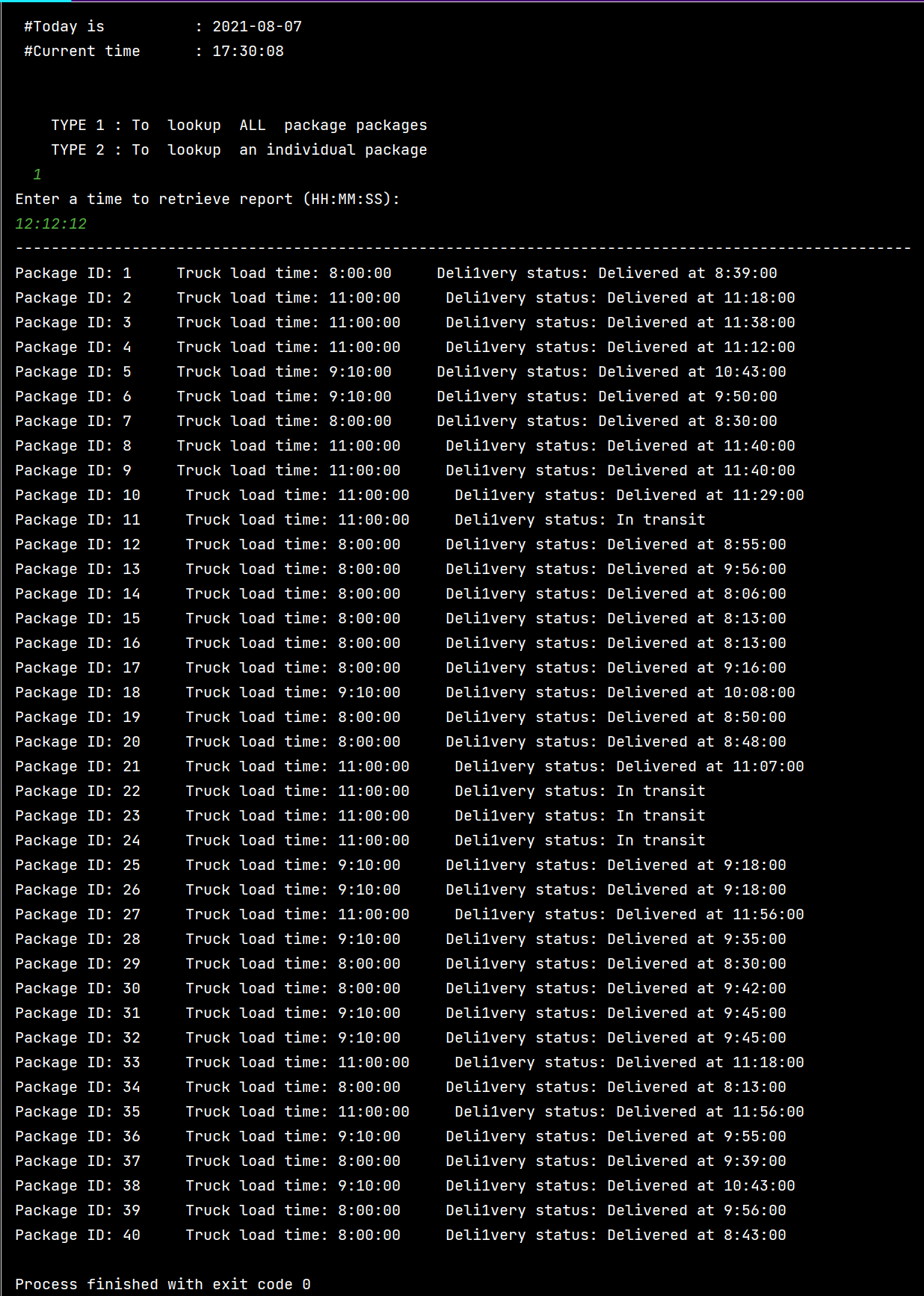
# G1. First Status Check

Time of report: 09:12:00

G2. Second Status Check

Time of report: 10:12:00

# G3. Third Status Check

Time of report: 12:12:00

# H. Screenshots of Code Execution

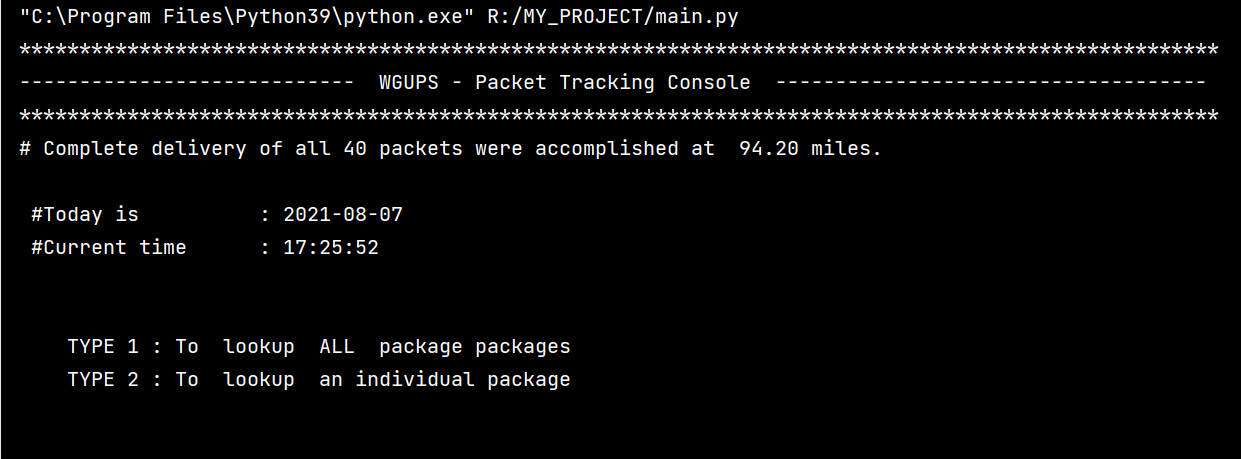


Image H1: Welcome screen: asking the user to type 1 or 2

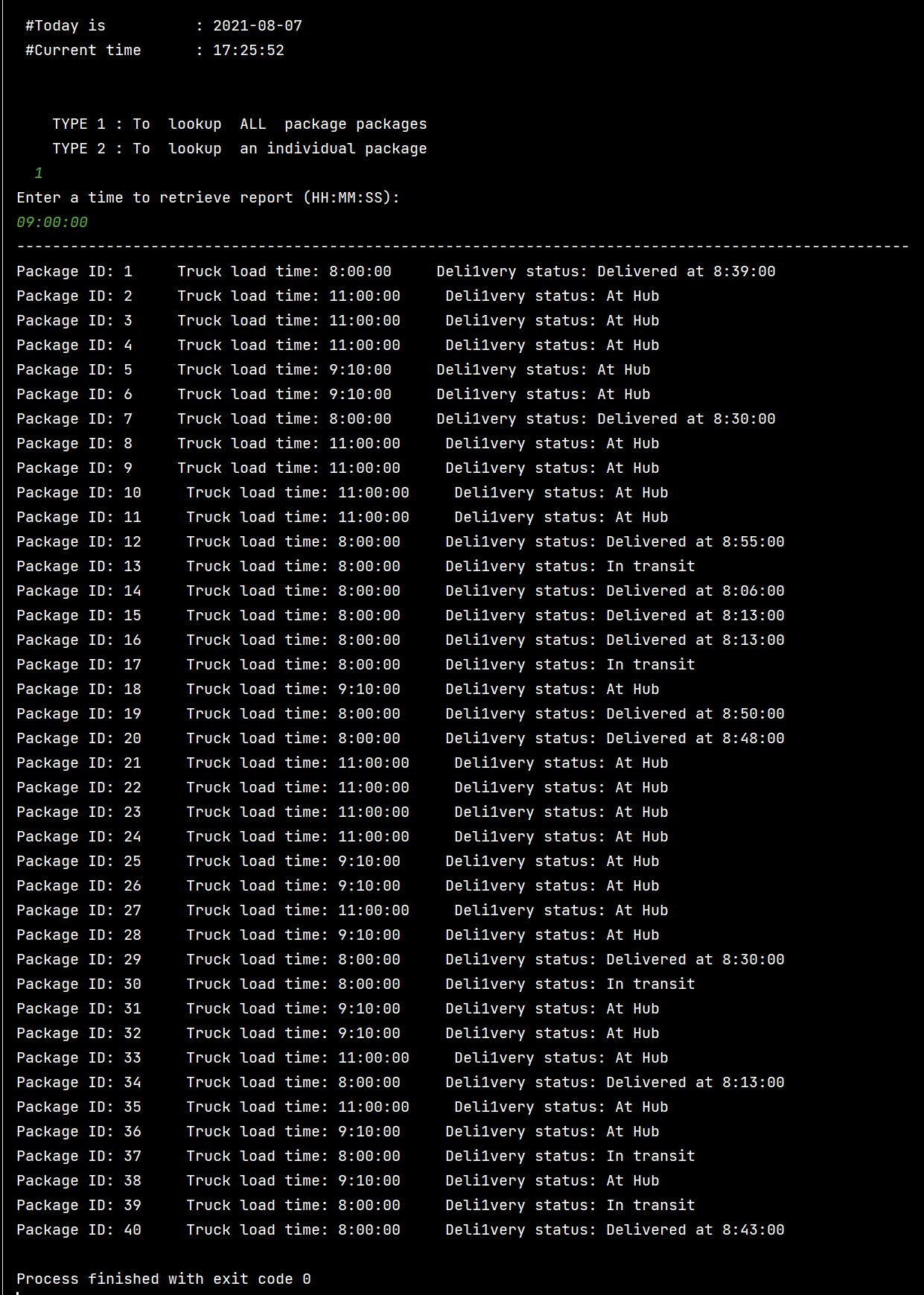


Image H2: user retrieves the report for all packages at 09:00:00

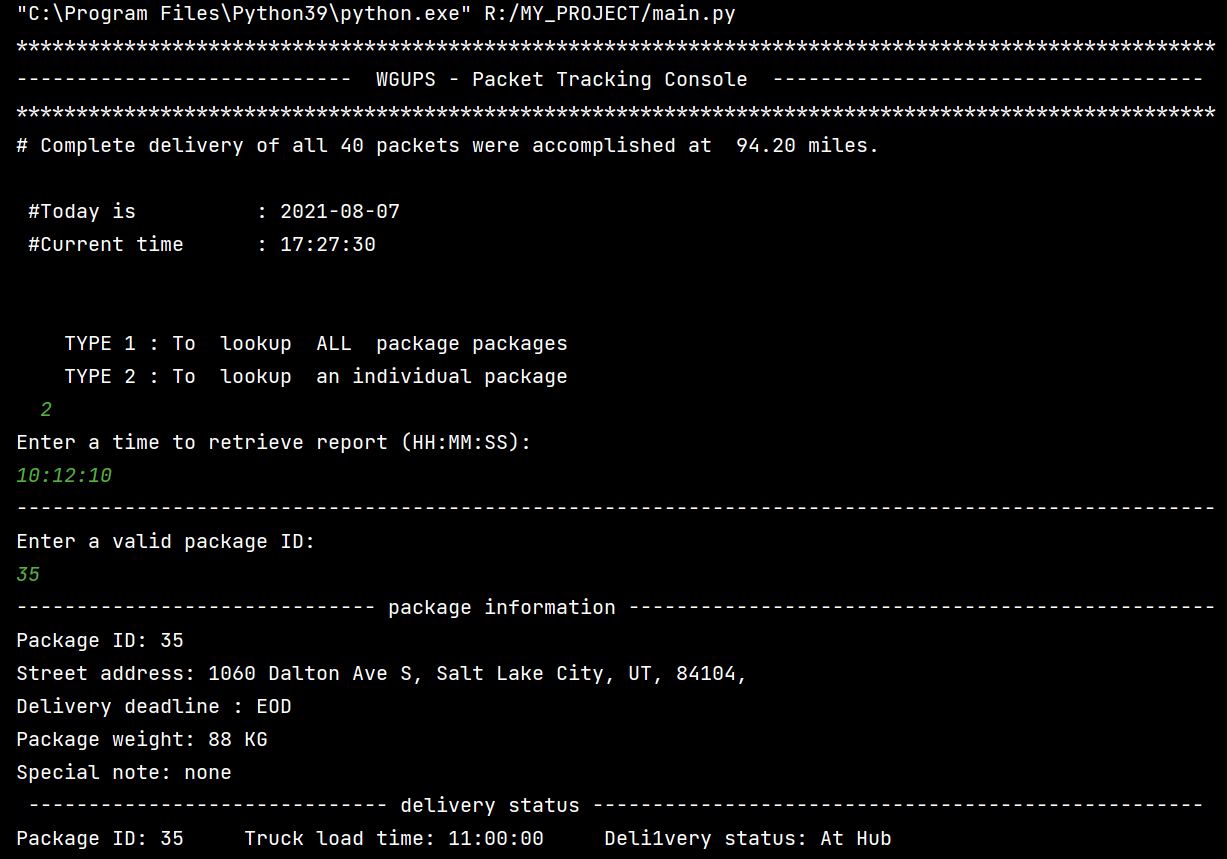


Image H3: user retrieves the report of package id 35 at 10:12:10

# 11. Strengths of Chosen Algorithm

Accuracy:

The algorithms meet all the needs and requirements specified. The algorithm embraces the “less than the total of 140 miles” requirement by meeting the milestone at just 94.20 miles. It tracks each of the movements of packets until the last packet reaches the destination with outstanding accuracy.

Flexibility

The algorithm is user-friendly at its finest. It is very simple but powerful enough to adapt to any changes. It is tailored to consume minimum memory space. Increase in the number of packets, trucks will be handled smoothly. For any changes in packets information, the algorithm will reassign the packet to the appropriate truck automatically.

# I3. OTHER POSSIBLE ALGORITHMS

The two other possible algorithms different from the one used in the solution, that also meet the requirements in the scenario are as follows:

1: Simulated Annealing (metaheuristic)

2. Genetic Algorithm (metaheuristic)

# I3A. Algorithm Differences

1: Simulated Annealing (metaheuristic): It is a simple optimization algorithm that compares the outputs of the objective functions running with the current and neighboring point in the domain iteratively. if the neighboring point generates an optimal result than the current one, then it is saved as a base solution for the next phase of iteration. Otherwise, the algorithm terminates the procedure without searching the wider domain for better results. (”SA algorithm”, n.d.).

2. Genetic Algorithm (metaheuristic): Genetic Algorithm uses techniques inspired from nature, more specifically evolution, to find an optimal or near-optimal solution towards a problem. It recalls evolution concepts such as reproduction and survival of the fittest to solve a real-time problem (Bani-Hani, 2020).

# J. Different Approach

if I would reattempt the project, I would possibly change the following:

**packet assignment:** as of now, the packets dispatching process needs manual labor to decide to select the right track. Since there are only 40 packets and 3 trucks it wasn’t challenging. A remarkable change in trucks and packages might bring the real challenge. on the next attempt, I would try to find a more accurate and automatic solution (a function or algorithm) to dispatch the packets. The solution would analyze the packets on the ground level. It will make different groups and subgroups of packets based on the matched patterns on them automatically.

**Linear probing (hash table)**: our current chaining hash table uses the linear probing search technique. As the number of linked list items grows the search time will be longer. On the next attempt, I would rather use quadratic probing to make searching faster and avoid the primary clustering of data.

# K1. Verification of Data Structure

The proposed chaining hash table implementation can meet all requirements. It delivered all the packages at a combined 94.20 miles total. All packets were delivered before deadlines. The Hash table was able to look up any data inside it efficiently. And the report was timely and accurate.

# K1 a. Efficiency

The chaining hash table implementation is quite efficient for the key-value pair data management process. The implemented hash table uses linear probing for its functions. The worst-case scenario space-time complexity for linear probing is O(n). The increase in the number of packages will affect the time needed to perform look-up functions linearly.

# K1 b. Overhead

The best thing about linear probing in chaining hash is that it is finest at space optimization. As the number of packages will increase it will just simply increase the items in the linked list. It gets slower we always can increase the size of keys of hash to make it faster.

# K1 c. Implications

The changes in the number of trucks will have very minimal or no effect on the lookup time-space use. The system does not store truck-specific information. trucks are implemented via lists that have no significant contribution toward time or space consumptions.

The increase in the number of cities also should have no significant effect on the lookup process and time-space complexities. The city information is only used to calculate and track the distance and time for each package.

The number of trucks and cities have minimal contribution toward look-up time-space usage compared with increased package’s number.

K2. Other Data Structures

The other 2 data structures that could also meet the requirements are as follows :

1. Weighted graph

2. Stack

# K2 a. Data Structure Differences

Graphs or nodes could be used to represent packages destinations. Implementation of the weighted graph can eliminate our hustle of calculating and storing the distances of each movement. The graph would likely have more time and space-efficient.

We could also implement a stack to simply pop the delivered packets from the truck’s assignment list. We could set the closest next destination as index 0 value for every iteration. The stack would be a more storage-efficient method than the current 1D list implementation.

# L. Sources

Bani-Hani, D. (2020). Genetic Algorithm (GA): A Simple and Intuitive Guide*.*

Retrieved from <https://towardsdatascience.com/genetic-algorithm-a-simple-and-intuitive-guide-51c04cc1f9ed>

Bowman, W. (2021). *Advantages and disadvantages of chaining - Delphi data structures*. Retrieved from <https://www.delphipower.xyz/data_structures/advantages_and_disadvantages_of_chaining.html>

Simulated Annealing Algorithm. (n.d.). Retrieved from https://www.sciencedirect.com/topics/engineering/simulated-annealing-algorithm