WGUPS Routing Program Documentation

Caleb N. Lugo

Western Governors University

Algorithm Identification

For the WGUPS routing program, a nearest neighbor algorithm was implemented. The nearest neighbor algorithm finds a package that has the closest delivery point relative to the trucks current position. Once the shortest route has been determined, the truck will deliver the package and remove it from the list of items. The algorithm will repeat until there are no more packages to be delivered. Once the truck is empty, it will return to WGUPS.

Logic Comments

The main algorithm pseudocode is placed underneath this text. Additional comments have been left in the python files to help explain and clarify the program logic.

Create an item list

For item in trucks storage

Put items into the item list

While length of item list is not zero:

For items in item list

Route initialized to current potential route

If route is greater than current best route

Pass

Else if route is less than current best route

Current best route is now equal to route

If current best route is less than best route

Best route is equal to current best route

The trucks’ current location is now set to package location

Increment time

Update the package status

Remove package from list

Development Environment

The WGUPS routing program was developed with python on the latest version of the Windows 10 operating system, along-side PyCharm as the IDE. As for the hardware, a personal computer was used to run the program.

Space Time and Big-O

Each function in the program has a comment stating the Big-O time complexity. The overall time complexity of the main program is O(N^3).

Scalability and Adaptability

The algorithm takes a truck object and accesses the packages from a list the algorithm creates via a for loop upon startup. The for loop will loop through the entire truck’s storage, meaning that there is no real limit besides having at least one package in the list at minimum. Due to the requirements of the project itself, the trucks have a maximum storage capacity of 16 packages. If the trucks were to have increased or decreased capacity, the algorithm would not be affected outside of marginal performance shifts.

Software Efficiency and Maintainability

The program runs in polynomial time, meaning that unless a ludicrous amount of data was inserted the run time shall not be noticeably affected. With multiple comments explaining the logic and flow of the program, other programmers should not have much trouble modifying and maintaining the program.

Self-Adjusting Data Structures

The main data structure of this program is a hash table. The hash table has an array space of 10 cells, which can easily be further increased or decreased if necessary. The nearest neighbor algorithm is also a self-adjusting data structure, as it will scale appropriately according to the number of items inside of a truck object.

Original Code

All the code has been written by me. Any sources used are cited in the sources section at the bottom of the document.

Identification Information

An identifying comment has been placed at the top of main.py, which includes my full name and student ID.

Process and Flow Comments

Multiple comments have been added into the python files which help explain and guide any readers on the logic of the program.

Data Structure

Packages are stored into a list, which is stored into another list which resembles a hash table without using a built-in hash function from python. A hash table provides incredibly fast utility as it runs at a time of O (1). (Joe James.)

Explanation of Data Structure

Package data is stored in a list, which is the inserted into the hash table via another list. Each piece of data is identified with a key. For the case of package information, the package ID shall be the key, and will be used to search for the package within the table.

Hash Table

The hash table functions correctly and has an insert function to add new data.

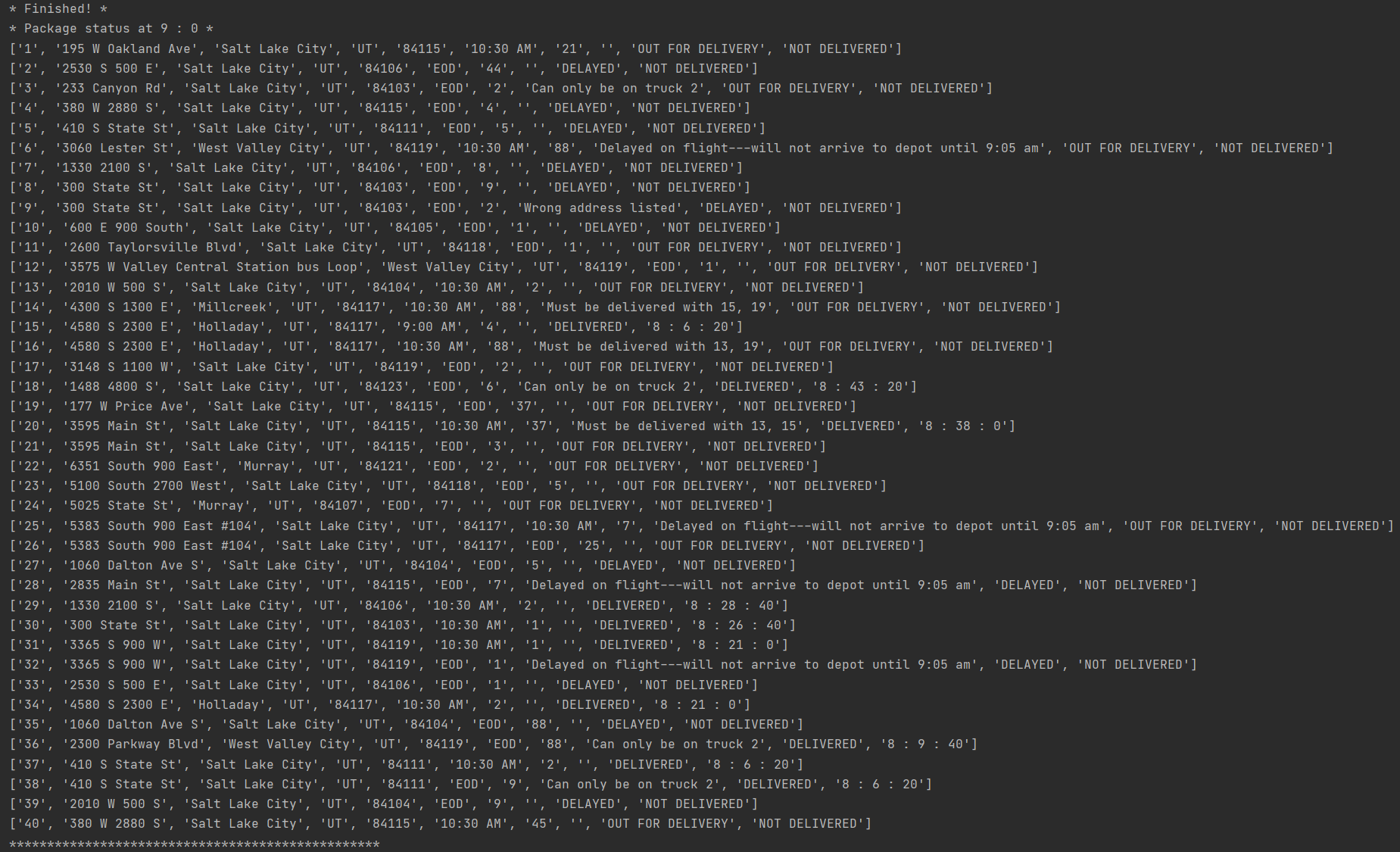
Lookup Function

The data in the hash table is accessed via a search function. Since the hash table is meant to contain packages, the package class has a couple of functions that retrieve data from the hash table’s search function. Packages are searched by their ID.

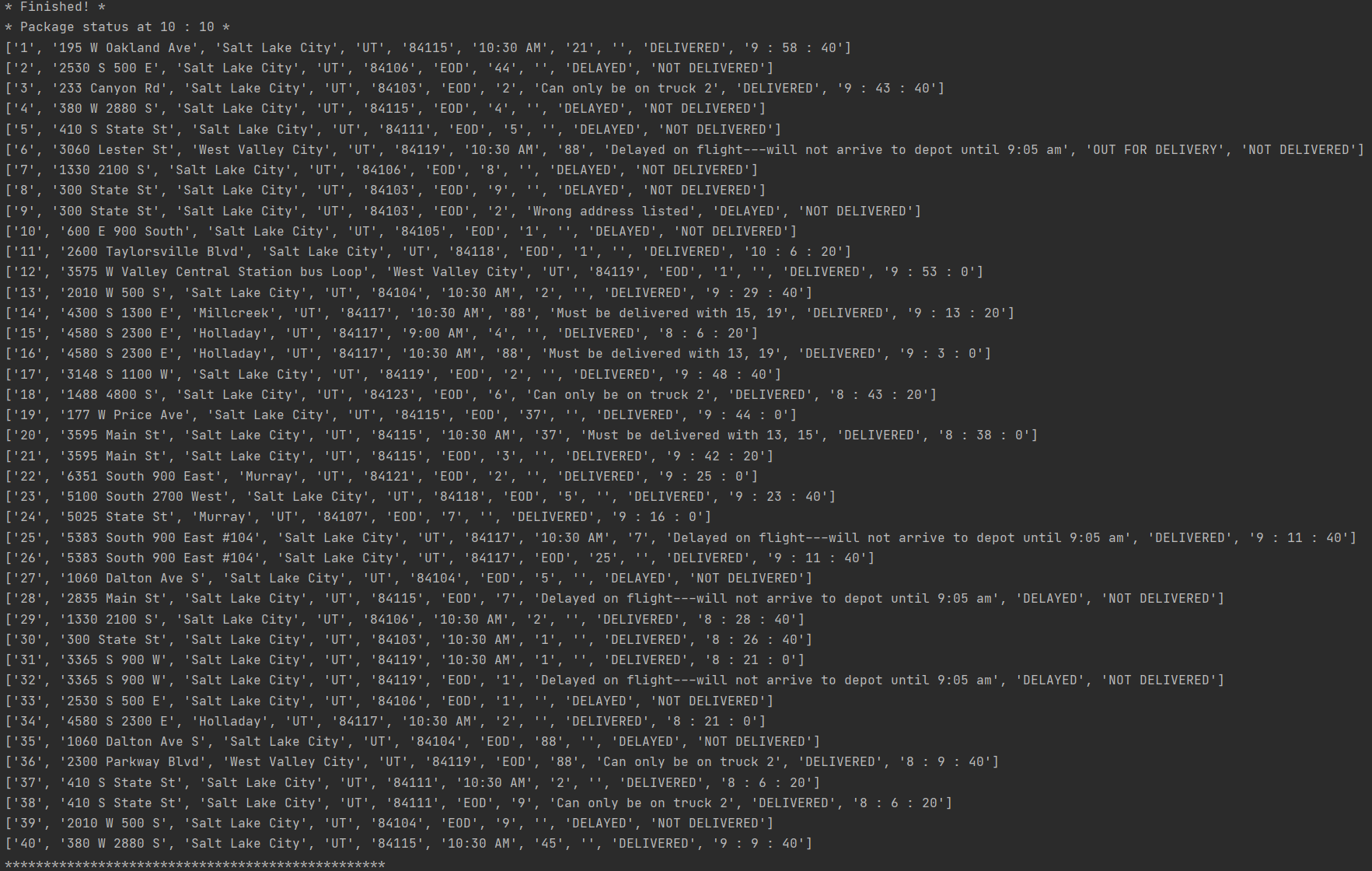
Interface

Upon starting the program, the user will be presented with a menu that offers five options. These options being: Deliver all packages, view package status, check truck mileage and return time, check delivery at a specific time, and to close the program.

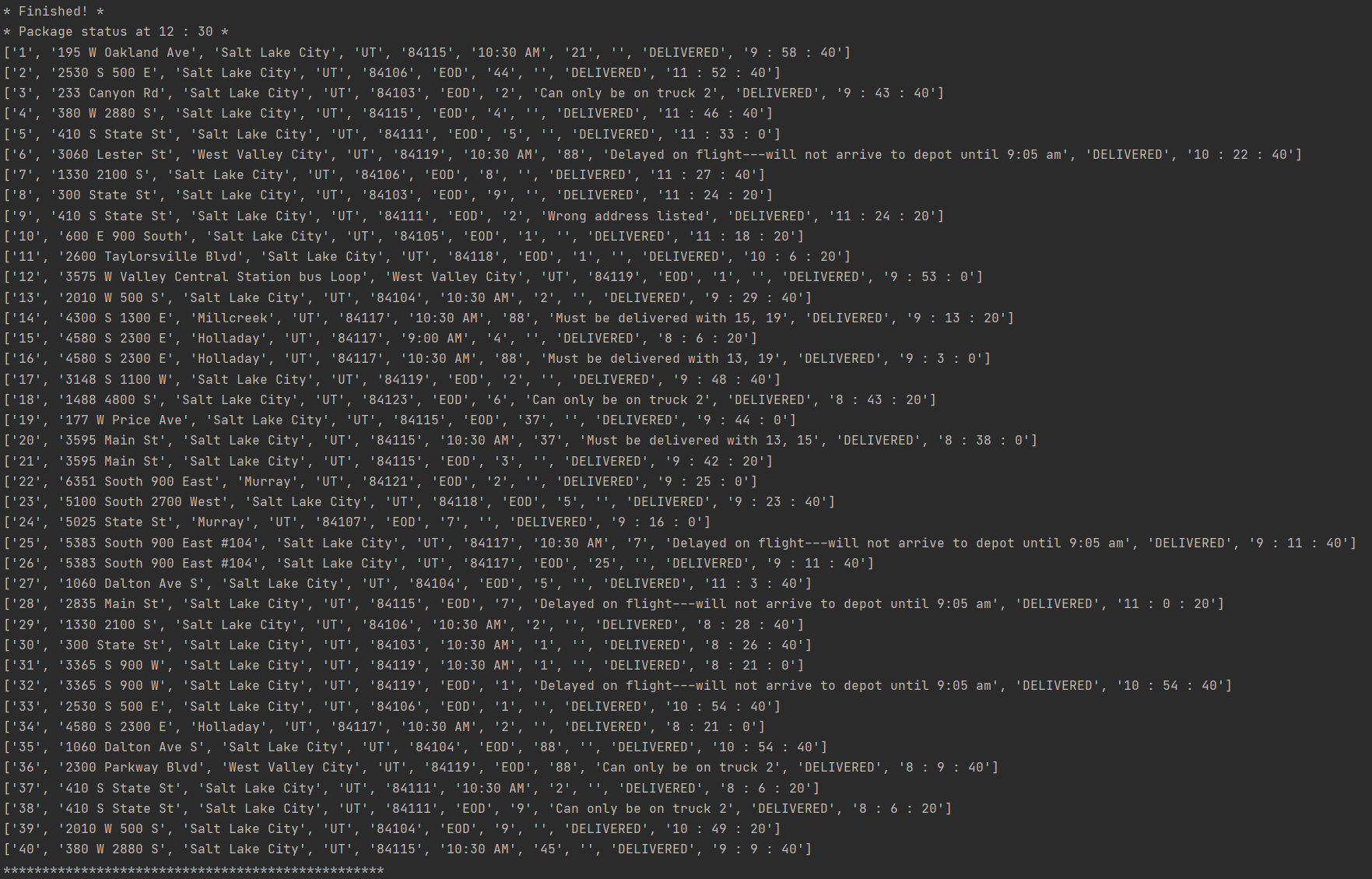
First Status Check



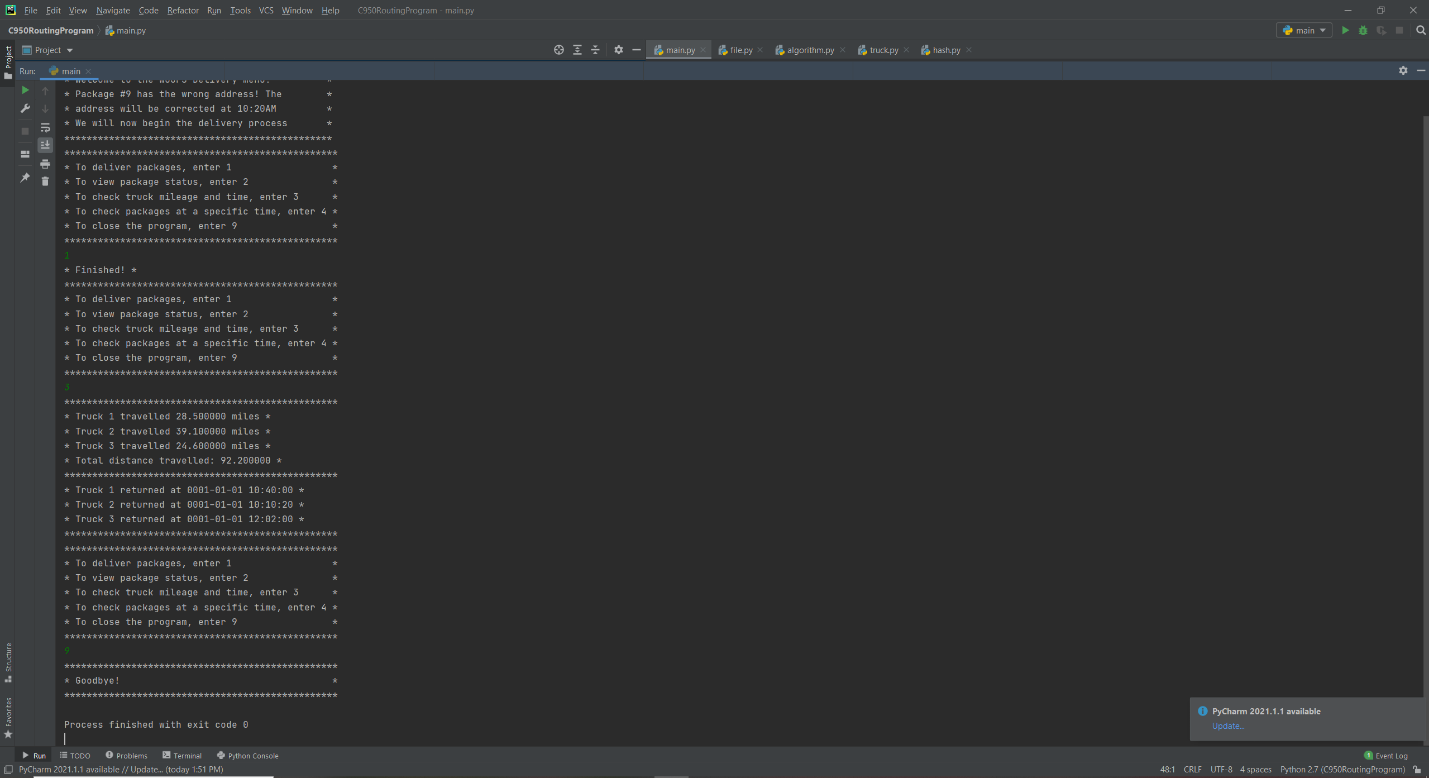
Second Status Check



Third Status Check



Screenshots of Code Execution



Strengths of The Chosen Algorithm

The nearest neighbor algorithm will execute quickly (specifically, N^2 execution time). It will also scale with the size of the truck, meaning that it can handle much larger deliveries if desired.

Verification of Algorithm

As seen in the user interface, all packages are delivered on time, meeting deadlines when presented. Also, the trucks travel a total distance of 92.2 miles.

Other Possible Algorithms

Two alternative algorithms that would be acceptable for use for this program are the 2-opt and the multi-fragment algorithm.

Algorithm Differences

The 2-opt algorithm would further optimize the program by checking the entire route and optimizing the paths to be shorter, rather than the greedy nature of the nearest neighbor algorithm not considering its final route. The multi-fragment algorithm would find the closest route between multiple stops, eventually making a full route, but it has a runtime of N^2 Log N, making it slower than the nearest neighbor algorithm.

Different Approach

If given the opportunity, the 2D array that stores the location data would be replaced with a weighted graph. A weighted graph would allow for a cleaner implementation of the distance data for the algorithm, rather than having multiple functions to access the data when needed.

Verification of Data Structure

All requirements have been met, it can be verified either by the user interface or within the code itself.

Efficiency

If an insane number of packages were stored into the hash table, it can cause a noticeable decrease in performance. For the requirements of this project, the hash table and its functions run flawlessly and respond in a timely manner.

Overhead

The current algorithm and hash table should be able to handle any reasonable number of packages. If thousands of packages were put into the program it may slow down but for the current objective, the program is efficient.

Implications

Any trucks would require the algorithm to run separately for it, at a time of O(N^2). As the time complexity will be the same, additional trucks should not pose a threat to performance. If any additional cities were added and were needed to be visited, the nearest neighbor algorithm would have to loop through each one for each package, potentially slowing down the program.

Other Data Structures

Instead of using a list, a dictionary could be used to store package information. Likewise, a binary tree could potentially store the package ID, and be used to pull up a package in a similar fashion to the hash table.

Data Structure Differences

A dictionary could store package information by using the package ID as a key and keeping the rest of the information in a list. While this can be viable, it would be much slower than the hash table. A binary tree could work in a similar way to the hash table by pulling up packages from their ID, but it would have a worse time complexity of O(N).

Sources

Joe James. (2016, January 22) *Python: Create a HASHMAP using Lists.* [Video]. YouTube. <https://www.youtube.com/watch?v=9HFbhPscPU0>

Professional Communication

This document is written professionally without any grammatical or spelling errors.