C950 WGUPS Algorithm Overview

Steven Evans

ID #000391474

WGU Email: [sevan48@wgu.edu](mailto:sevan48@wgu.edu)

Date

9/29/2023

C950 Data Structures and Algorithms II

* **A. Identify a named self-adjusting algorithm (e.g., nearest neighbor algorithm, greedy algorithm) that could be used to create your program to deliver the packages.**

For this program, I used a Greedy Algorithm to deliver the packages. I use the distance information given in the WGUPS Distance File to determine the shortest path from where the delivery truck is currently at to the addresses of the packages in the truck. The truck will go to the nearest address, drop off the packages (removing them from the list of packages in the truck), and then run the algorithm again to find the next nearest address from the truck's new position.

* **B. Identify a self-adjusting data structure, such as a hash table, that could be used with the algorithm identified in part A to store the package data.**
  + **1. Explain how your data structure accounts for the relationship between the data components you are storing.**

The chaining hash table was used for this project. The hash table allows for fast data storage and retrieval. The hash table uses a bucket to store items. The bucket is usually much larger than the expected number of items to avoid any kinds of problems to issues like collisions which helps avoid performance issues. The hash table uses insertion, deletion and search to add, remove, and find items in the bucket. Insertion adds to the bucket, deletion removes items, and search finds specific items in the bucket. For this project the hash table is used to update, remove, add, and search for packages. Packages are assigned an id, address, city, state, zip, weight, deadline. These attributes are used to add the items to the

* **C. Write an overview of your program in which you do the following:**
  + **1. Explain the algorithm’s logic using pseudocode.**

This code used the min() function in Python to determine the shortest distance between the truck's location and the next nearest stop.

def truck\_delivery(truck):   
 current\_time = truck.time\_left\_hub   
 current\_location = truck.current\_location   
#While loop for when truck has more than 0 packages.   
 while len(truck.packages) > 0:   
 min\_distance = 1000   
 min\_package = None   
 for id in truck.packages:   
 package = package\_hashtable.search(id) # Searches hashtable   
 distance = get\_distance\_two\_addresses(current\_location, package.address) # Distance between two addresses   
 if distance < min\_distance:   
 min\_distance = distance   
 min\_package = package   
   
 current\_time = current\_time + datetime.timedelta(hours= min\_distance/18) # Calculates the time for the truck traveling at 18 mph   
 min\_package.delivery\_time = current\_time   
 min\_package.status = "Delivered"   
 min\_package.left\_hub = truck.time\_left\_hub   
 truck.packages.remove(min\_package.id)   
 truck.miles += min\_distance   
 current\_location = min\_package.address

* + **2. Describe the programming environment you will use to create the Python application, including *both* the software and hardware you will use.**

PyCharm Version: 2021.3.3 Community Edition

Python Version: 3.10

Processor Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz 4.00 GHz

Installed RAM 16GB

Operating System: Windows 10 Home

The environment I used for this program was PyCharm Community Edition. I added several extensions to enhance the coding experience. The most useful extension I used was the GitHub extension. It allowed me to save my code online and access it from either my laptop or my desktop.

* + **3. Evaluate the space-time complexity of *each* major segment of the program and the entire program using big-O notation.**

This program has a Big-O time complexity of O(n^2), which is the same time complexity for the load\_distance algorithm. The time complexity is O(n^2) because both loop over a list that contains the distances and next calls itself. The delivery algorithm has the same time complexity of O(n^2) as well. Both functions use nested loops. The hash\_table and the function that prints all packages have a complexity of O(n) because they loop over the packages, while the functions in the hash\_table have a complexity of O(1).

* + **4. Explain the capability of your solution to scale and adapt to a growing number of packages.**

It allows the program to self-adjust to the number of packages. It uses a bucket list to add, remove, and search the bucket list. If a package is added to the list, it will be added at the end of the list. Search runs through the list until it finds the item being searched for and returns it.

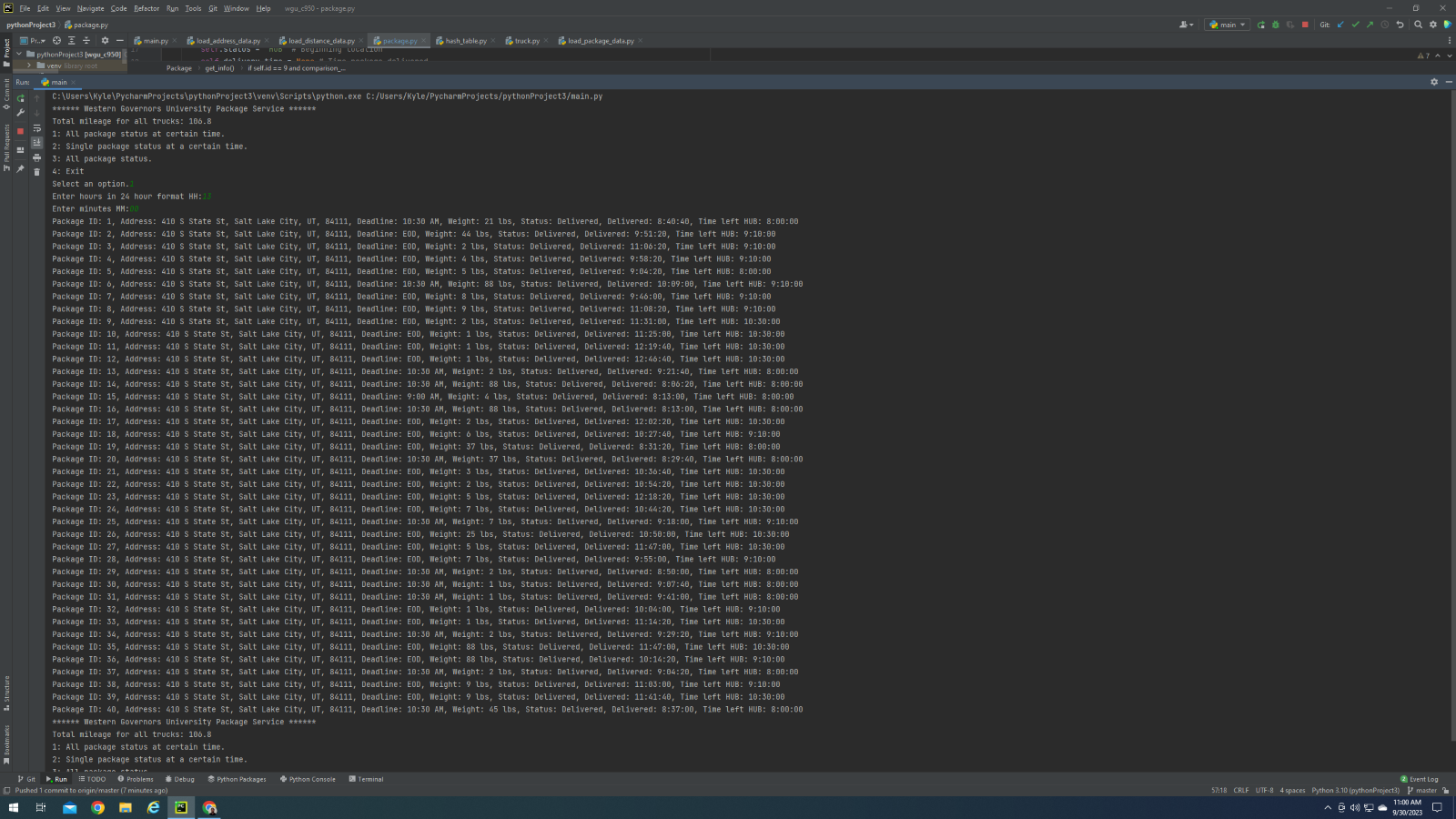
* + **5. Discuss why the software design would be efficient and easy to maintain.**

I tried to make this program as simple as possible, I used several online resources to find different ways to reduce the amount of code. I also worked with several different course instructors to help with problems I encountered along the way. Having less code makes the program easier to read and improve upon in the future, so it gives it a good baseline to work with. It also makes it cleaner and easier to read and understand.

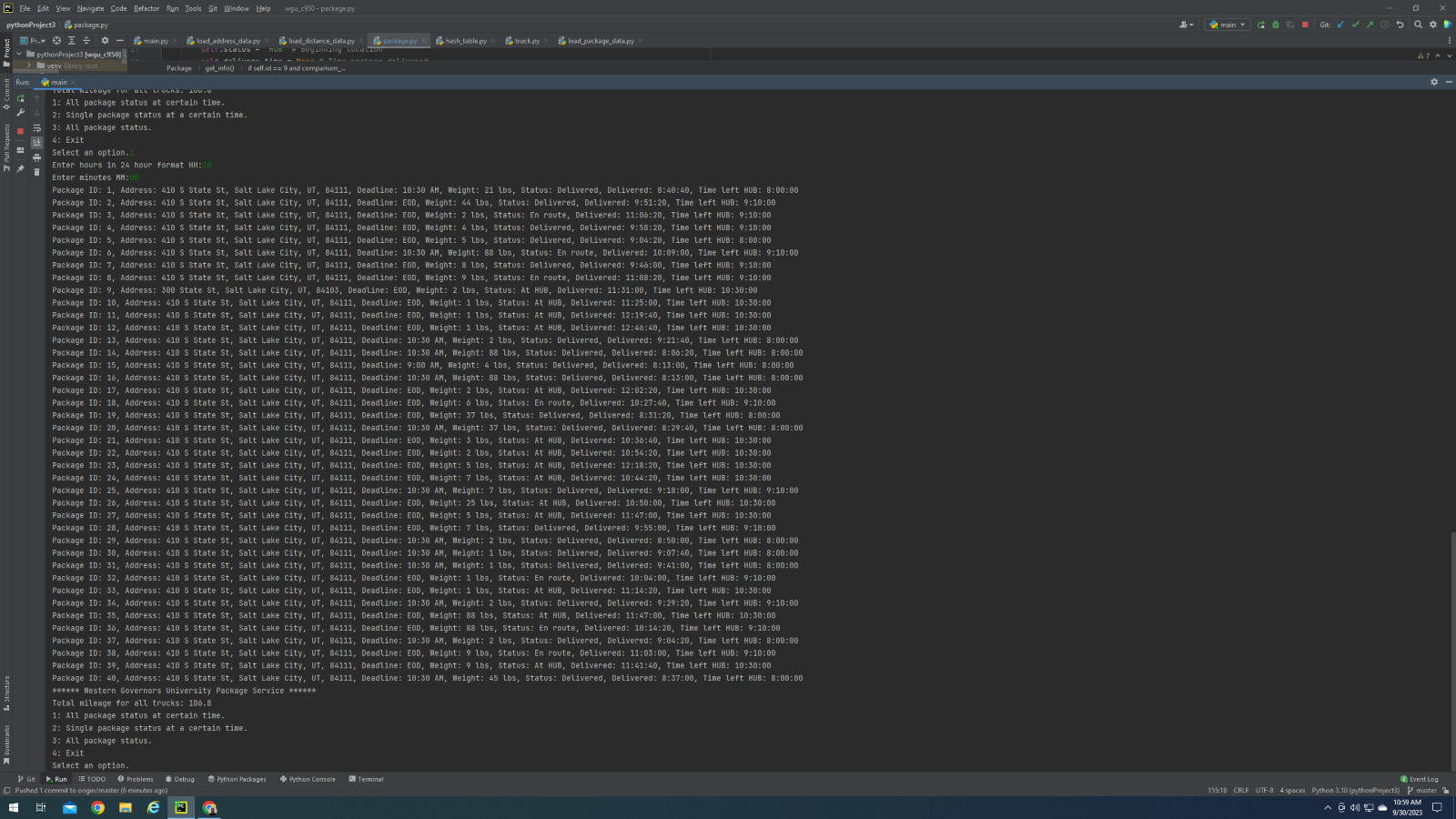
* + **6. Describe *both* the strengths and weaknesses of the self-adjusting data structure (e.g., the hash table).**

The hash\_table provides a data structure that can insert, remove, and update data, and it allows the data to be interacted with very fast. Hash tables enable very fast retrieval and storage of items based on a certain key value. The hash table uses buckets to store items, the buckets are only used if there is an item present, leaving the other buckets empty. This is one downside of using hash tables, it causes wasted storage.

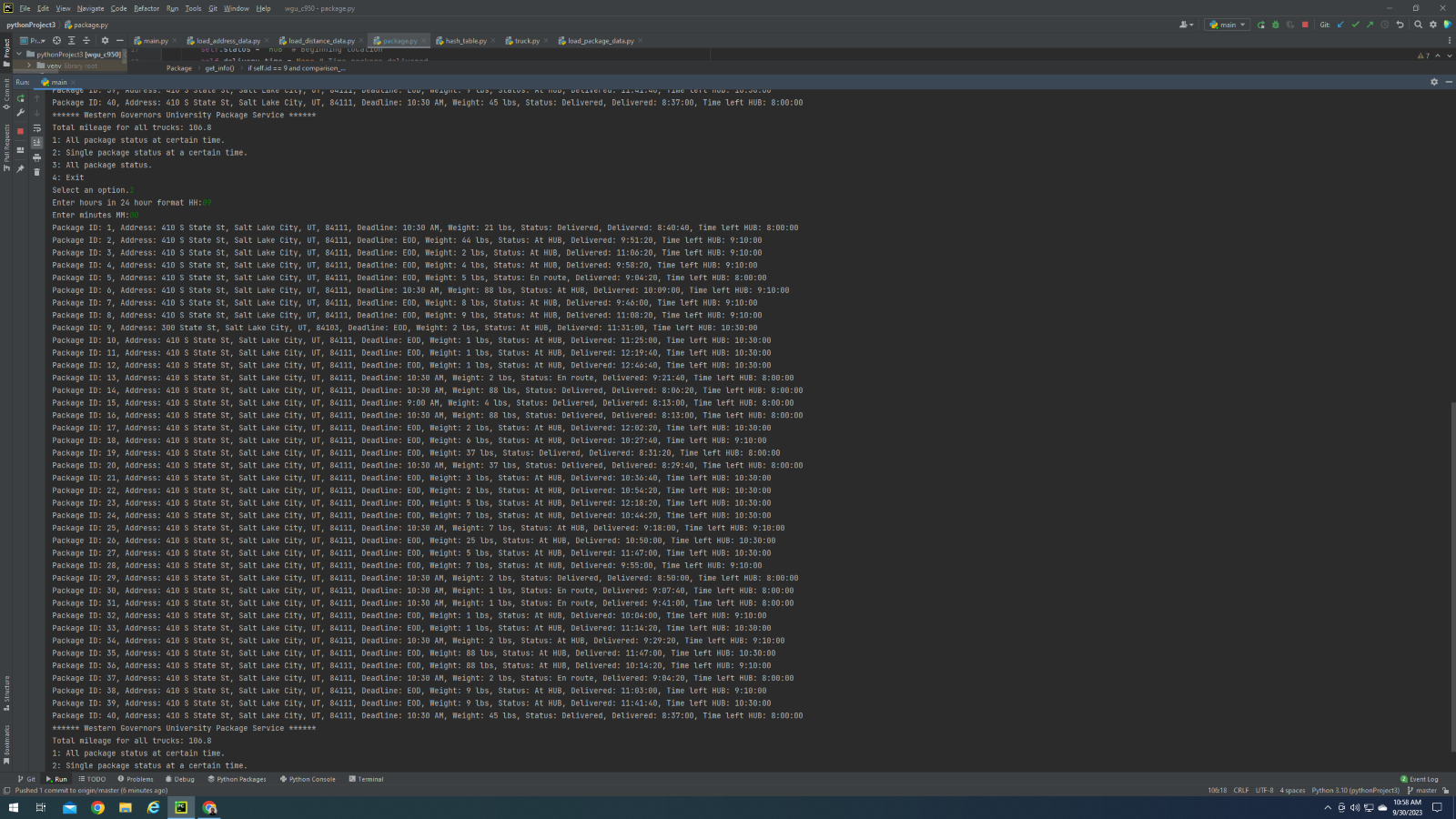
* + **7. Justify the choice of a key for efficient delivery management from the following components:**
* **delivery address:** The delivery address identifies the destination of the package. It helps with sorting the packages based on distances to come up with an efficient delivery route. This helps when creating the greedy algorithm, I used in the program. Delivery address would not be useful as the primary key due to several addresses being the same.
* **delivery deadline:** The deadline plays a large role in how the trucks are loaded. In my program I loaded the trucks manually to account for the deadlines of certain packages. Some trucks left later than others, I put packages that did not have a deadline on the EOD on the last truck leaving. The deadline would not be useful for the PK, many trucks share the same deadline.
* **delivery city:** The city is useful for grouping packages based on their destination city. If several packages are going to the same location, they should be loaded onto the same truck. It would not make sense to have another truck carry the package with the same address one truck is already going to. City would not be helpful for the PK, many packages share the same city.
* **delivery zip code:** The zip code is like city; it helps keep packages organized when loading them. As with delivery city, it would not make sense to load packages with the same zip code onto multiple trucks. Zip would not be helpful for the PK, many packages share the same zip.
* **package ID:** The package ID is the best choice for the primary key. It is unique to each package. It does not help much with the actual package destinations and delivery times. Delivery address, city, state, and zip is the best for that situation.
* **package weight:** Package weight can be useful for the actual handling of the package, or size of the package, and which truck it should be loaded on. This was not a requirement of this assignment, so it was not considered. Weight would not be helpful for the PK, many packages share the same city.
* **delivery status (i.e., at the hub, en route, or delivered), including the delivery time:** Delivery status is a must for this program. This was a requirement of the assignment. The user interface has 4 options on my program: get status of single package at certain time, get status of all packages at a certain time, get status of all packages, and exit. When the user selects an option and inputs a time, the interface returns the status of that package at the specified time. Delivery status would not be useful as the primary key because many packages will have the same status at any given time. For example, at 0800 all packages will be at the hub.
  + Screenshot at 1300:



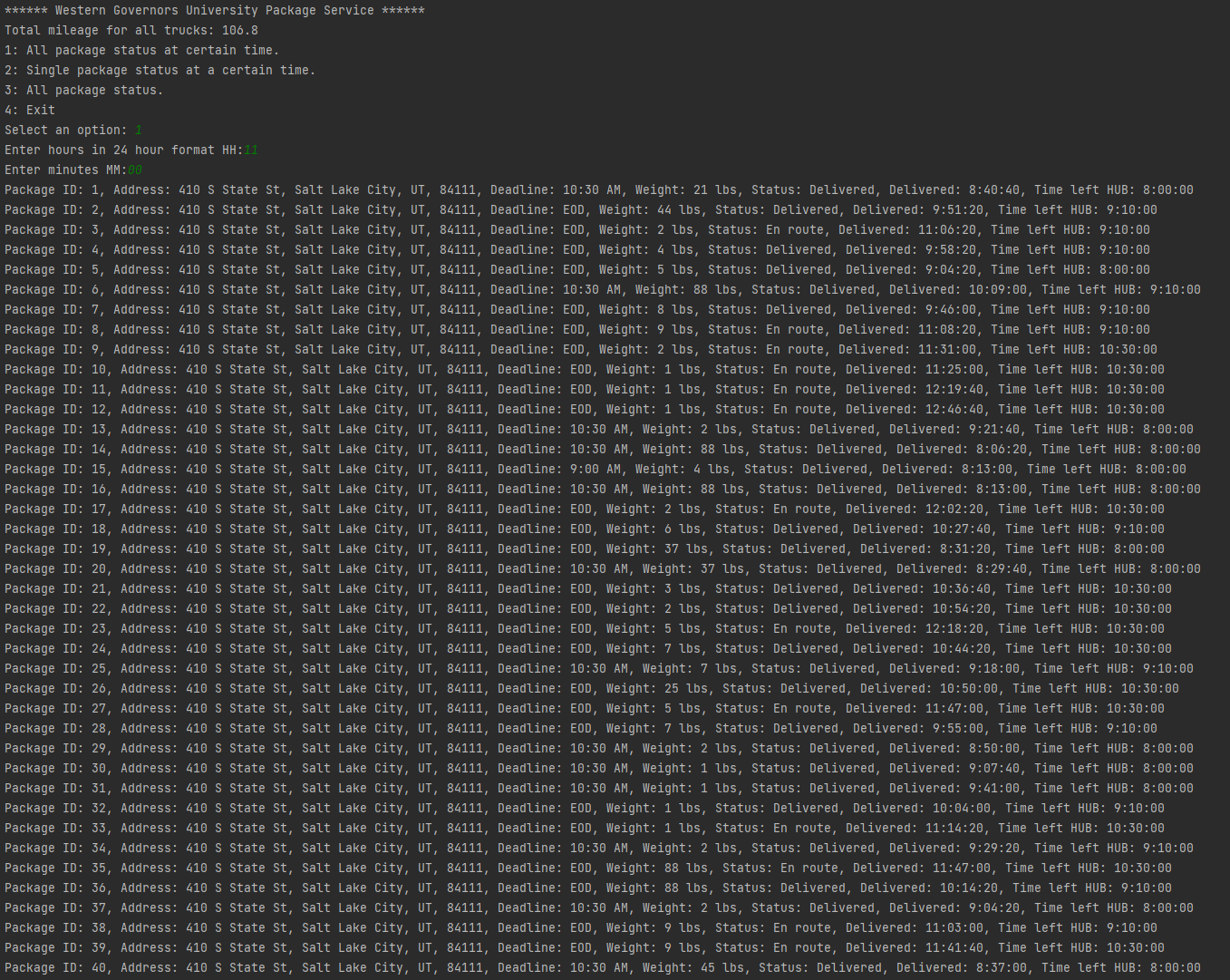
* Screenshot at 1000:



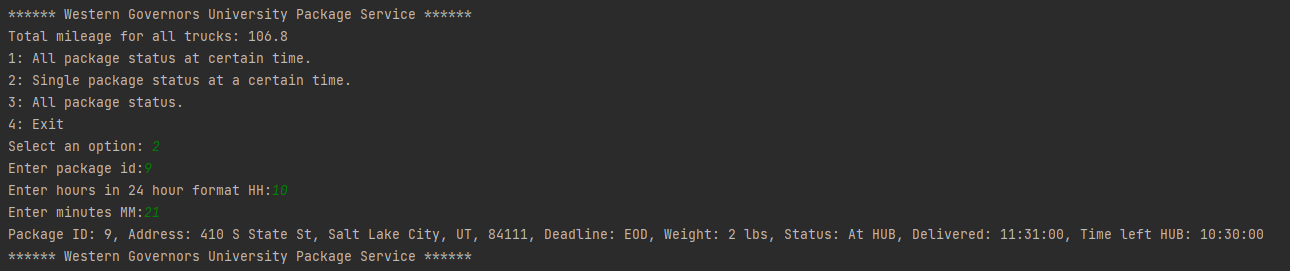
* Screenshot at 0900:



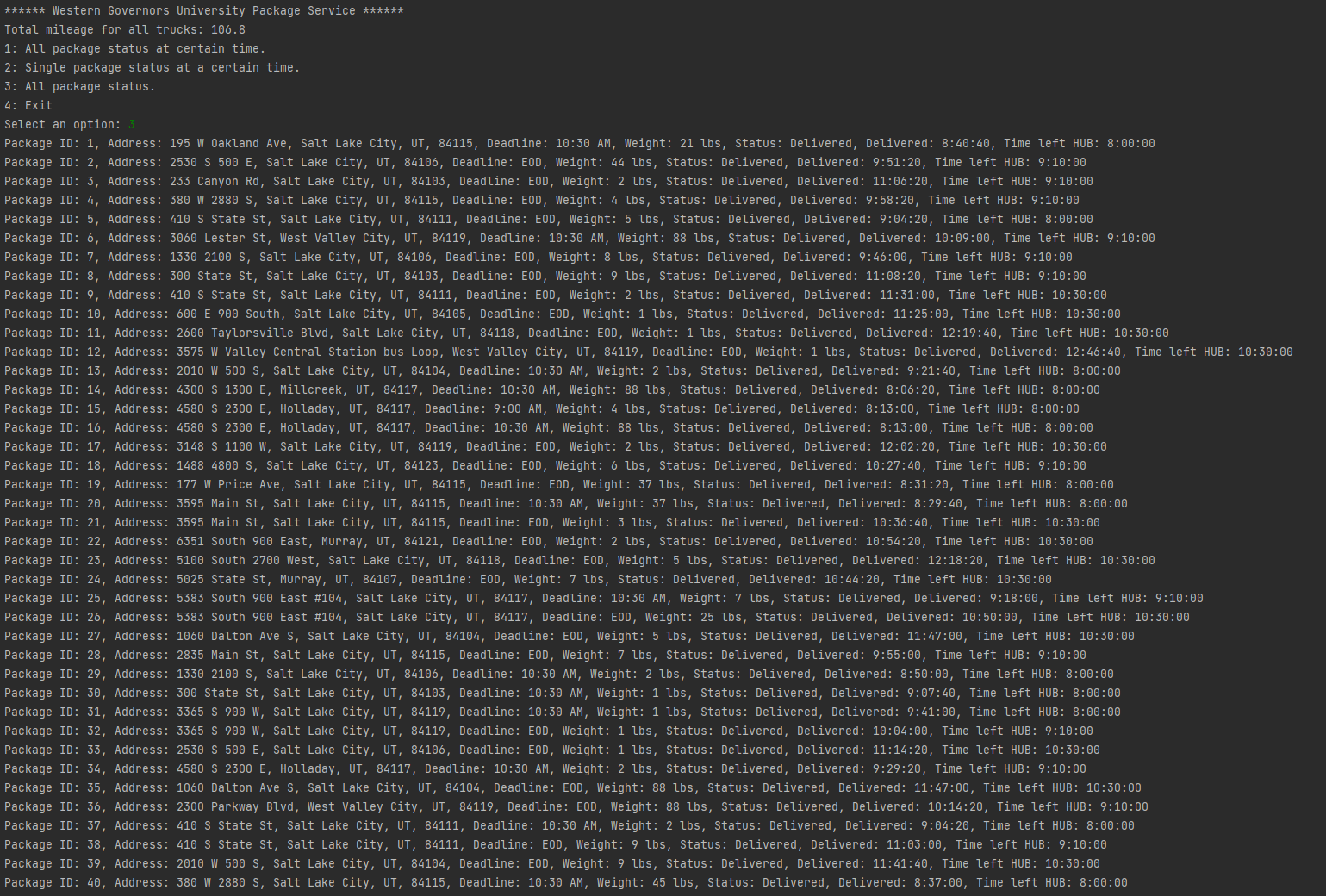
* Option 1 from user interface:



* Option 2 from user interface:



* Option 3 from user interface:



* Option 4 from user interface:

