Algorithm Overview

James Laurie

Date 04/04/2023

C950

The aim of this assignment is to create an algorithm that will deliver the required packages while taking into consideration their specific constraints and staying under the 140-mile mileage limit..

A.

Nearest Neighbor is the greedy method I selected. After looping through a list of potential locations, this algorithm chooses the closest spot. The selected place has been designated as the reference position and has been removed from the list of possible locations. The algorithm will keep running until there are no more possible places.

B1.

The nearest neighbor algorithm is used to order packages on a specific load. Once the packages are sorted, this technique calculates the distance traveled by a particular load.

Clear the package list for the given load to allow the package to be reinserted into the load in the order # of the closest neighbor.

Continue going through the inventory of the still-needed list until there are none remaining, then add the nearest packet to the list.

```
def nearest_neighbor_delivery(load):
    still_needed = []
    for package_id in load.package:
        pack = packages_table.lookup_id(package_id)
        still_needed.append(pack)
    load.package.clear()


    while not len(still_needed) <= 0:
        next_local = 2000
        next_load = None
        for pack in still_needed:
                    if  distance(get_address(load.address),  get_address(pack.address))  >
next_local:
                continue
            next_local = distance(get_address(load.address), get_address(pack.address))
            next_load = pack


        load.package.append(next_load.id)


        still_needed.remove(next_load)
        load.mile += next_local
```

```
        load.address = next_load.address

        load.time += datetime.timedelta(hours=next_local / 18)

        next_load.delivery_time = load.time

        next_load.departure_time = load.depart


nearest_neighbor_delivery(first_truck)

nearest_neighbor_delivery(second_truck)

nearest_neighbor_delivery(third_truck)
```

B2. PyCharm Version: 17.0.6

Python Version: Python 3.10

Hardware: OMEN by HP Desktop PC 870-2XX running Windows 10 Home

Pycharm is the programming environment that I used for this application. I did not use any extensions to enhance my coding experience.

B3.

The program's total time complexity is O(n2). Because it loops over a list of distances and then uses itself recursively, the delivery method is O(n2).
The time complexity of the function that loads distance data into the packages, like the function that places the packages onto the truck, is O.(n2). Nested for loops are used in both of these methods.

Because it loops through all packages, the function that prints all package data is O(n). Because it only searches the hash table for a single package, the function that prints a single package to the UI is O(1).

The hash table has an O(n) space complexity, and each of its functions has an O(1) time complexity. (Lysecky, Section 2.6).

B4.

My solution would need to be updated to handle more packages. I followed the advice of one of my teachers and manually loaded the trucks, but this is not a scalable solution. I'd need to create a function that loads the trucks programmatically based on time constraints and notes. Given the variety of special cases, such as when a package is delayed in flight or can only be delivered on a specific truck, this would be challenging, but it would be absolutely essential for my application to be scalable.

B5.

This software has a solid structure to build on and is efficient when compared to similar applications that do not use a hash map. With a solid skeleton like this, the project will be simple to build on while staying organized as new features are added. The project structure can easily stay organized because utility functions are separated from the package and truck class files.

B6.

The primary advantage of the hash table is the speed with which its data can be interacted with. It is a significant benefit to have a data structure that can update, remove, and insert data in linear time. Another benefit is that it can gracefully handle collisions. These benefits combine to create an excellent data structure in terms of scalability. One disadvantage of this data structure is that if it is designed to be larger than required to handle future packages, it may waste space. Because the hash table was intended to contain thousands of items, most of its buckets are likely to be empty at times. Wasted storage is a small disadvantage in comparison to the benefits of speed.

D.

The hash table is a self-adjusting data structure that can be used with the algorithm discussed in Part A.

D1.

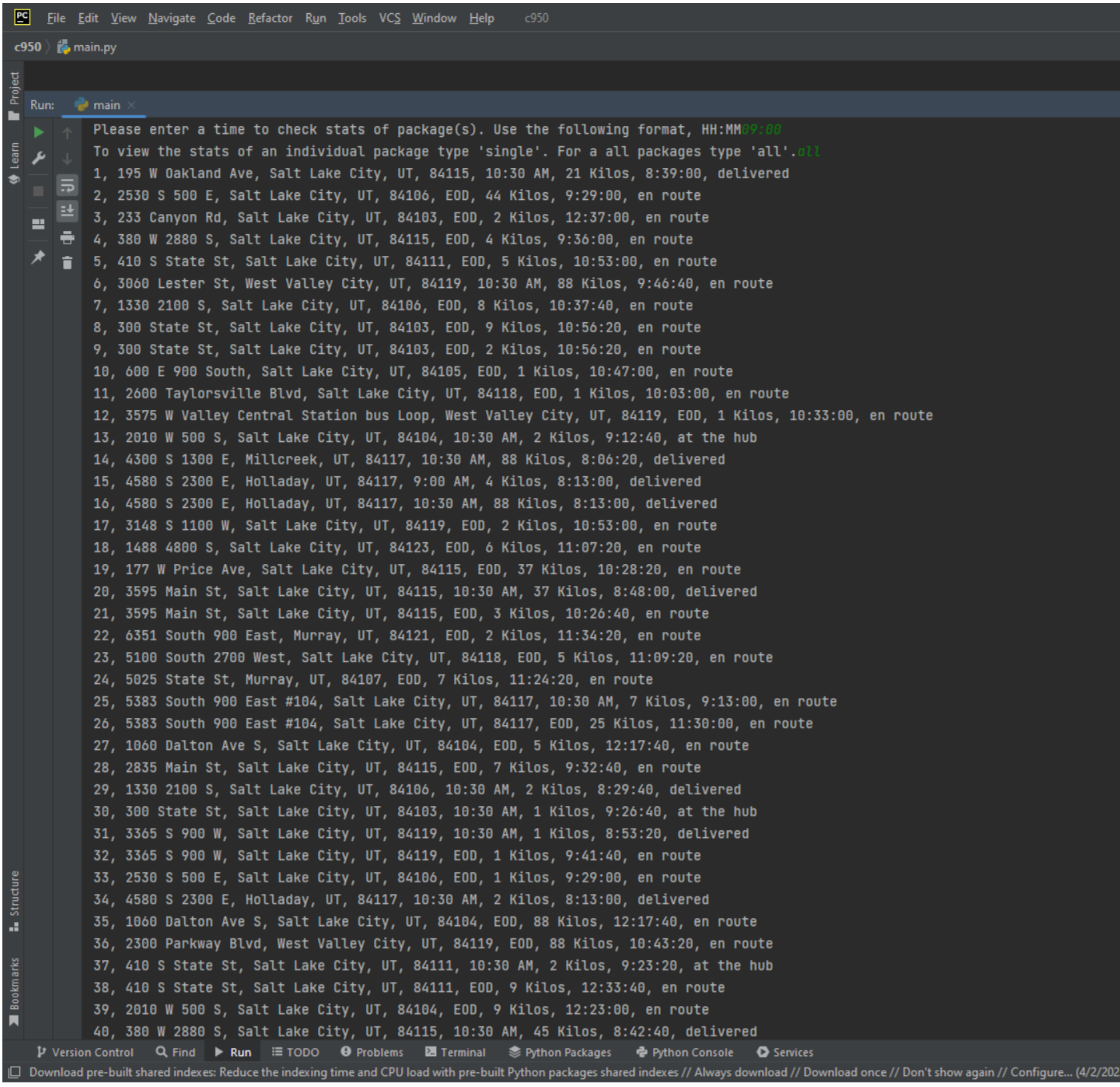
The structure of the hash table described in our Zybooks curriculum influenced the hash

table used in this project. (Lysecky, section 7). It implements an insertion function that uses a key-value pair to store and retrieve data. For this project, the hash table's initial size is forty. The key is one of the forty 'buckets,' and the value is the object that will be stored in that bucket. Use HT.insert to insert package p into hashtable HT, for example.(id, p). As a result, the unique package id indicates which bucket the item is stored in.

The hash table first uses the ID number to account for the relationship between the stored data points. This ID serves as the key for retrieving the entire package object being stored or one of its specific attributes, such as its destination or deadline.

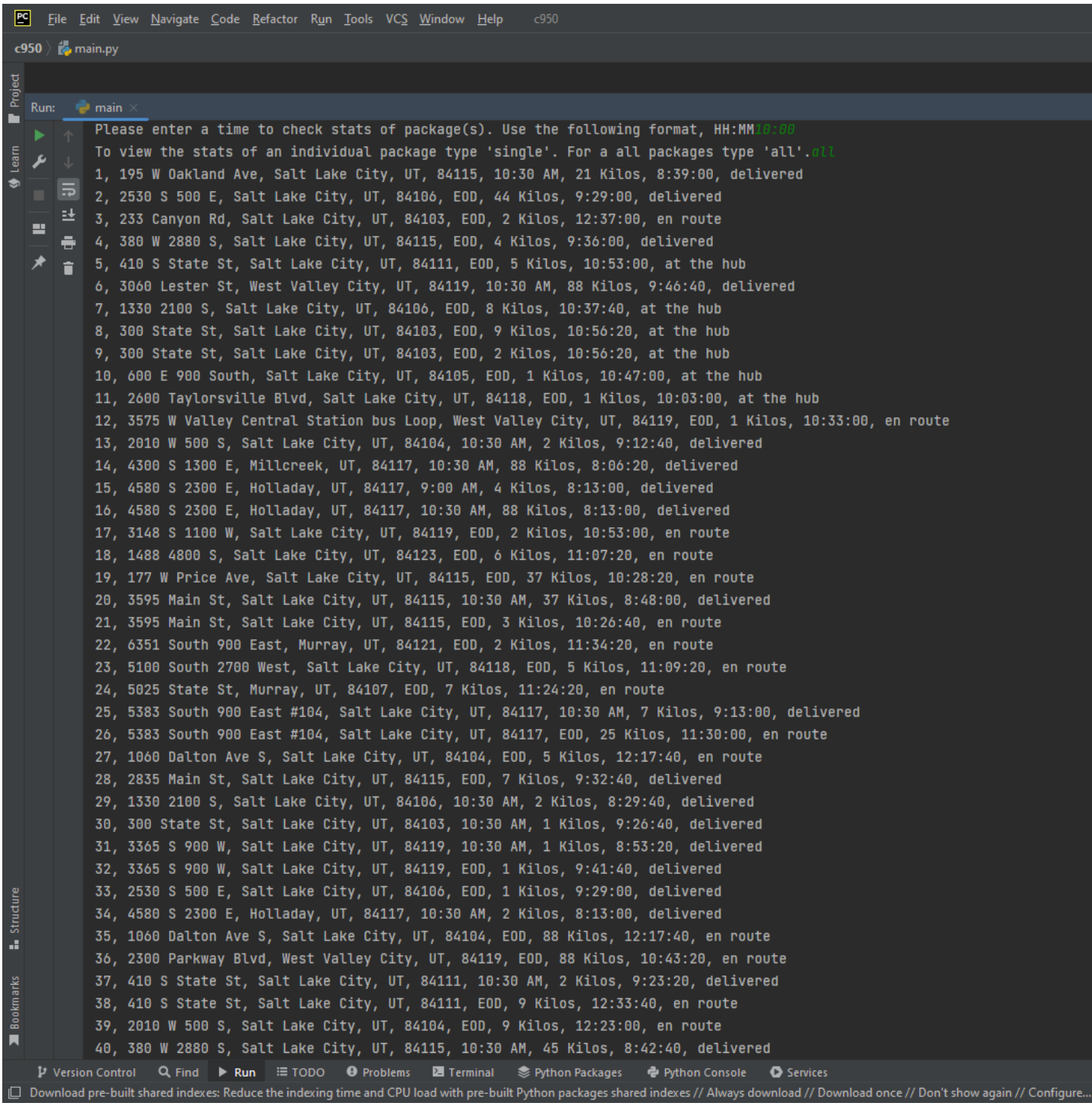
This model can access any piece of information about any package. If you wanted to see the destination of a package with ID 13, you would use the hash table search function (HT.search(13)) to retrieve the entire package. They now have complete access to the package object. They could then quickly obtain the destination by appending ".destination" to the end of the search function, such as ht.search(13).destination. This can be done for any of the package attributes in a similar way, so the hash table accounts for all relationships between the data points it is storing.

G1.
Selecting ‘all’ and searching for package statuses at 9:00.



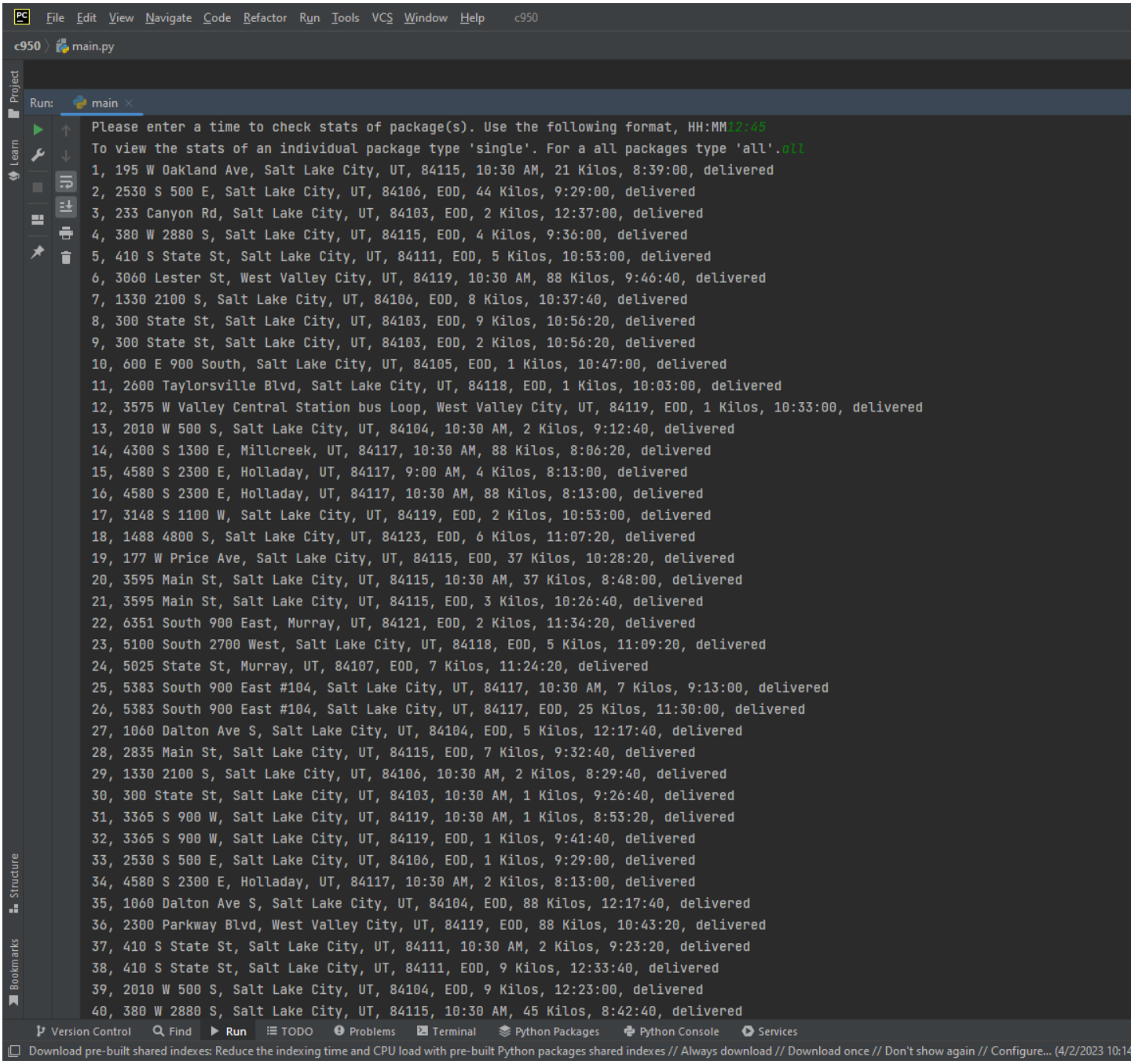
G2.

Selecting ‘all’ and searching for package statuses at 10:00.

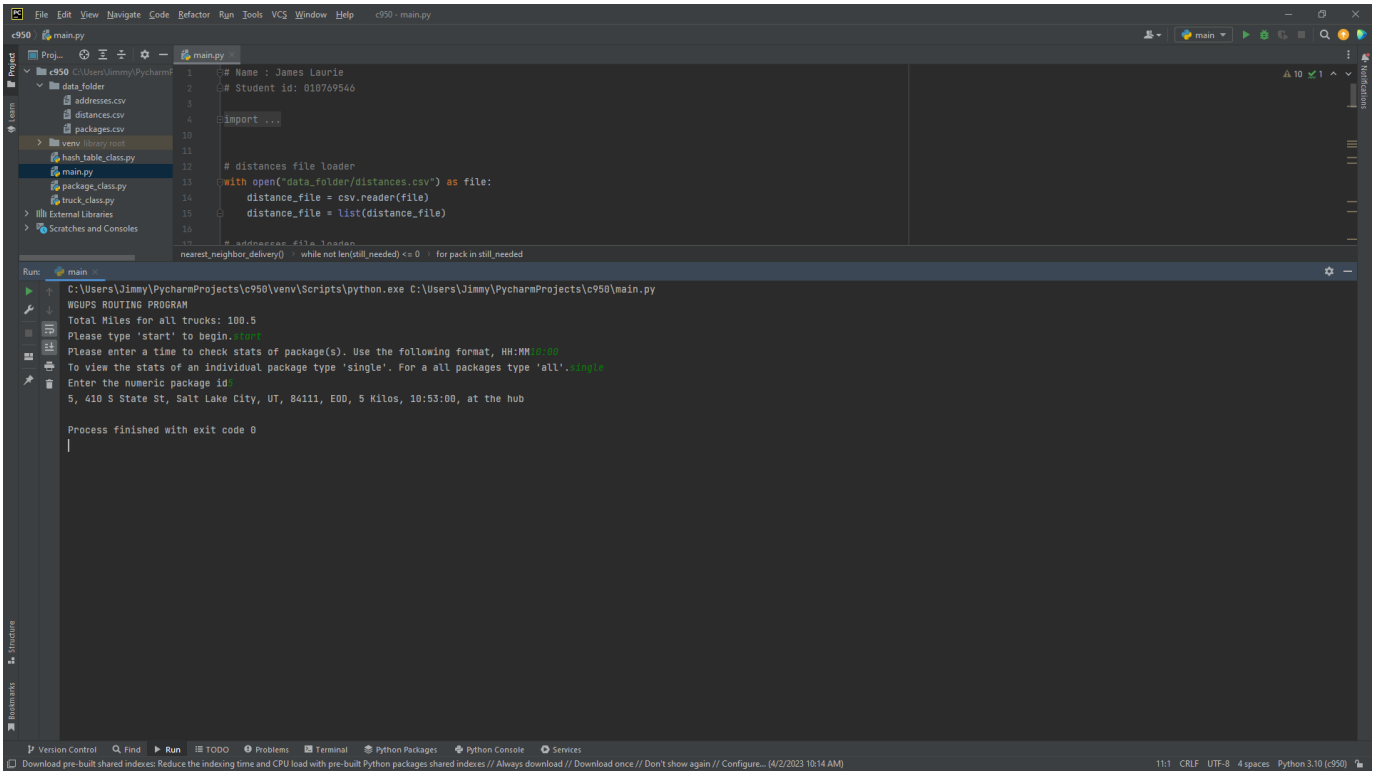


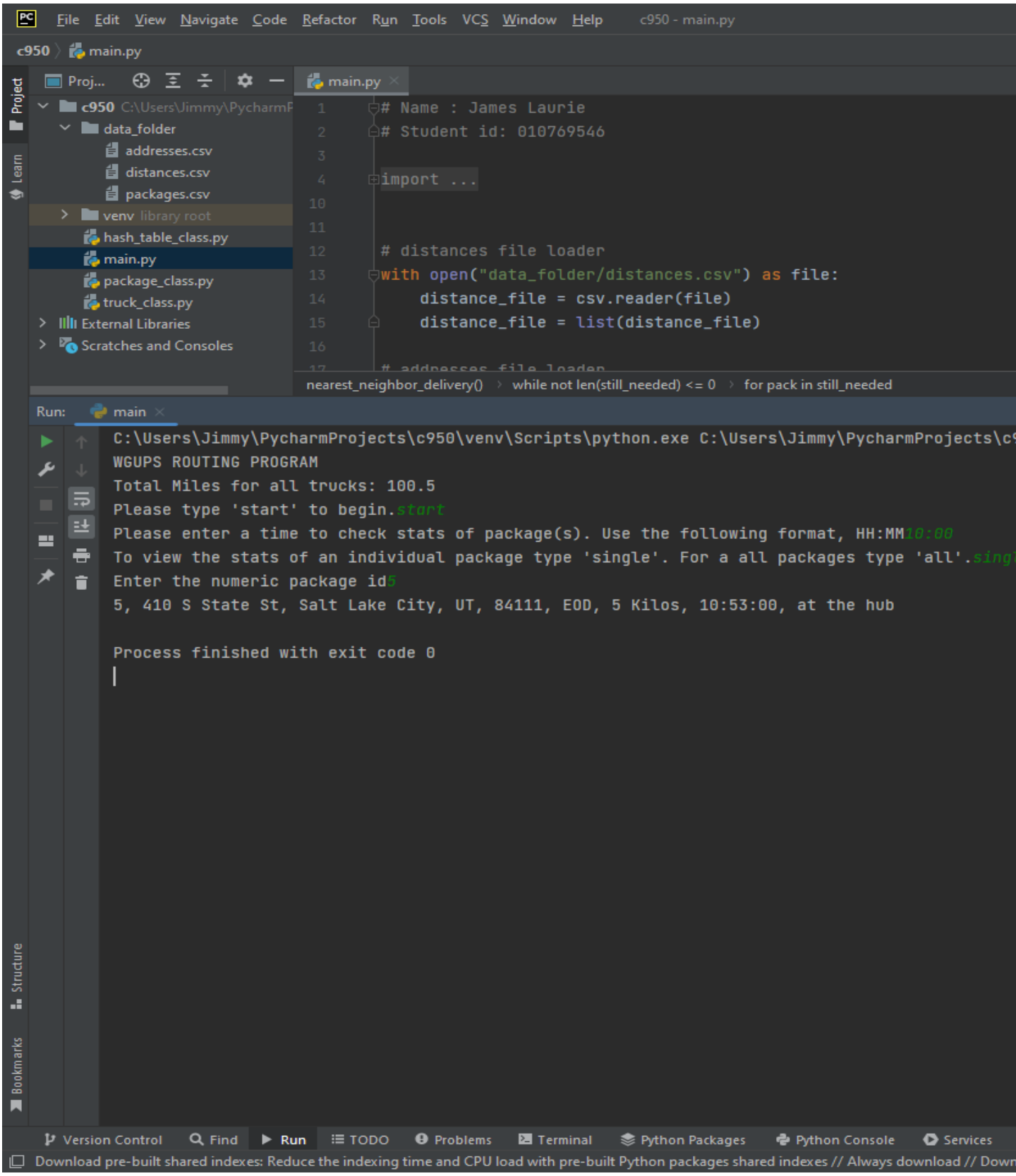
G3.

Selecting ‘all’ and searching for package statuses at 12:45.


File Edit View Navigate Code Refactor Run Tools VCS Window Help c950
c950 main.py
Project
Learn
Run: main
Please enter a time to check stats of package(s). Use the following format, HH:MM12:45
To view the stats of an individual package type 'single'. For a all packages type 'all'.all
1, 195 W Oakland Ave, Salt Lake City, UT, 84115, 10:30 AM, 21 Kilos, 8:39:00, delivered
2, 2530 S 500 E, Salt Lake City, UT, 84106, EOD, 44 Kilos, 9:29:00, delivered
3, 233 Canyon Rd, Salt Lake City, UT, 84103, EOD, 2 Kilos, 12:37:00, delivered
4, 380 W 2880 S, Salt Lake City, UT, 84115, EOD, 4 Kilos, 9:36:00, delivered
5, 410 S State St, Salt Lake City, UT, 84111, EOD, 5 Kilos, 10:53:00, delivered
6, 3060 Lester St, West Valley City, UT, 84119, 10:30 AM, 88 Kilos, 9:46:40, delivered
7, 1330 2100 S, Salt Lake City, UT, 84106, EOD, 8 Kilos, 10:37:40, delivered
8, 300 State St, Salt Lake City, UT, 84103, EOD, 9 Kilos, 10:56:20, delivered
9, 300 State St, Salt Lake City, UT, 84103, EOD, 2 Kilos, 10:56:20, delivered
10, 600 E 900 South, Salt Lake City, UT, 84105, EOD, 1 Kilos, 10:47:00, delivered
11, 2600 Taylorsville Blvd, Salt Lake City, UT, 84118, EOD, 1 Kilos, 10:03:00, delivered
12, 3575 W Valley Central Station bus Loop, West Valley City, UT, 84119, EOD, 1 Kilos, 10:33:00, delivered
13, 2010 W 500 S, Salt Lake City, UT, 84104, 10:30 AM, 2 Kilos, 9:12:40, delivered
14, 4300 S 1300 E, Millcreek, UT, 84117, 10:30 AM, 88 Kilos, 8:06:20, delivered
15, 4580 S 2300 E, Holladay, UT, 84117, 9:00 AM, 4 Kilos, 8:13:00, delivered
16, 4580 S 2300 E, Holladay, UT, 84117, 10:30 AM, 88 Kilos, 8:13:00, delivered
17, 3148 S 1100 W, Salt Lake City, UT, 84119, EOD, 2 Kilos, 10:53:00, delivered
18, 1488 4800 S, Salt Lake City, UT, 84123, EOD, 6 Kilos, 11:07:20, delivered
19, 177 W Price Ave, Salt Lake City, UT, 84115, EOD, 37 Kilos, 10:28:20, delivered
20, 3595 Main St, Salt Lake City, UT, 84115, 10:30 AM, 37 Kilos, 8:48:00, delivered
21, 3595 Main St, Salt Lake City, UT, 84115, EOD, 3 Kilos, 10:26:40, delivered
22, 6351 South 900 East, Murray, UT, 84121, EOD, 2 Kilos, 11:34:20, delivered
23, 5100 South 2700 West, Salt Lake City, UT, 84118, EOD, 5 Kilos, 11:09:20, delivered
24, 5025 State St, Murray, UT, 84107, EOD, 7 Kilos, 11:24:20, delivered
25, 5383 South 900 East #104, Salt Lake City, UT, 84117, 10:30 AM, 7 Kilos, 9:13:00, delivered
26, 5383 South 900 East #104, Salt Lake City, UT, 84117, EOD, 25 Kilos, 11:30:00, delivered
27, 1060 Dalton Ave S, Salt Lake City, UT, 84104, EOD, 5 Kilos, 12:17:40, delivered
28, 2835 Main St, Salt Lake City, UT, 84115, EOD, 7 Kilos, 9:32:40, delivered
29, 1330 2100 S, Salt Lake City, UT, 84106, 10:30 AM, 2 Kilos, 8:29:40, delivered
30, 300 State St, Salt Lake City, UT, 84103, 10:30 AM, 1 Kilos, 9:26:40, delivered
31, 3365 S 900 W, Salt Lake City, UT, 84119, 10:30 AM, 1 Kilos, 8:53:20, delivered
32, 3365 S 900 W, Salt Lake City, UT, 84119, EOD, 1 Kilos, 9:41:40, delivered
33, 2530 S 500 E, Salt Lake City, UT, 84106, EOD, 1 Kilos, 9:29:00, delivered
34, 4580 S 2300 E, Holladay, UT, 84117, 10:30 AM, 2 Kilos, 8:13:00, delivered
35, 1060 Dalton Ave S, Salt Lake City, UT, 84104, EOD, 88 Kilos, 12:17:40, delivered
36, 2300 Parkway Blvd, West Valley City, UT, 84119, EOD, 88 Kilos, 10:43:20, delivered
37, 410 S State St, Salt Lake City, UT, 84111, 10:30 AM, 2 Kilos, 9:23:20, delivered
38, 410 S State St, Salt Lake City, UT, 84111, EOD, 9 Kilos, 12:33:40, delivered
39, 2010 W 500 S, Salt Lake City, UT, 84104, EOD, 9 Kilos, 12:23:00, delivered
40, 380 W 2880 S, Salt Lake City, UT, 84115, 10:30 AM, 45 Kilos, 8:42:40, delivered
Structure
Bookmarks
Version Control Find Run TODO Problems Terminal Python Packages Python Console Services
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Python packages shared indexes // Always download // Download once // Don't show again // Configure... (4/2/2023 10:14

H.:


File Edit View Navigate Code Refactor Run Tools VCS Window Help
c950 - main.py
c950 main.py
main.py
Name : James Laurie
Student id: 010769546
import ...
distances file loader
with open("data_folder/distances.csv") as file:
distance_file = csv.reader(file)
distance_file = list(distance_file)
data_folder
addresses.csv
distances.csv
packages.csv
venv library root
hash_table_class.py
main.py
package_class.py
truck_class.py
External Libraries
Scratches and Consoles
nearest_neighbor_delivery() while not len(still_needed) <= 0 for pack in still_needed
Run: main
C:\Users\Jimmy\PycharmProjects\c950\venv\Scripts\python.exe C:\Users\Jimmy\PycharmProjects\c950\main.py
WGUPS ROUTING PROGRAM
Total Miles for all trucks: 108.5
Please type 'start' to begin.start
Please enter a time to check stats of package(s). Use the following format, HH:MM10:00
To view the stats of an individual package type 'single'. For a all packages type 'all'.single
Enter the numeric package id5
5, 410 S State St, Salt Lake City, UT, 84111, EOD, 5 Kilos, 10:53:00, at the hub
Process finished with exit code 0
Version Control Find Run TODO Problems Terminal Python Packages Python Console Services
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Python packages shared indexes // Always download // Download once // Don't show again // Configure... (4/2/2023 10:14 AM)
11:1 CRLF UTF-8 4 spaces Python 3.10 (c950)


File Edit View Navigate Code Refactor Run Tools VCS Window Help
c950 - main.py
c950 main.py
Proj...
main.py
c950 C:\Users\Jimmy\PycharmP
data_folder
addresses.csv
distances.csv
packages.csv
venv library root
hash_table_class.py
main.py
package_class.py
truck_class.py
External Libraries
Scratches and Consoles
1 # Name : James Laurie
2 # Student id: 010769546
4 import ...
12 # distances file loader
13 with open("data_folder/distances.csv") as file:
14 distance_file = csv.reader(file)
15 distance_file = list(distance_file)
nearest_neighbor_delivery() while not len(still_needed) <= 0 for pack in still_needed
Run: main
C:\Users\Jimmy\PycharmProjects\c950\venv\Scripts\python.exe C:\Users\Jimmy\PycharmProjects\c9
WGUPS ROUTING PROGRAM
Total Miles for all trucks: 100.5
Please type 'start' to begin.start
Please enter a time to check stats of package(s). Use the following format, HH:MM10:00
To view the stats of an individual package type 'single'. For a all packages type 'all'.singl
Enter the numeric package id5
5, 410 S State St, Salt Lake City, UT, 84111, EOD, 5 Kilos, 10:53:00, at the hub
Process finished with exit code 0
Version Control Find Run TODO Problems Terminal Python Packages Python Console Services
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Python packages shared indexes // Always download // Down

I1. The nearest neighbor algorithm was chosen because it was the only algorithm that I intuitively understood at the start of the project. I knew I could find a more efficient algorithm, but I decided that implementing the algorithm myself would help me improve my programming skills more. The nearest neighbor algorithm's strength is that it can handle increasing amounts of data while remaining relatively efficient. In general, this algorithm is extremely adaptable and can be modified to meet changing needs. Another advantage is that it is simple to grasp. If this were a production application with a constantly changing team, new developers would have a short learning curve.

I2.

Because each package is delivered before its deadline, the algorithm used in the solution meets all requirements. All package constraints are taken into account, such as which packages must be transported on a curtain truck or on the same truck as another package. I also made certain that the trucks left at times that allowed for any special cases, such as package number 9 having an incorrect address until 10:20. The trucks all contained fewer than 16 packages and delivered all packages in a total distance of 100.5 miles.

I3.

Dijkstra's shortest path and some type of insertion algorithm, such as farthest insertion, are two other algorithms that could be used for this assignment.

I3A.

Dijkstra’s algorithm is different from the nearest neighbor in that it begins from the source node (in this case the hub) and calculates the shortest distances between it and all other nodes on the graph (Ginting, 3). For this project, this algorithm employs weighted edges between each vertex, which are the pieces of distance data associated with each destination. This would almost certainly find a better solution than the nearest neighbor.

J.

Like the name suggests, the farthest insertion algorithm starts with the starting point and the destination that is farthest away. What makes this algorithm find an efficient path is that it incorporates those out of the way cities first, which eventually leads to an optimal path (Mumford, 1). This algorithm would also find a much more efficient route than the nearest neighbor, and it is the one I would use again if I were given this assignment.

K1.

If I were to redo this project, I would start by creating a function to load the trucks based on time constraints and proximity to one another. For this submission, I gave no weight to route optimization; I was more concerned with practicing object-oriented programming in Python, which I had no prior experience with. I would also prioritize writing code that can adapt to new data, such as additional packages, deadlines, available trucks/drivers, or constraints. If I were to try again, these would be my top priorities.

K1A.

The hash table is an excellent choice for this assignment. It is fast and can be used to load/update packages as needed. The most compelling justification for this data structure is that all packages were delivered on time, despite all constraints. Because hash tables perform searches in O(1) time on average, the time required to complete the look-up function for packages does not change as the number of packages increases or decreases. If the table is far too small for the number of packages, the worst-case scenario is an O(n)-time search, which is not bad when compared to other data structures.

K1B.

Because hash tables have a space complexity of O, the amount of space used by a hashmap changes proportionally to the number of packages to be delivered.(n).

K1C.

Changes in the number of trucks in the WGUPS fleet would increase space usage proportionally to the number of trucks, as each truck has its own list of packages to deliver. Adding cities, on the other hand, would exponentially increase space usage because each destination references one another in the delivery algorithm. Regardless of these changes, the look-up function would remain O(1).

K2.

A binary search tree or some type of queue would be suitable alternatives to the hash map for this application.

K2a.

In O(log n) time, the binary search tree performs all of the same tasks as the hash map. Hash maps may be slower in the worst-case scenario, despite the fact that all of these common operations are fundamentally less efficient than binary search trees. Binary search trees don't require resizing and use less RAM. .

A queue has the same time complexity as a hash table because they are both O(1), but a queue has a more efficient space complexity of O(1) when compared to hash tables, which are O(1).(n). However, the disadvantages of queues, such as having to delete an element in order to add a new one, may outweigh the benefits of efficient space complexity.

L.

Lysecky, R., & Vahid, F. (2018, June). *C950: Data Structures and Algorithms II*. zyBooks. Retrieved April 4, 2023, from https://learn.zybooks.com/zybook/WGUC950AY20182019/

Mumford, Christine. “Furthest Insertion.” *Farthest Insertion*, Cardiff University, Retrieved April 4, 2023, from https://users.cs.cf.ac.uk/C.L.Mumford/howard/FarthestInsertion.html

Ginting, Hagai Nuansa, et al. “Item Delivery Simulation Using Dijkstra Algorithm for Solving Traveling Salesman Problem.” *Journal of Physics: Conference Series*, vol. 1201, no. 1, 2019, p. 012068. Retrieved April 4, 2023, from https://doi.org/10.1088/1742-6596/1201/1/012068