Adam Brigham

Student ID: #001399776

Package Routing Project, C950

1. Algorithm identification:

The self-adjusting algorithm used in this program was based on the Nearest Neighbor Algorithm

1. Logic Comments:

Parameter for algorithm = Parameter IDs List

Algorithm IDs List = empty at start

Current Address = hub at start

Outer Loop -> while length(IDs List) < length(Parameter IDs List)

Minimum distance = 1000 (initial value > possible values within scope of algorithm)

Inner Loop -> for (int i = 0; i < parameter IDs list; i++)

If (distance between (package[i].address, current address) < minimum distance) && Parameter IDs List[i] is not in Algorithm IDs list

Minimum distance = distance between (package[i].address, current address)

Minimum distance index = i

Current address = package[minimum distance index]

Add Parameter IDs List [minimum distance index] to Algorithm IDs List

Return -> Algorithm IDs List

1. Development Environment:

PyCharm 2021.1.2 (Community Edition)

Build #PC-211.7442.45, built on June 1, 2021

Runtime version: 11.0.11+9-b1341.57 amd64

VM: Dynamic Code Evolution 64-Bit Server VM by JetBrains s.r.o.

Windows 10 10.0

GC: G1 Young Generation, G1 Old Generation

Memory: 2038M

Cores: 12

1. Space-time and Big-O:
   1. Nearest Neighbor Algorithm
      1. Time complexity O(n^2)
      2. Space complexity O(n)
   2. Hash Table
      1. Time complexity (insert, lookup, remove) O(1)
      2. Space complexity O(n)
   3. Parse Package
      1. Time complexity O(n)
      2. Space complexity O(n)
   4. Distance between
      1. Time complexity O(n)
      2. Space complexity O(n)
   5. Calculate time
      1. Time complexity O(n)
      2. Space complexity O(n)
   6. Begin deliveries
      1. Time complexity O(n)
      2. Space complexity O(n)
   7. Total program
      1. Time complexity O(n^2)
      2. Space complexity O(n)
2. Scalability and adaptability

The program is scalable to accommodate more packages because the list size is a variable, not a constant hardcoded into the program. The number of package objects is dependent on the number of rows in the csv source file. All the functions that utilize the package objects or lists of their ID’s, etc., calculate and use the package list size rather than a constant value (e.g., 40).

1. Software efficiency and maintainability

The software is easy to maintain because all variables and parameters are descriptive rather than ambiguous. The code is thoroughly commented, and it is organized in a logical way. It is efficient because the time-space complexity is at worst O(n^2).

1. Self-adjusting data structures

Strengths of the self-adjusting data structures. Hash tables have very fast lookup, insert, and remove functionality (O(1) 1 if there are no collisions). Linear searches are not required since each key is mapped to a value; one only must use the key for hash table functionality. Disadvantages to hash tables: a null value cannot be used as a key. Collisions can decrease efficiency.

The hash table meets all requirements of the program because it stores the package objects and can access them quickly. All the fields for the packages are easily accessible for use in algorithms used throughout the program. As the number of packages scales, as long as there are minimal collisions, time complexity remains O(1) 1. The space usage is linear with the number of packages, as the values are stored in a list. If there are collisions, then it has lists within lists, because the hash table handles collisions with chaining.

Adding more addresses or trucks would not affect the hash table’s lookup efficiency, as long as the number of collisions of package objects in the hash table buckets did not increase also.

Besides hash tables, an array list could be used to hold package objects. It would be less efficient because it would require less efficient searches, O(n) 1, so lookups would take considerable more time, and the program would be less scalable. A binary search algorithm could not be used in this context to make the list searches more efficient because it would require sorting them first, so sorting them with the nearest neighbor algorithm would preclude binary searches.

Binary Search Tree could also be used. It would be more difficult to implement than a list, but it would be more efficient for lookups, having O(log(n)) efficiency1.

Other data structures used in this program besides the hash table:

Lists – advantages: simplicity. Disadvantages: low efficiency when sorting and searching

2d arrays – advantages: simplicity, can be visualized as matrix. Disadvantages: efficiencies are low since accessing a value requires nested loops (O(n^2))

1. Strengths of the chosen algorithm

Nearest neighbor algorithm is advantageous in this case because the distances are held in a 2d array. Therefore, it is easy to implement a distance between function to access their indexes and be used in the nearest neighbor algorithm without implementing a tree data structure. It is simple and easy to understand and requires few parameters. However, it gets slower as the n and independent variables increase2.

1. Other possible algorithms and their differences

Prim’s algorithm used with a minimum spanning tree (MST). If the addresses were held as nodes in a weighted graph, and the edges were weighted with the distances between the nodes, then a MST could be created using Prim’s algorithm to find the shortest distance to visit all nodes3.

Dijkstra’s shortest path algorithm could also be used with a similar weighted graph data structure. [Section 6.11 - C950: Data Structures and Algorithms II (zybooks.com)](https://learn.zybooks.com/zybook/WGUC950AY20182019/chapter/6/section/11). Both of these algorithms are different from the nearest neighbor algorithm because they require a fundamentally different data structure, an ordered graph. The data structures used to hold the addresses and distances for this program was a list and 2d array respectively.

1. Different approach

If I were to do this project differently, I would attempt to load the trucks with a greedy algorithms, rather than manual loading. Given my time constraints with completing this project, it was not feasible. However, I may work on this in my spare time after the completion of C950. I would also use a weighted graph data structure with Prim’s algorithm to construct a MST to solve the problem presented in this project.

Resources

1 Big-O Algorithm Complexity Cheat Sheet (Know Thy Complexities!)

Author: Eric Drowell, Date: Not disclosed

bigocheatsheet.com

2 Machine Learning Basics with the K-Nearest Neighbors Algorithm

Author: Onel Harrison, Date: 10 Sep 2018

towardsdatascience.com

3 Prim’s Minimum Spanning Tree (MST) | Greedy Algo-5

Author: Not Disclosed, Date: 03 Nov 2020

geeksforgeeks.com

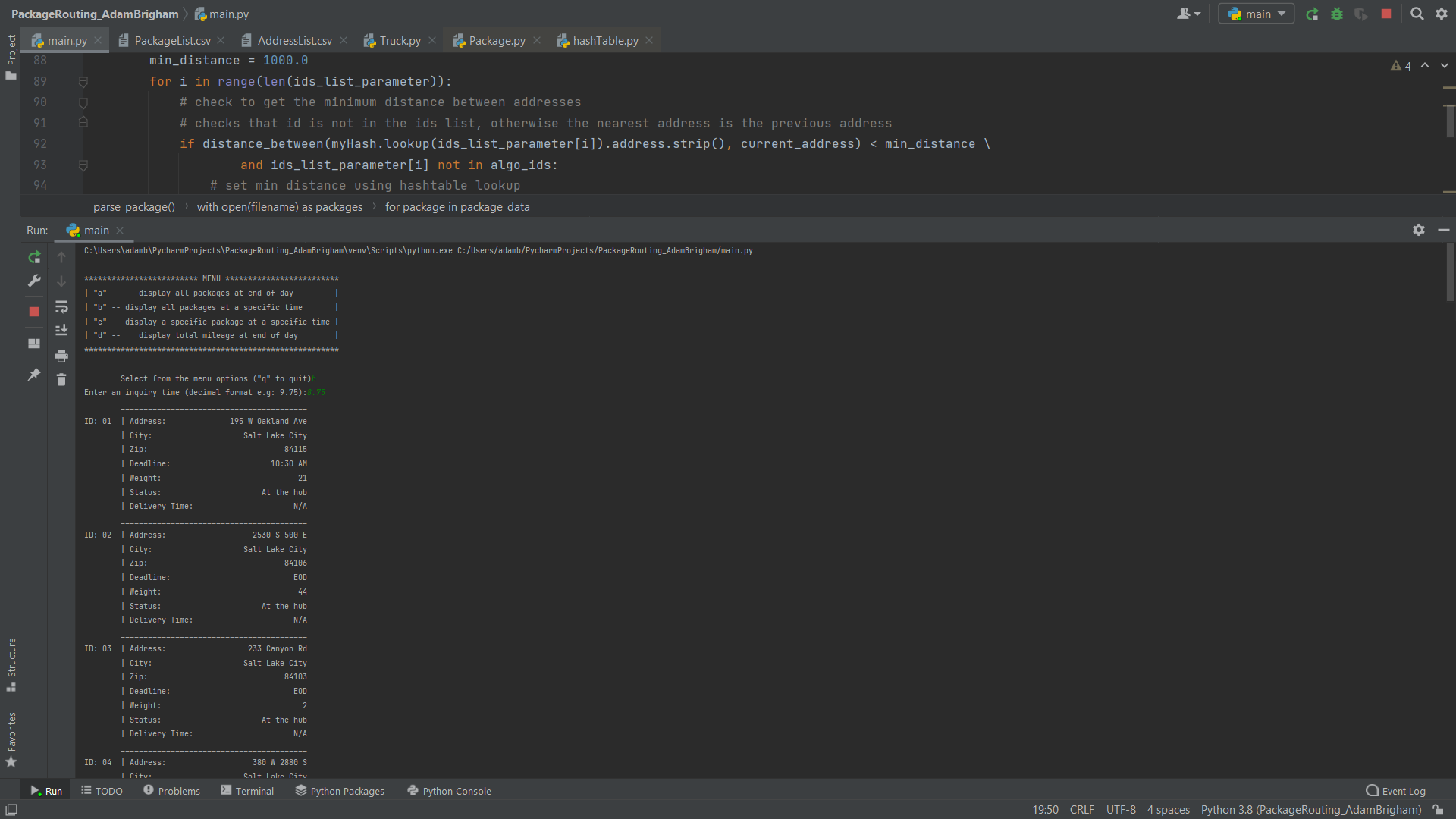
4 Algorithm: Dijkstra's shortest path

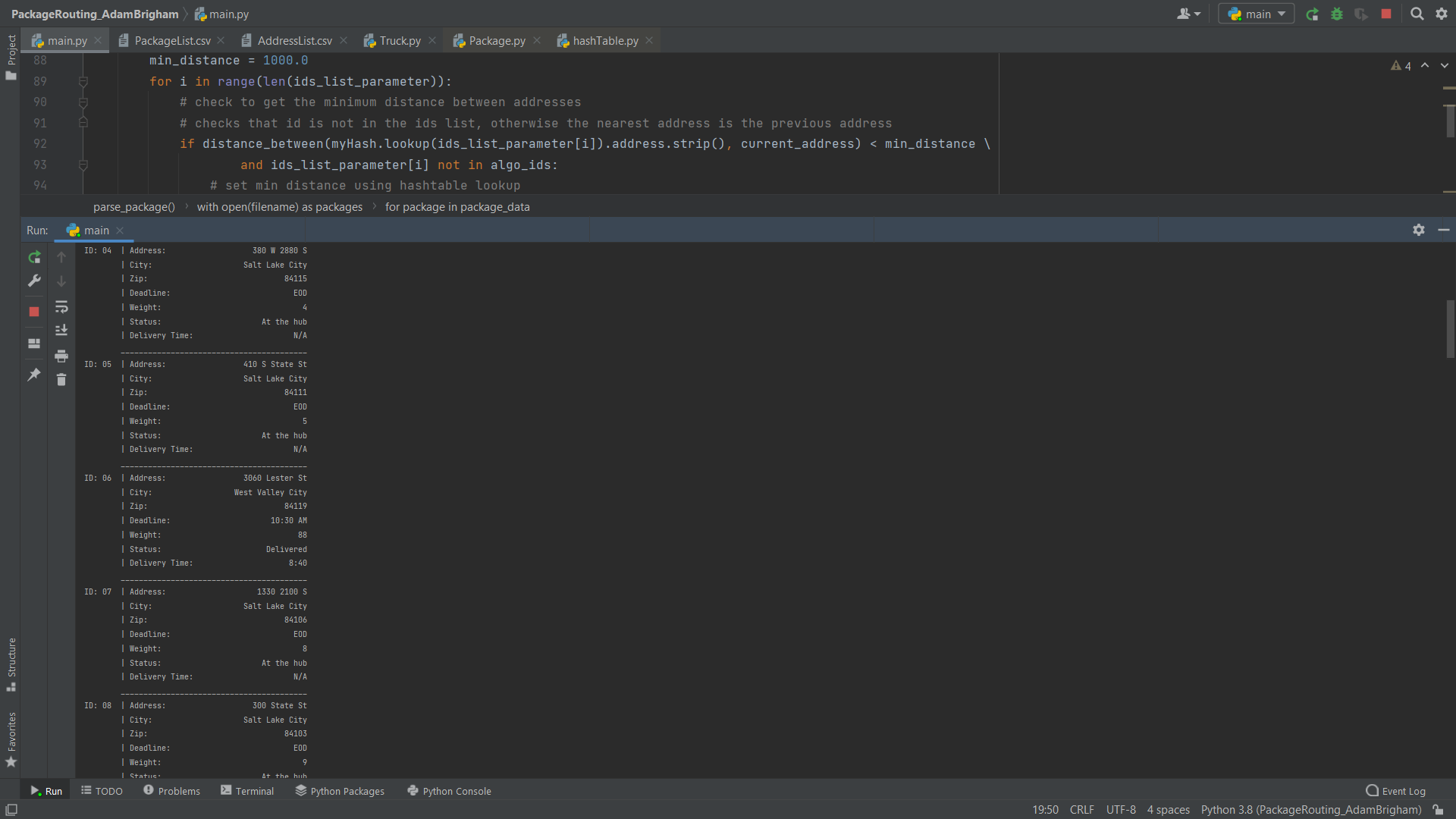
Author: Not Disclosed, Date: Not Disclosed

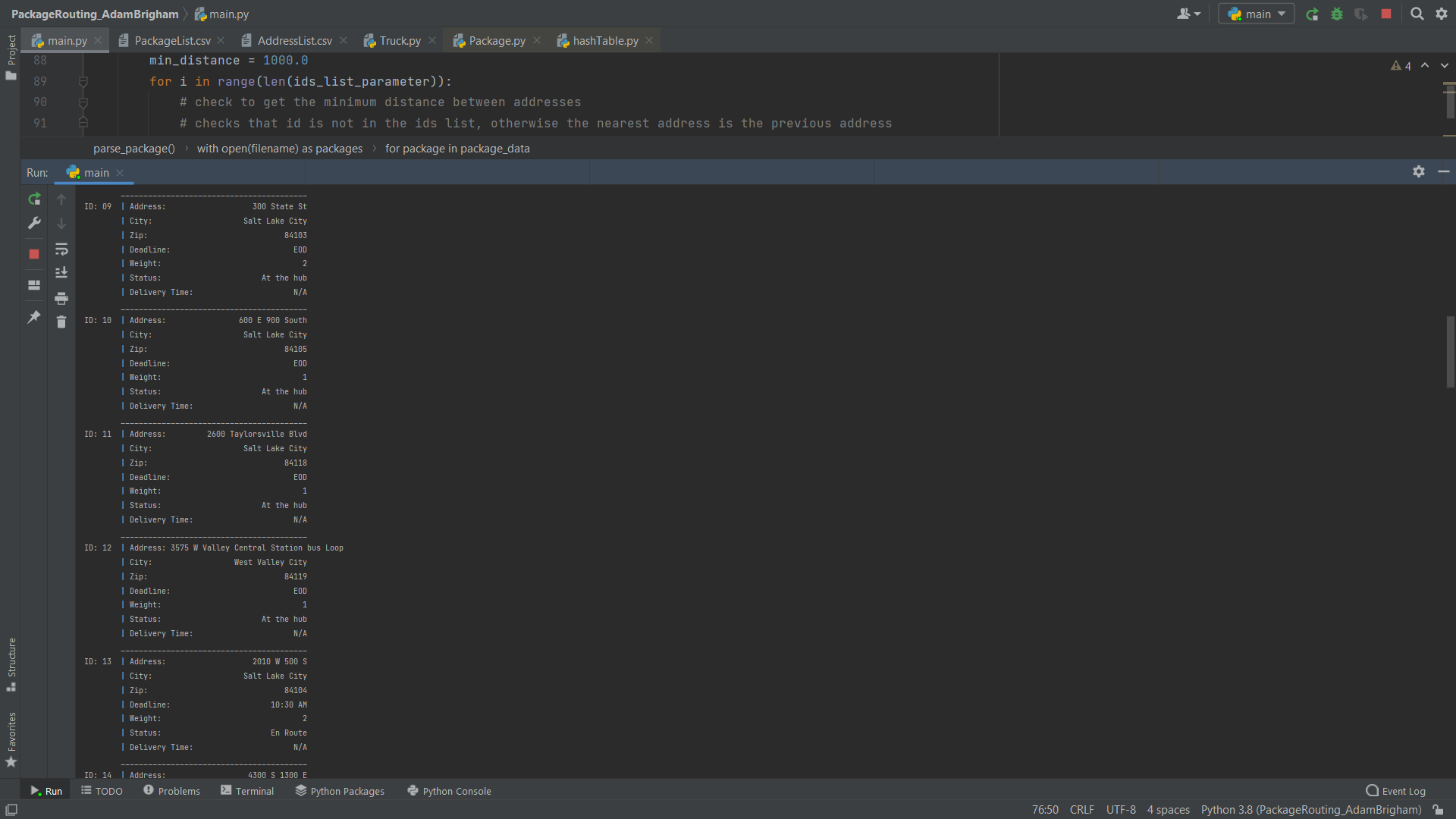
Section 6.11 - C950: Data Structures and Algorithms II (zybooks.com)

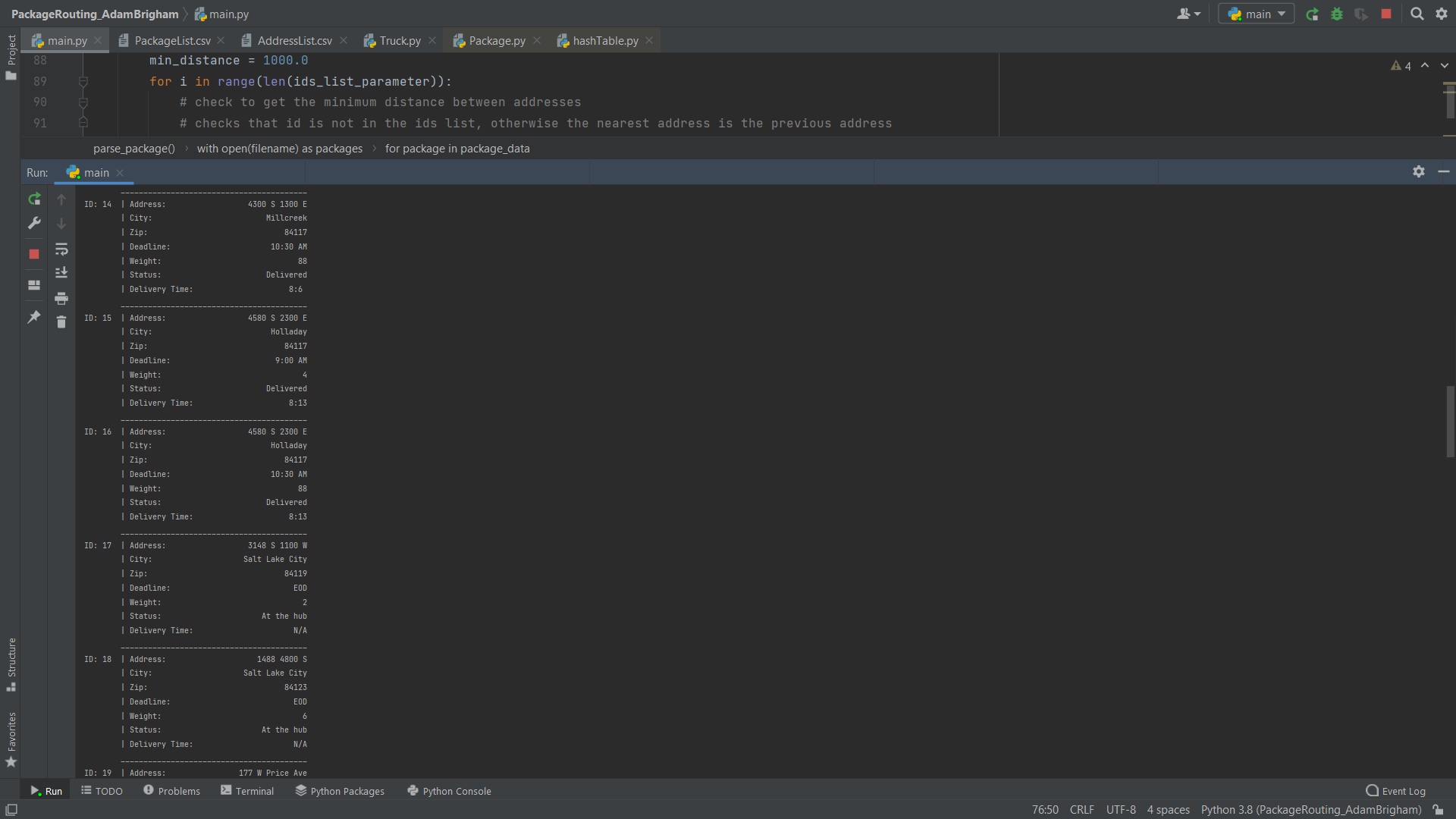
Screenshots:

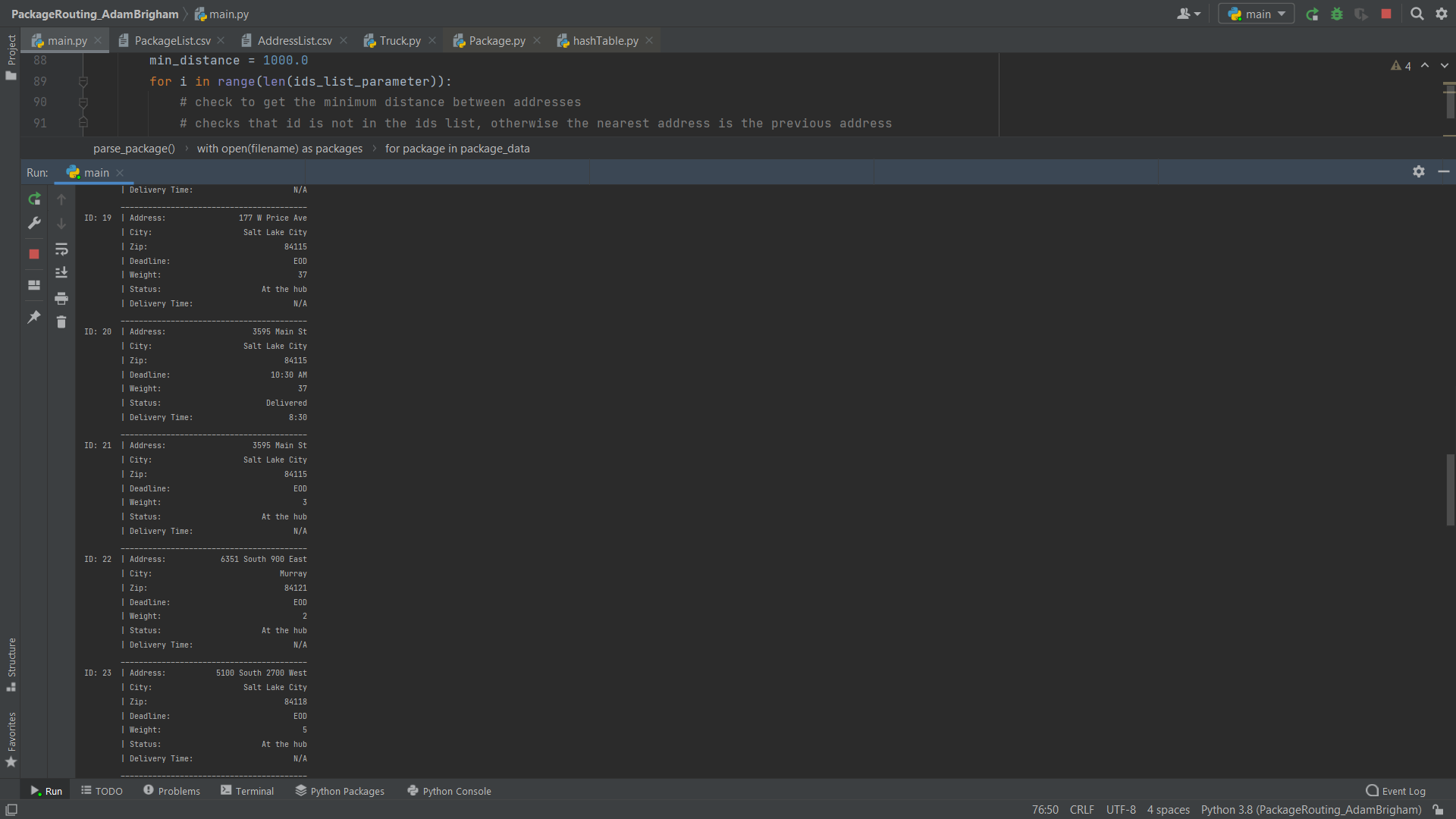
Status check 8:45 am

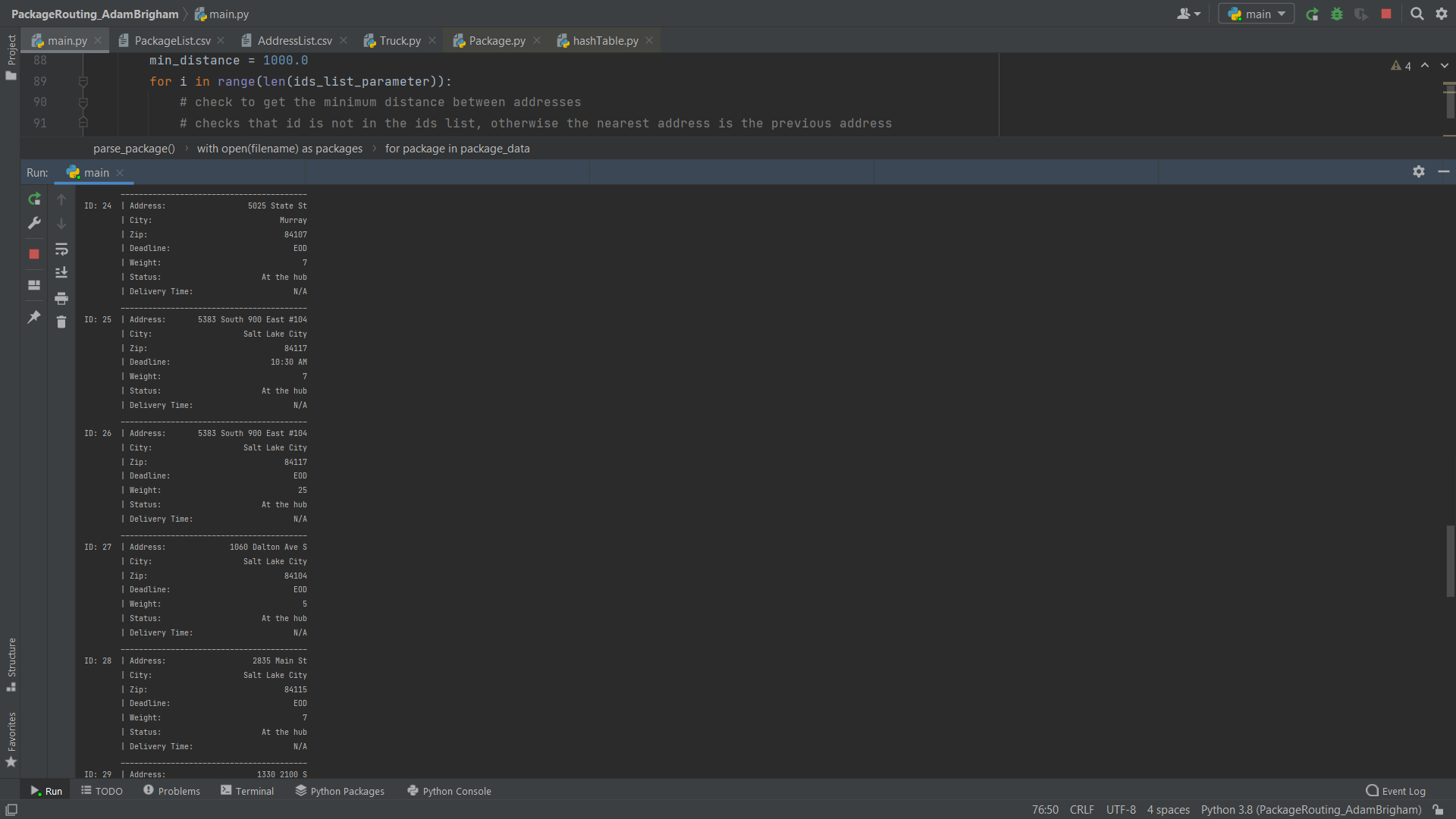


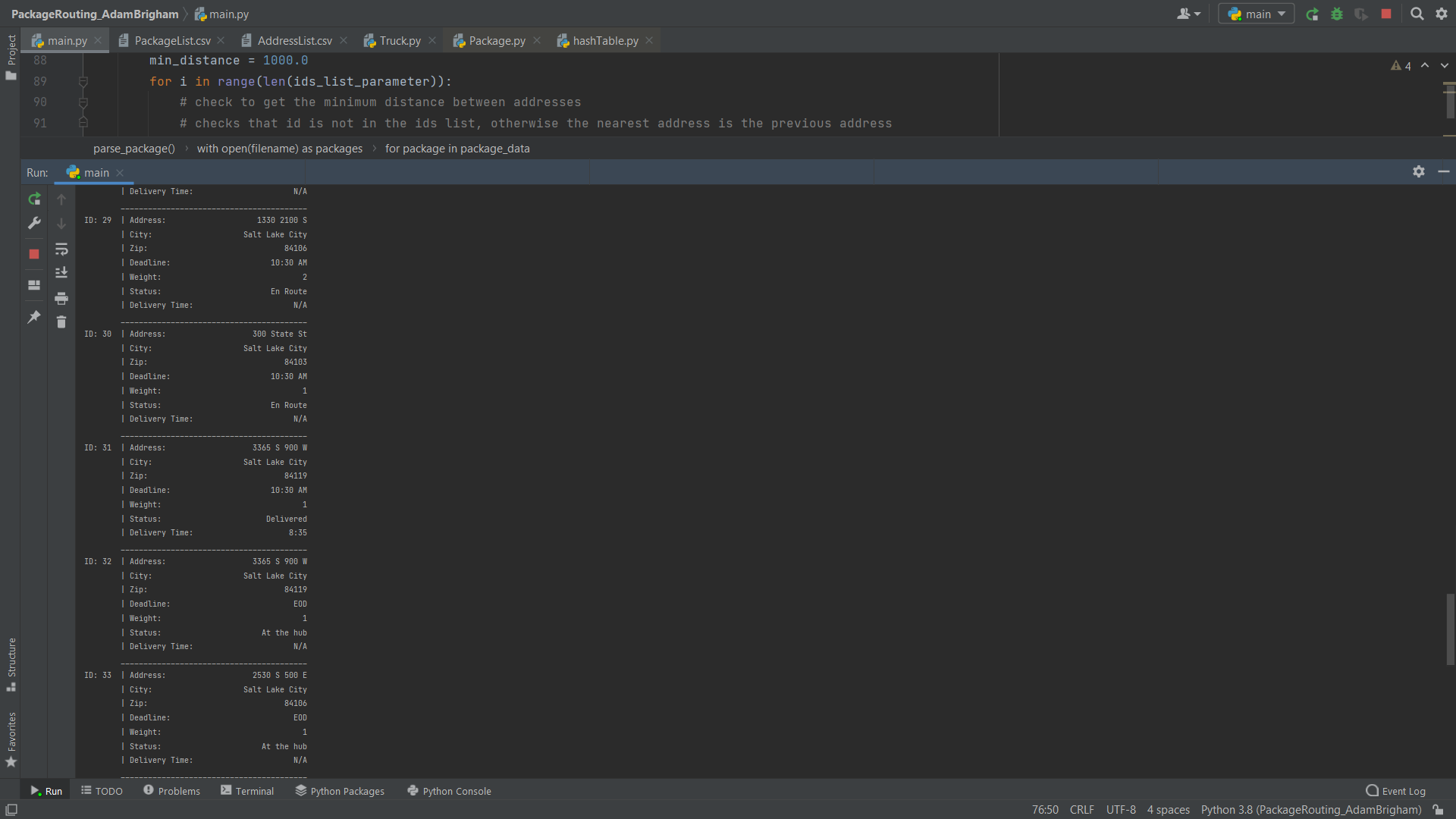


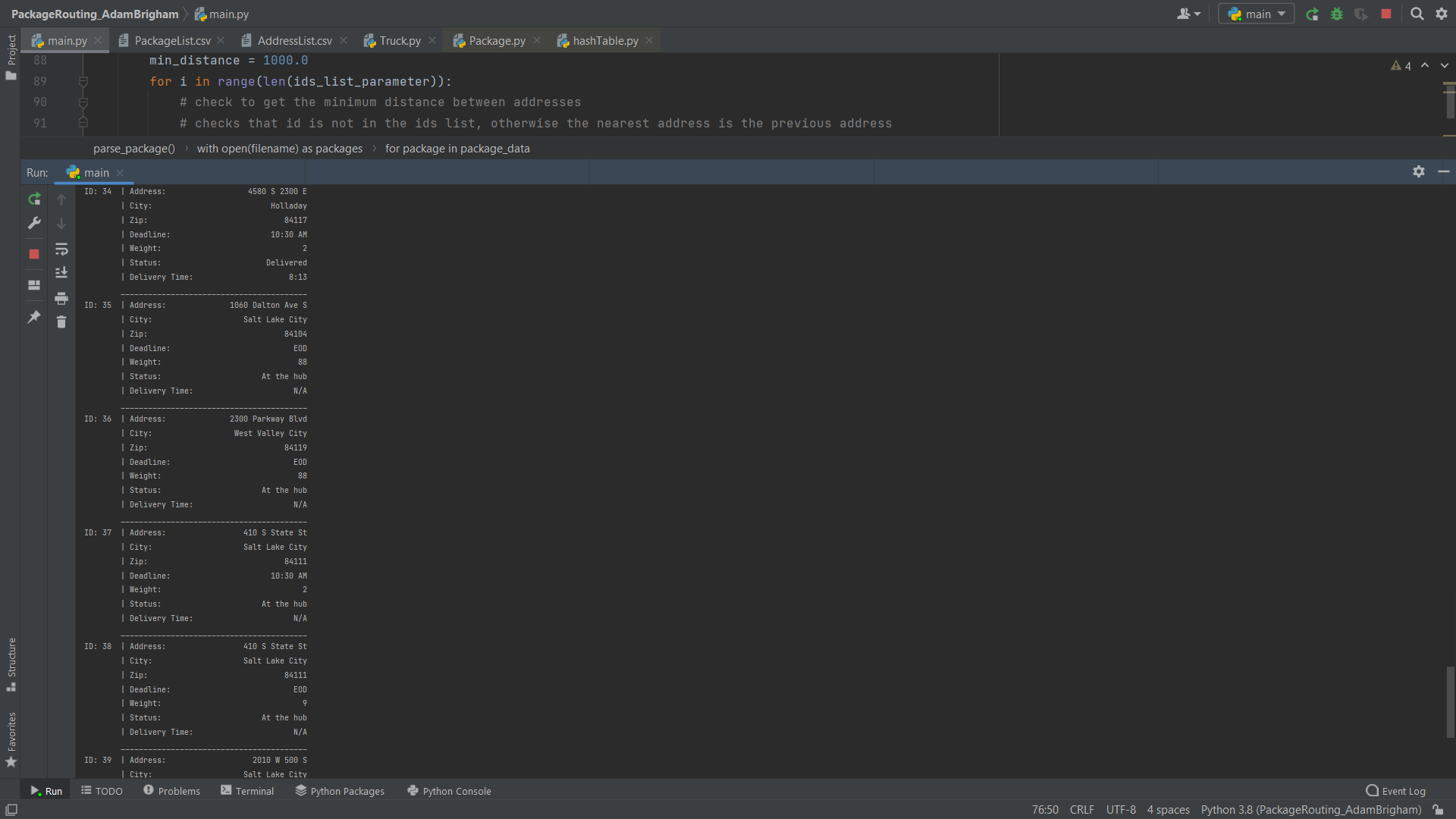


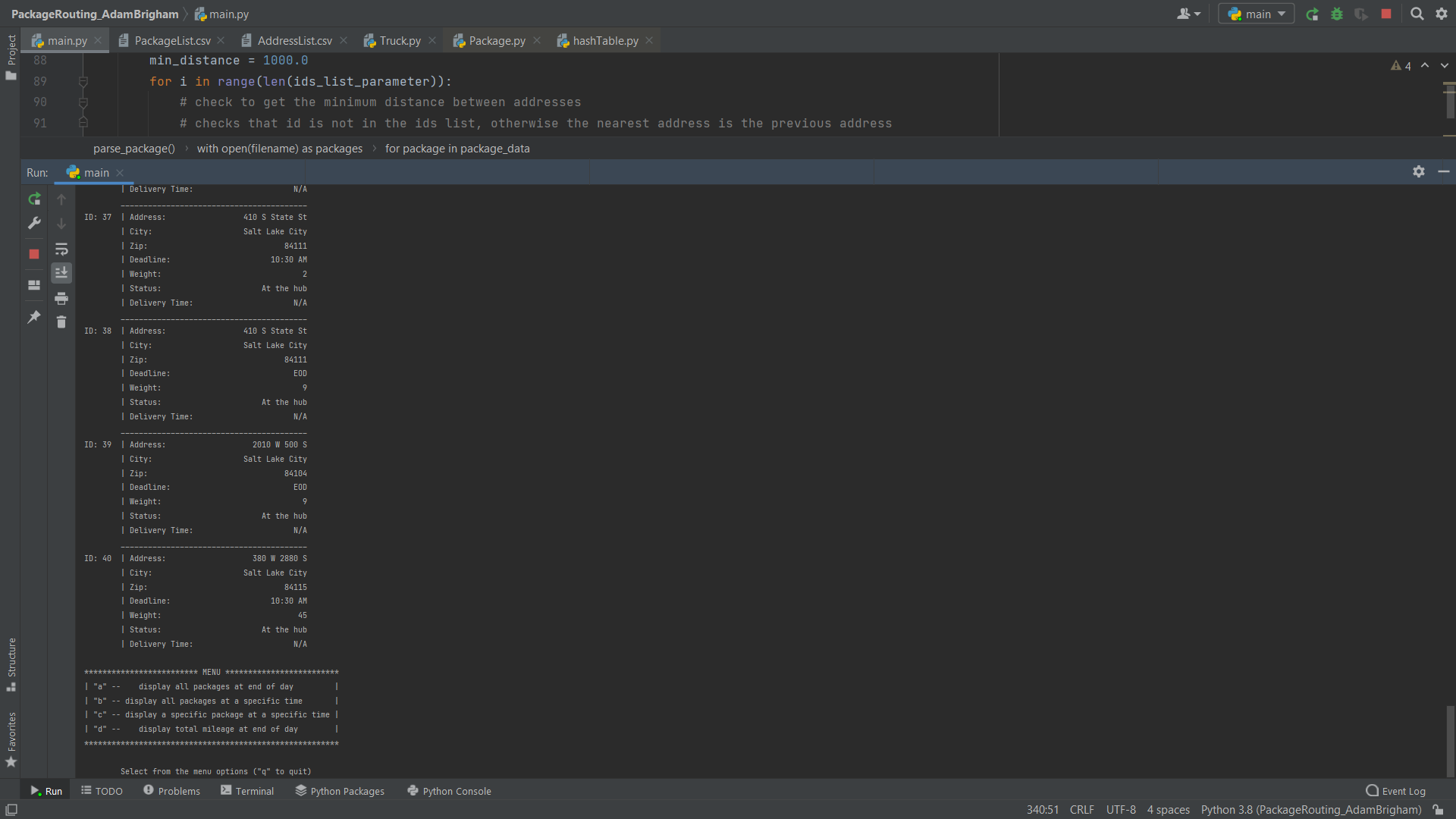




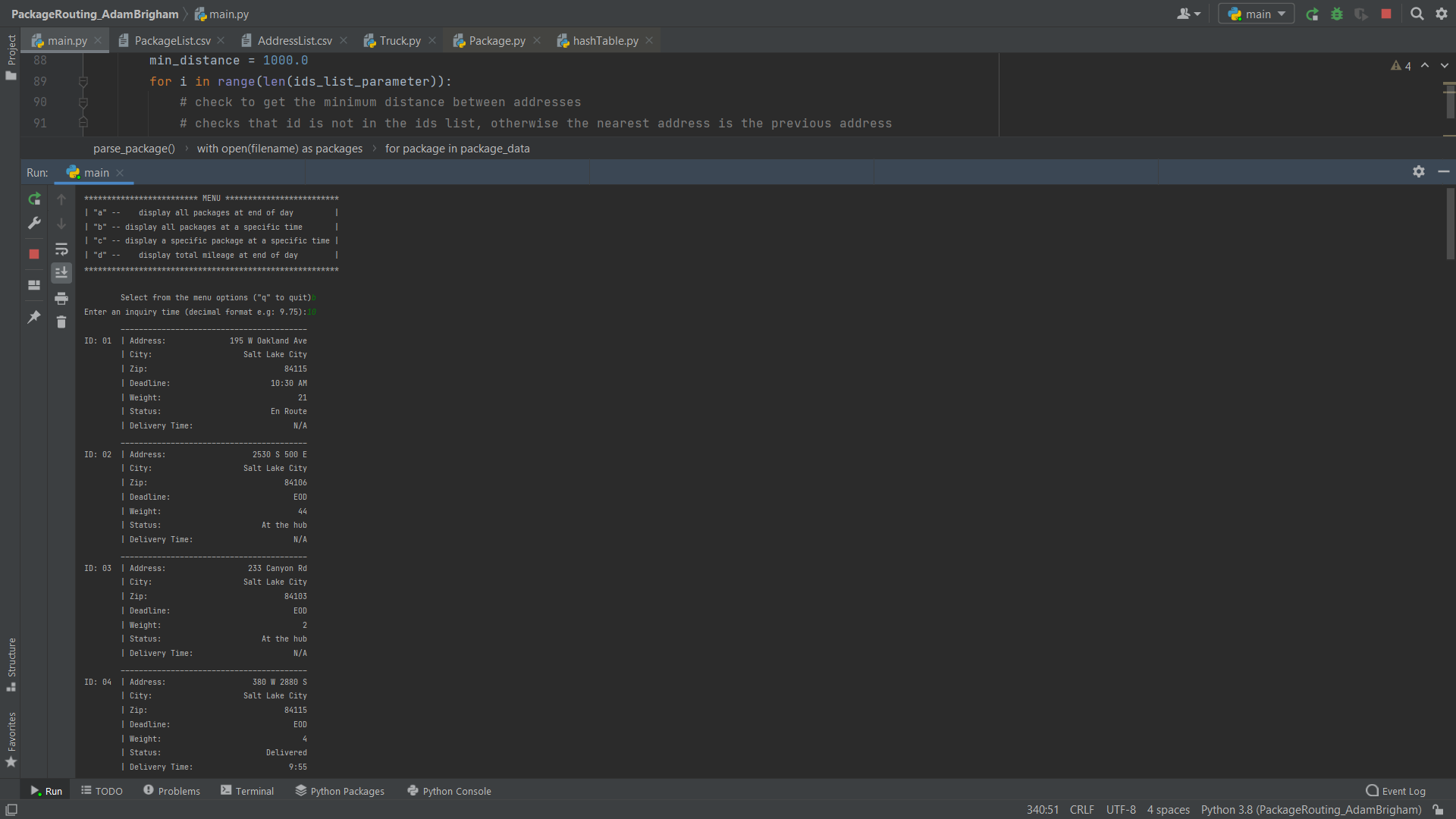


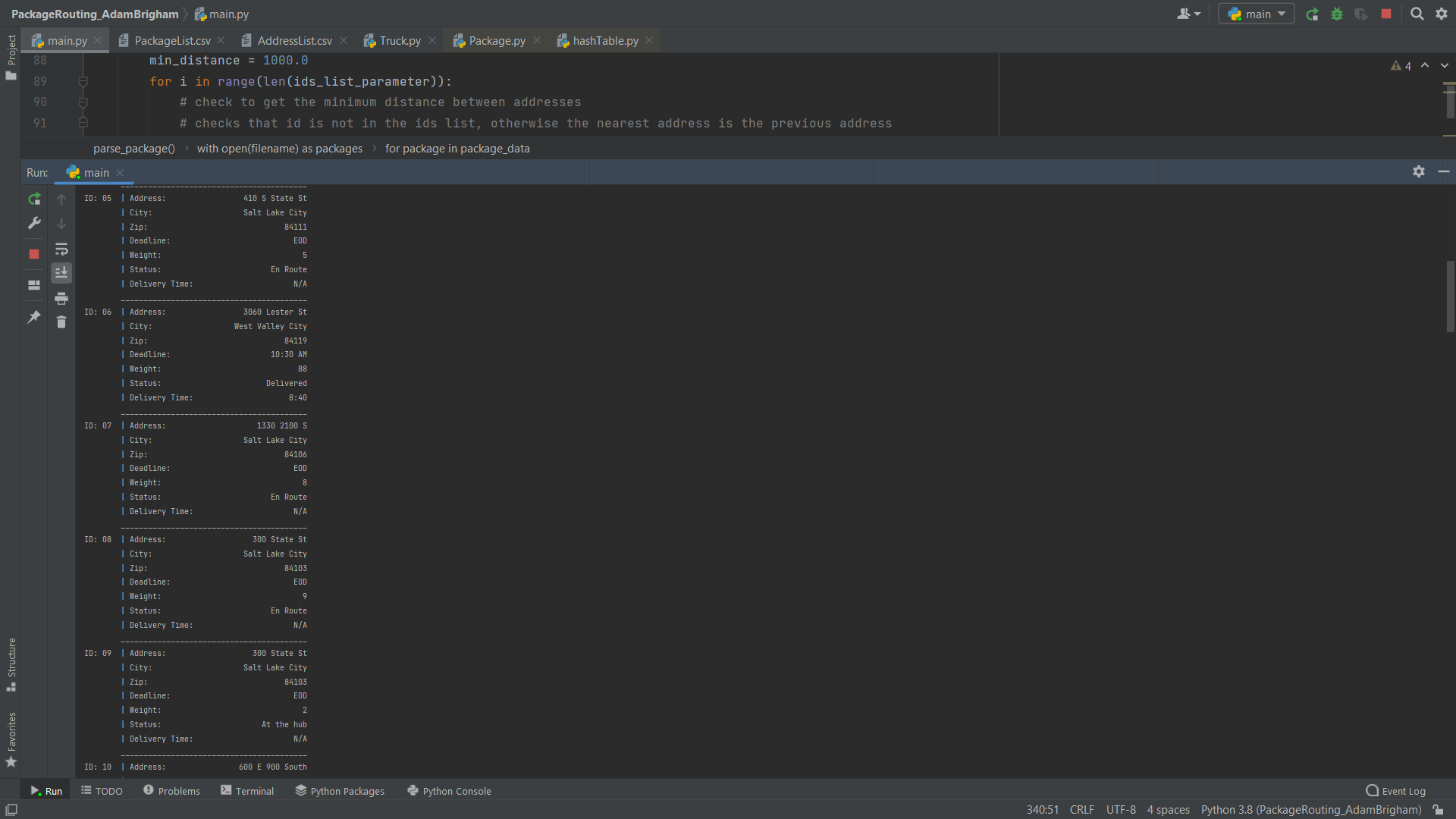


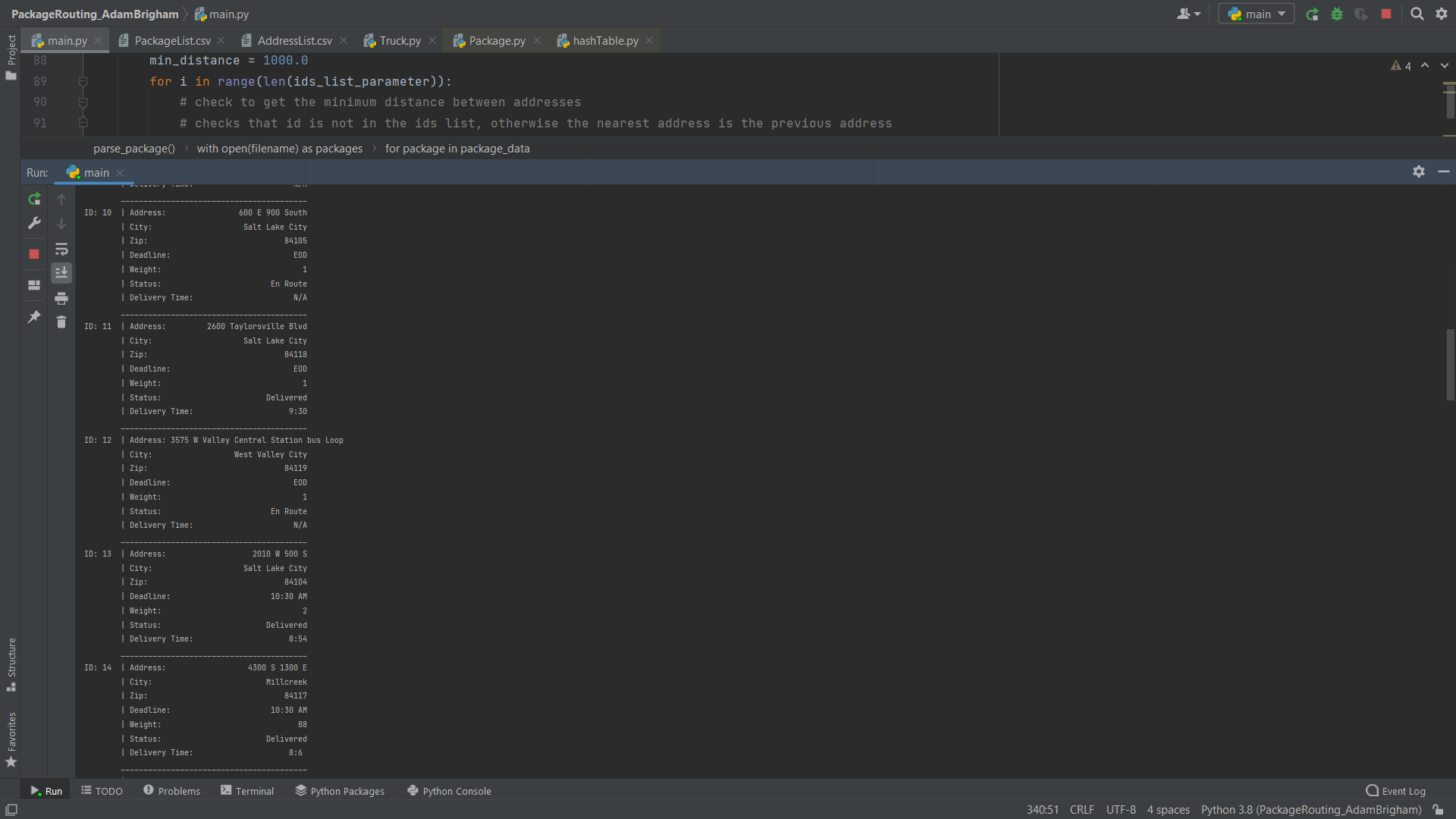


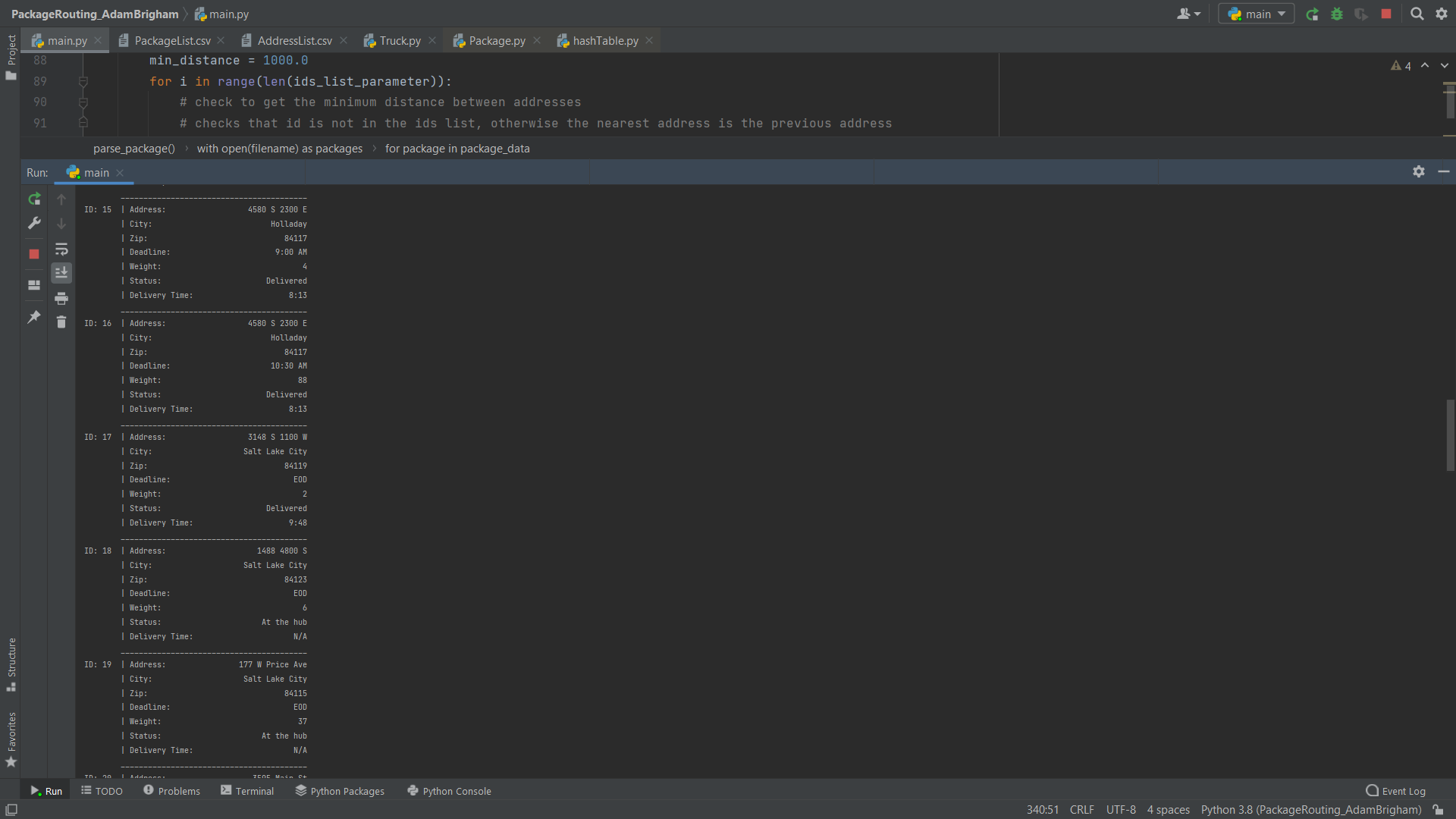


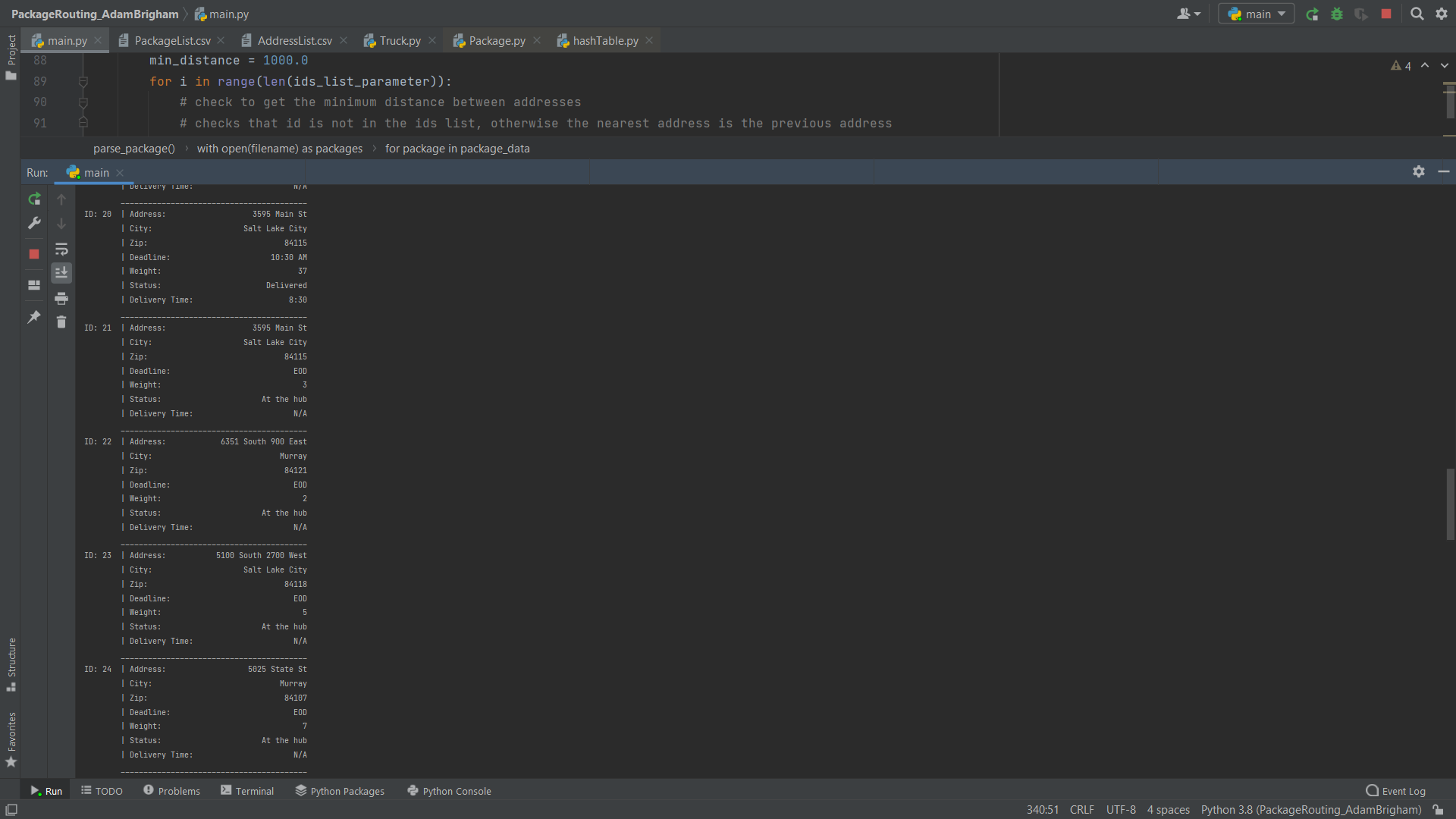
Status check 10:00 am

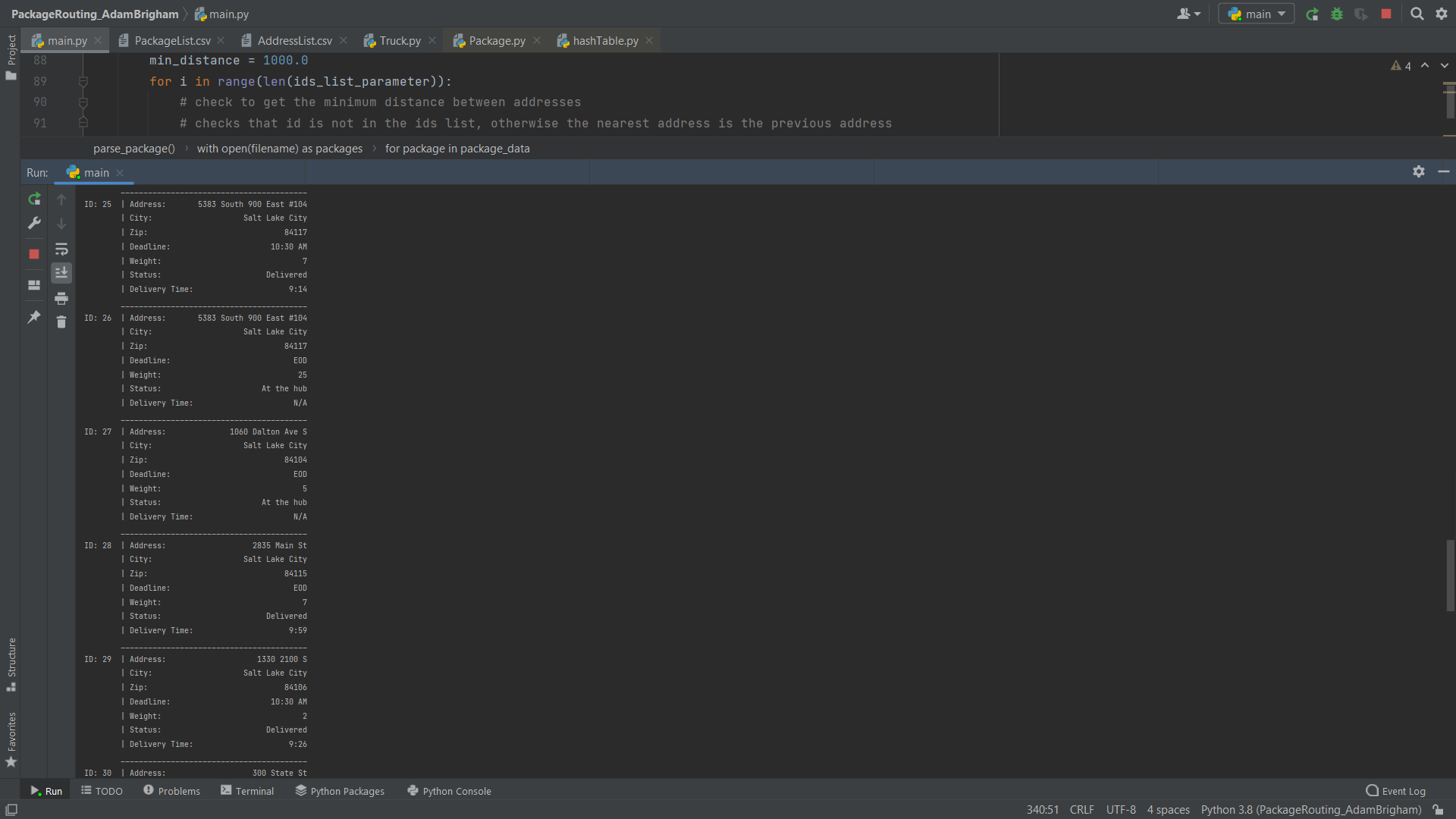


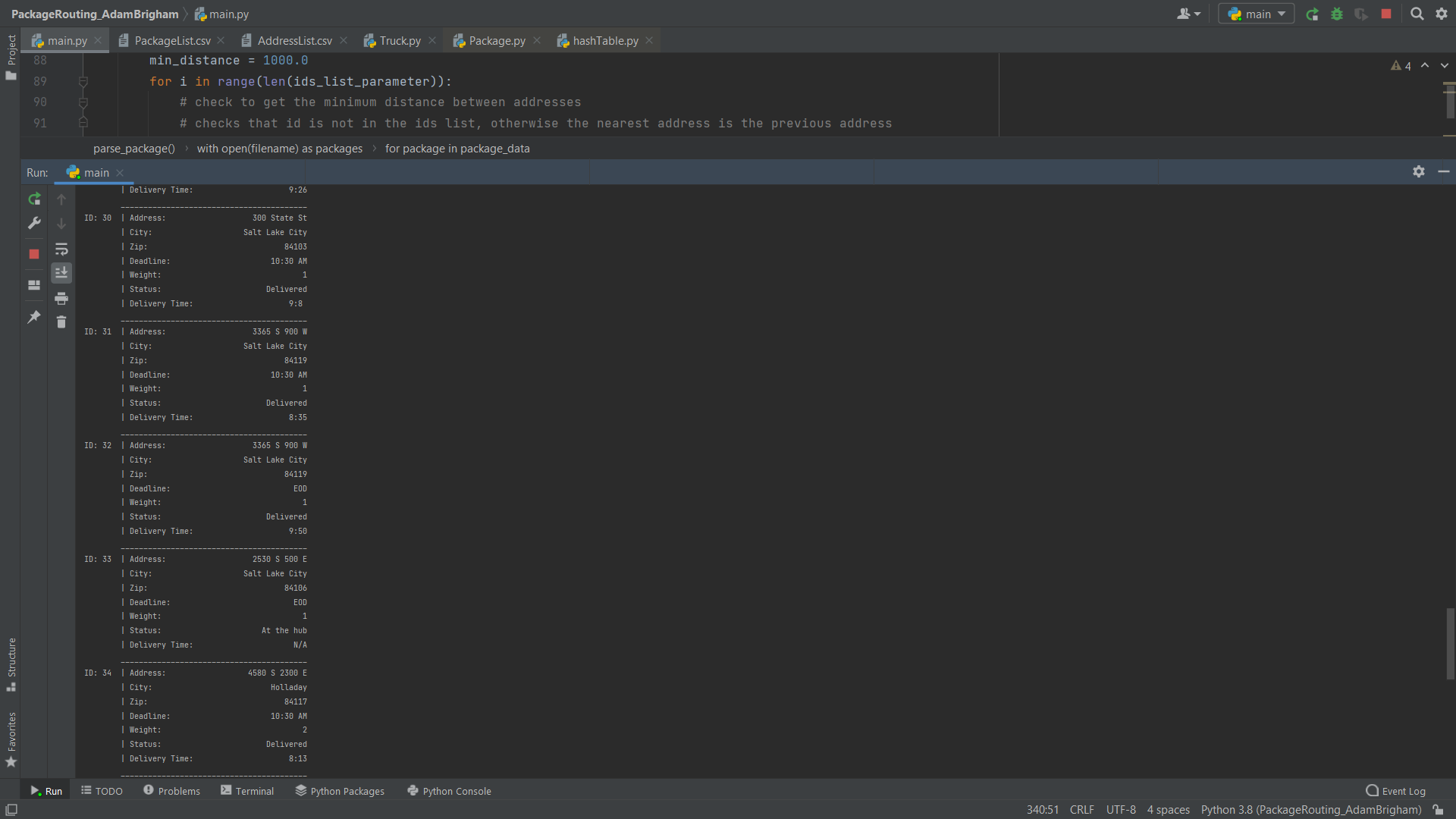


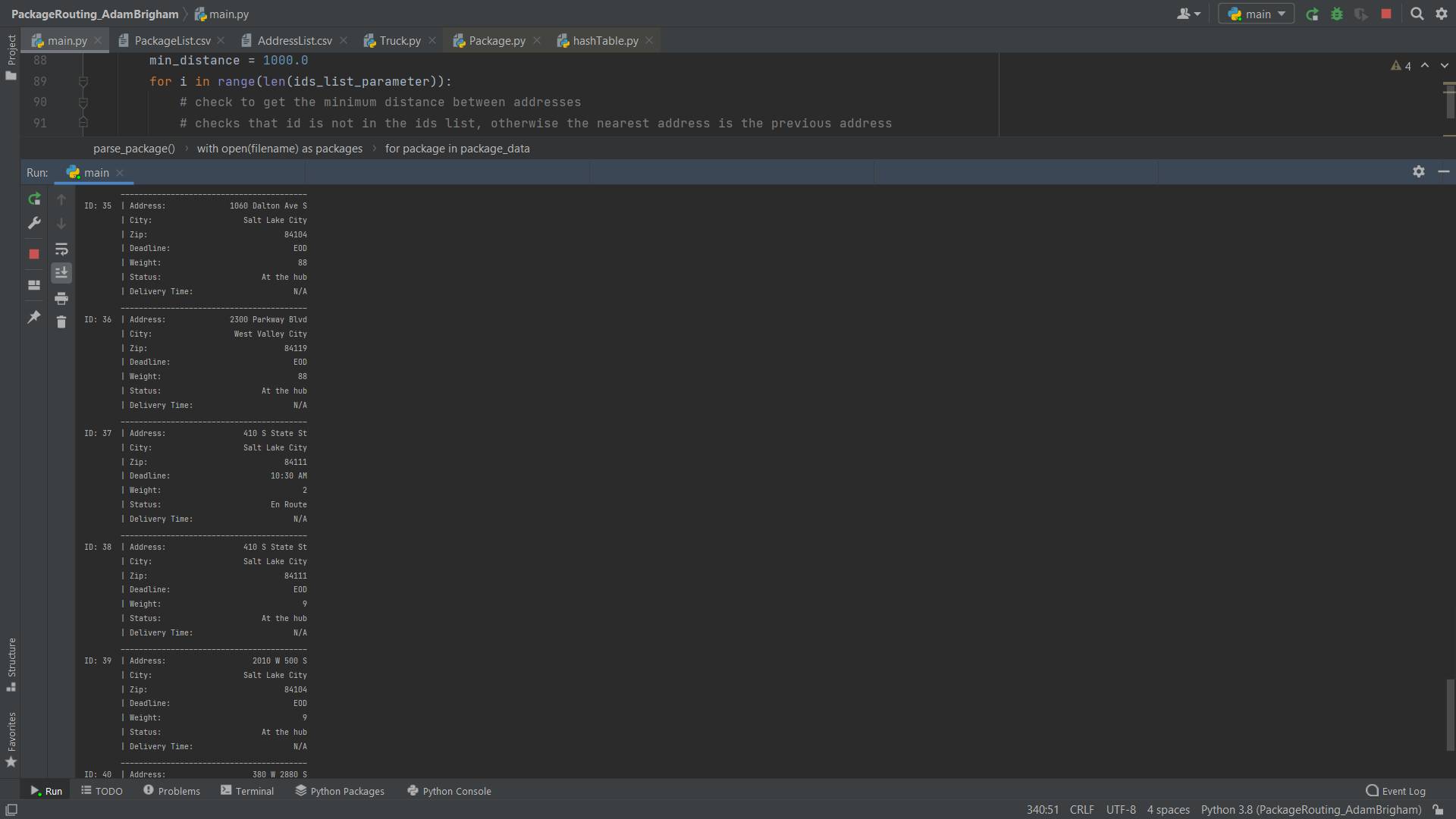


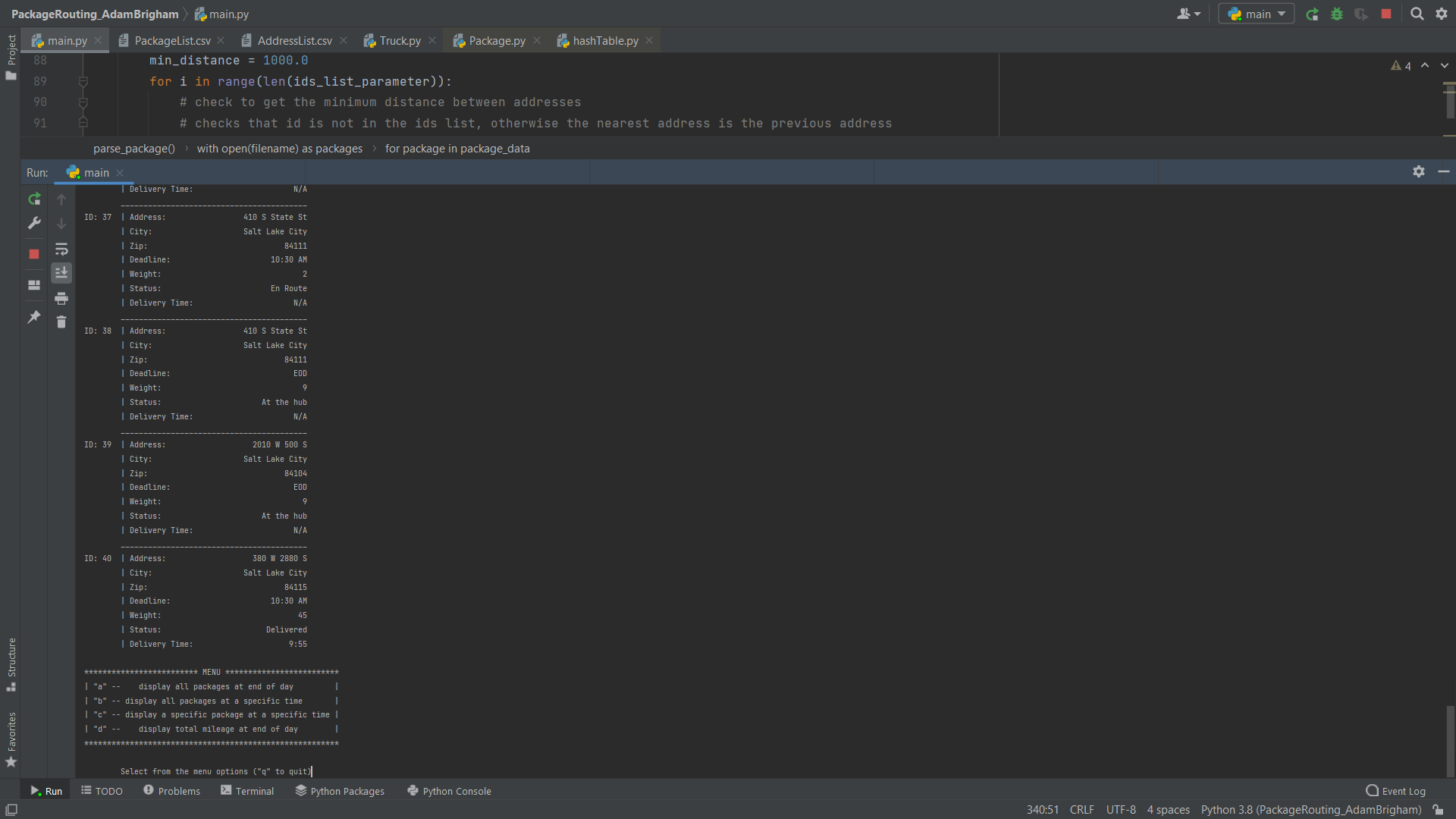




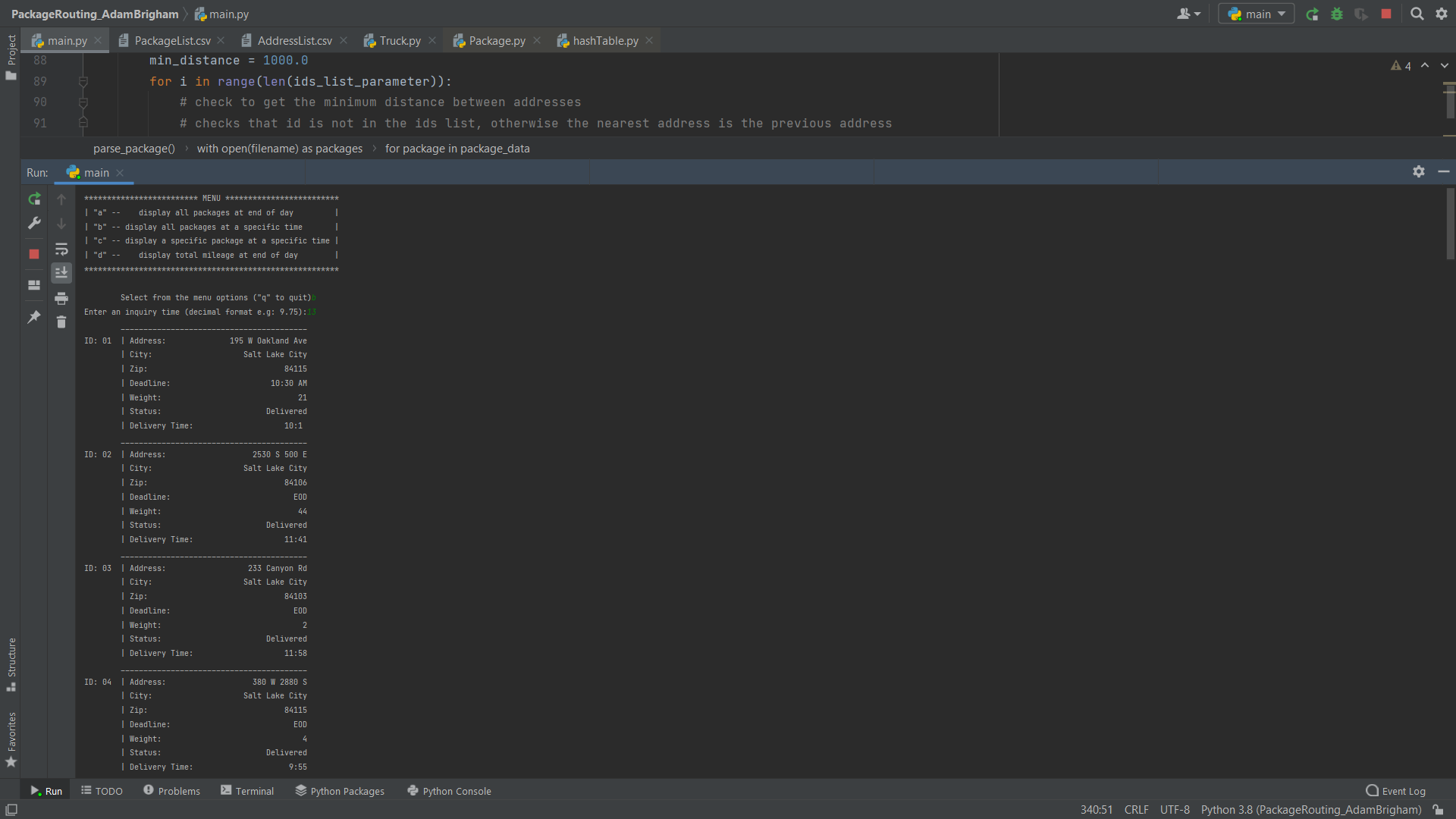


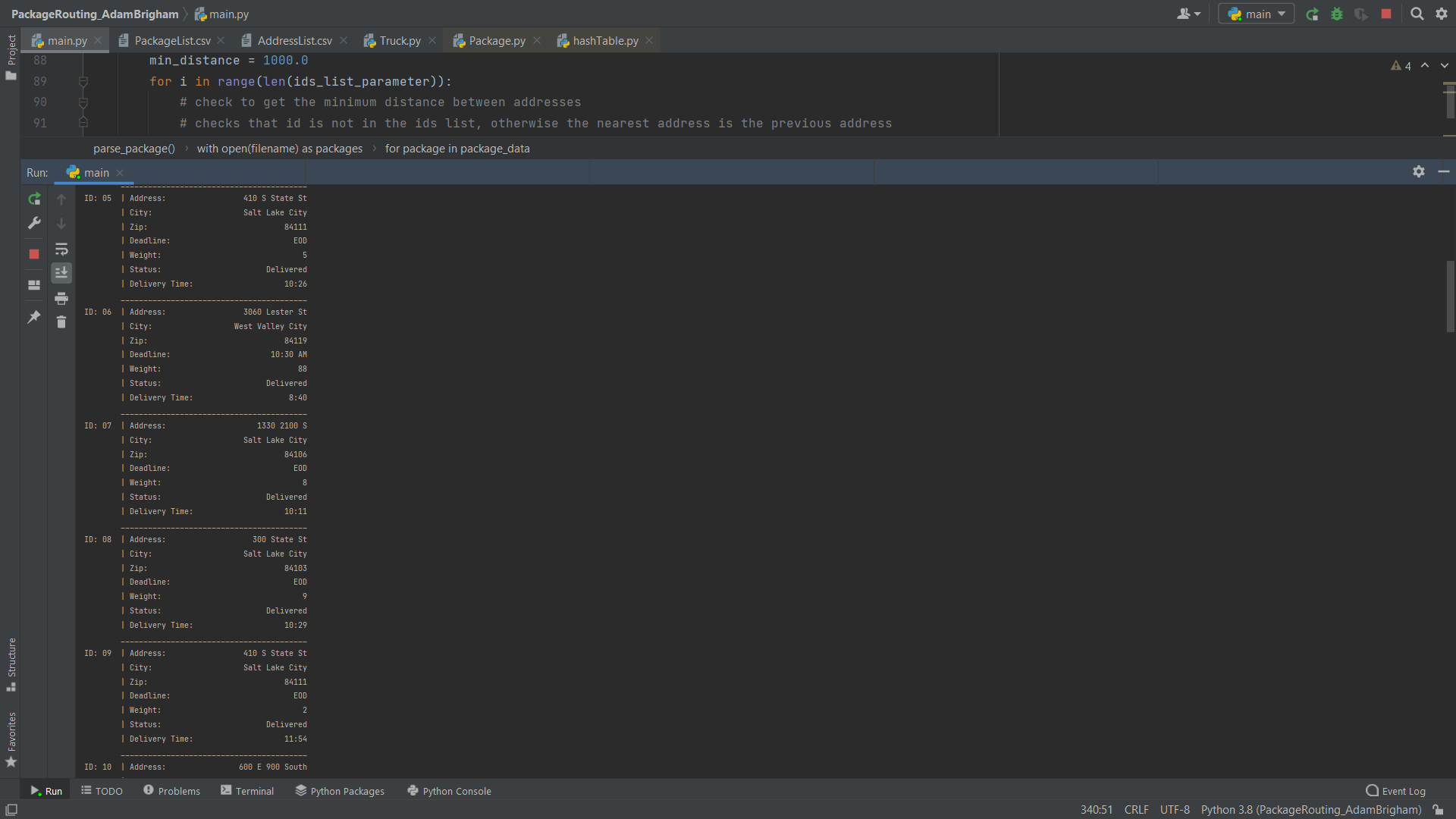


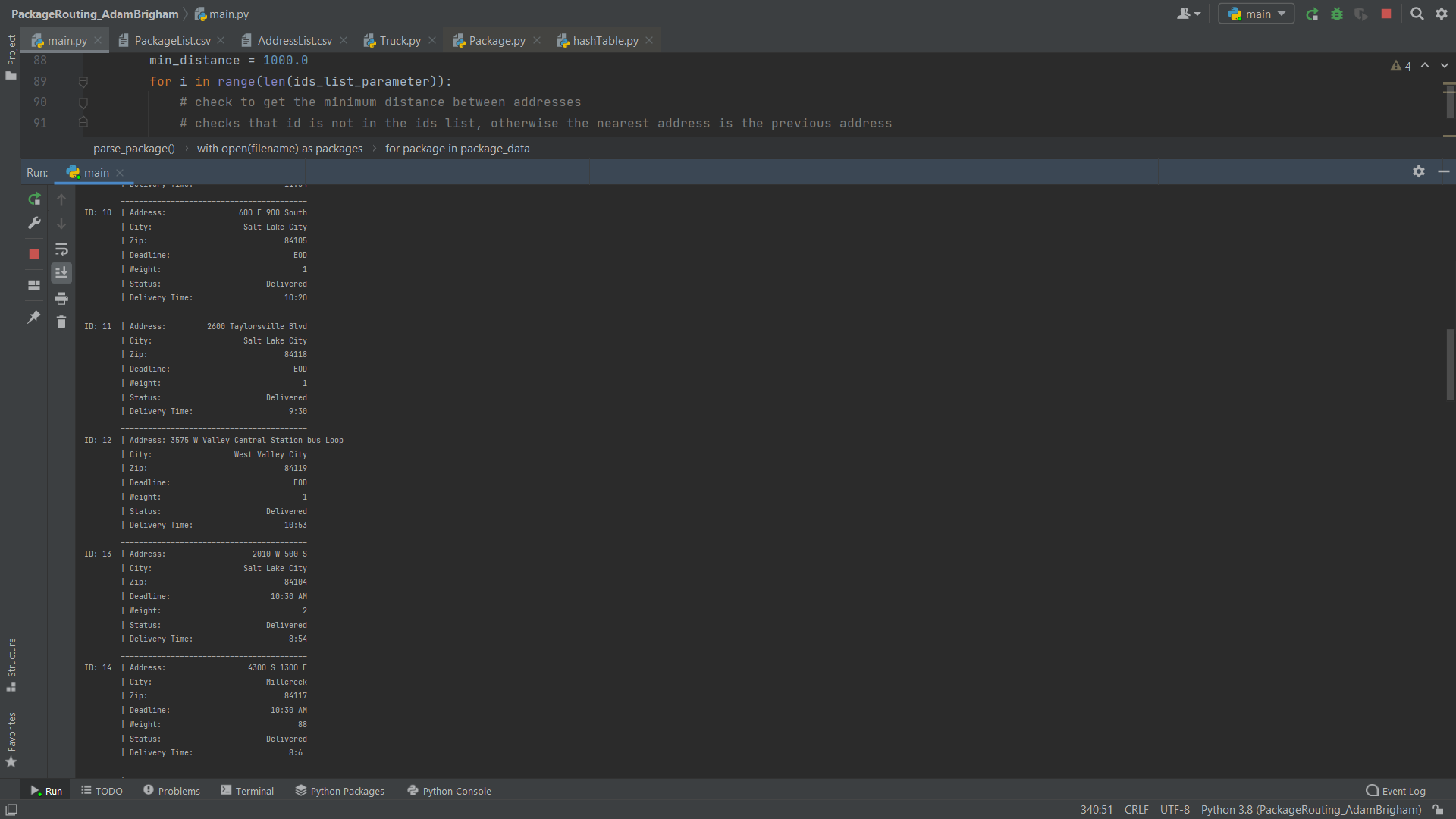


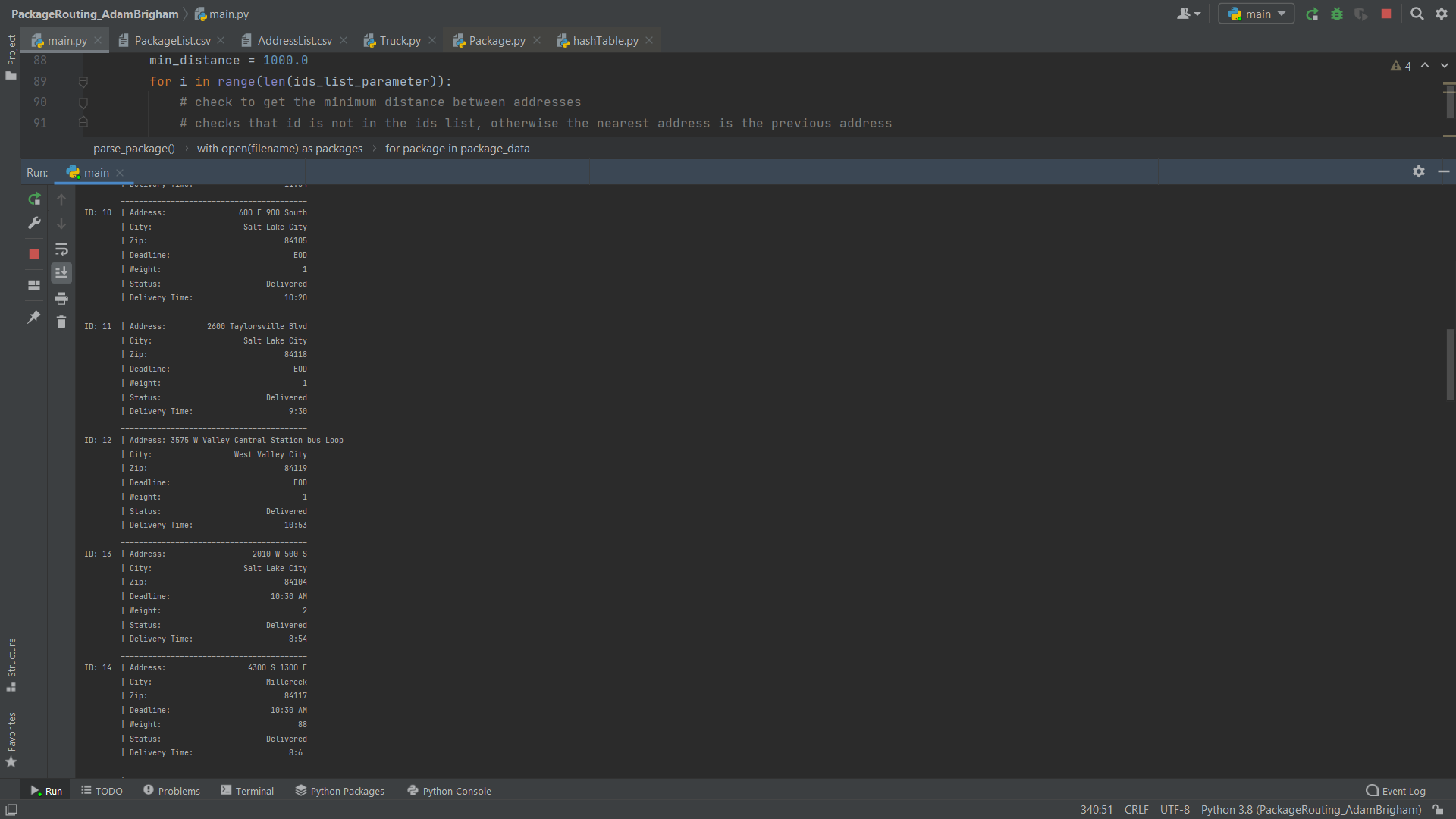


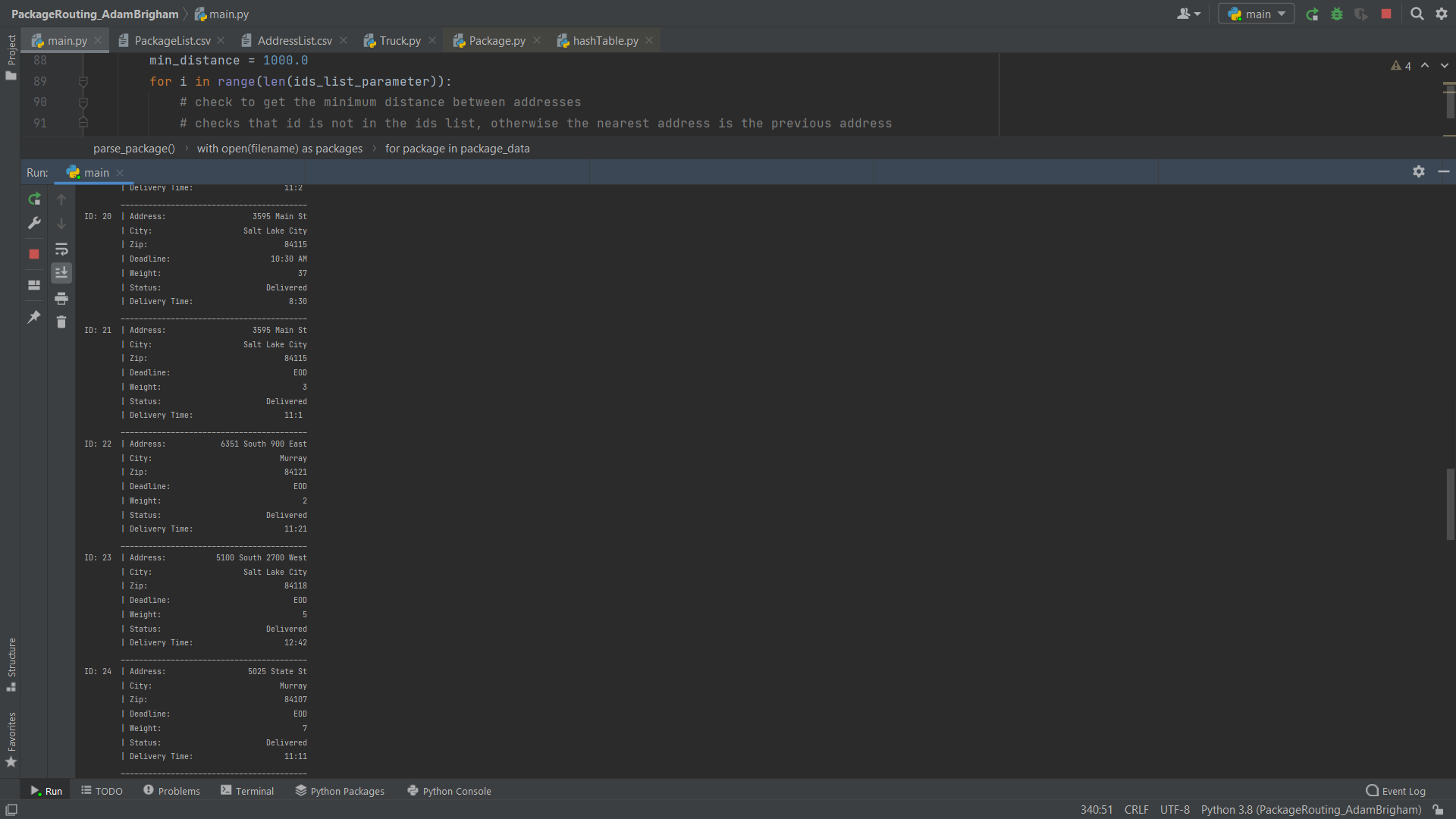
Status check 1:00 pm

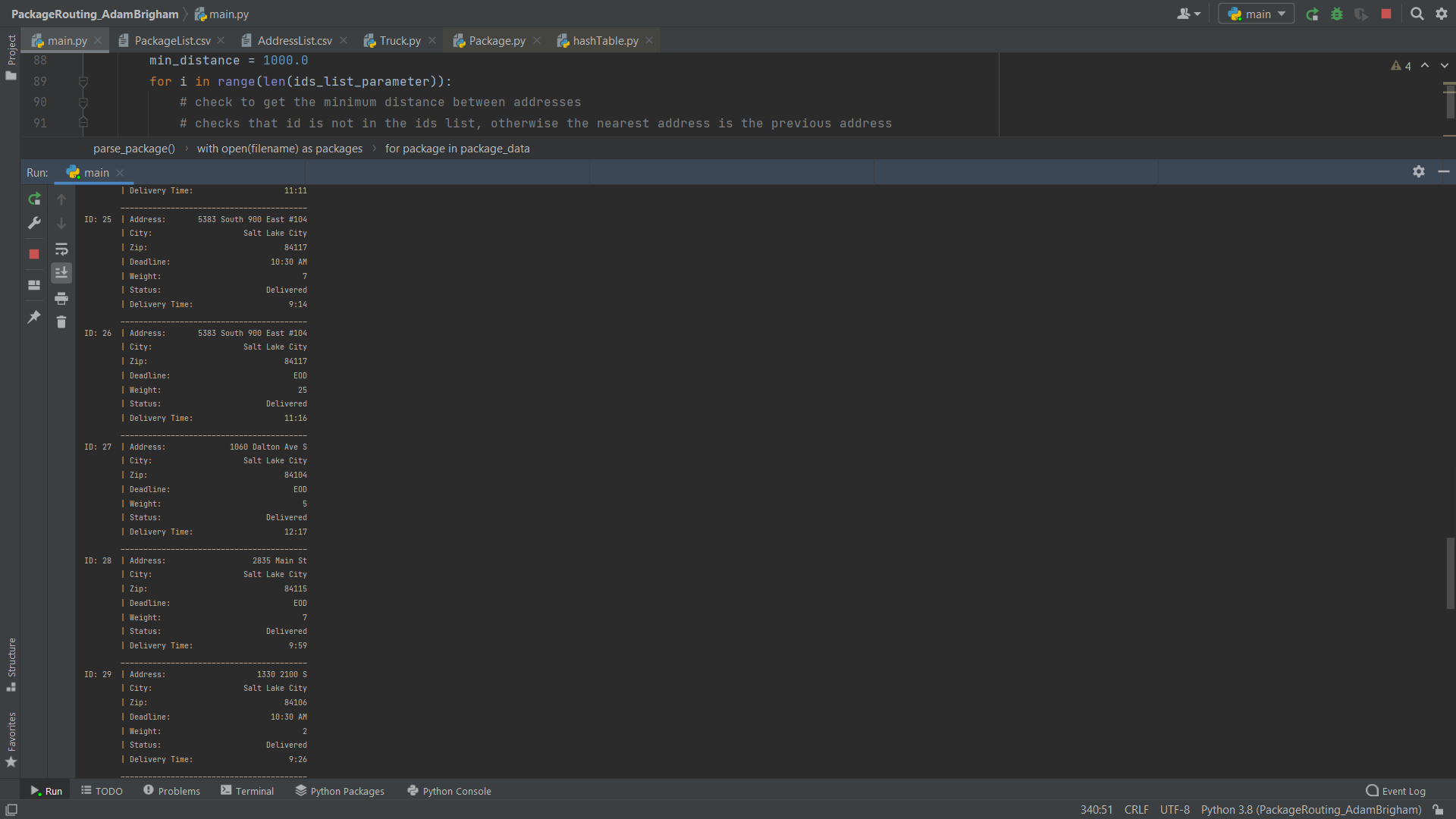


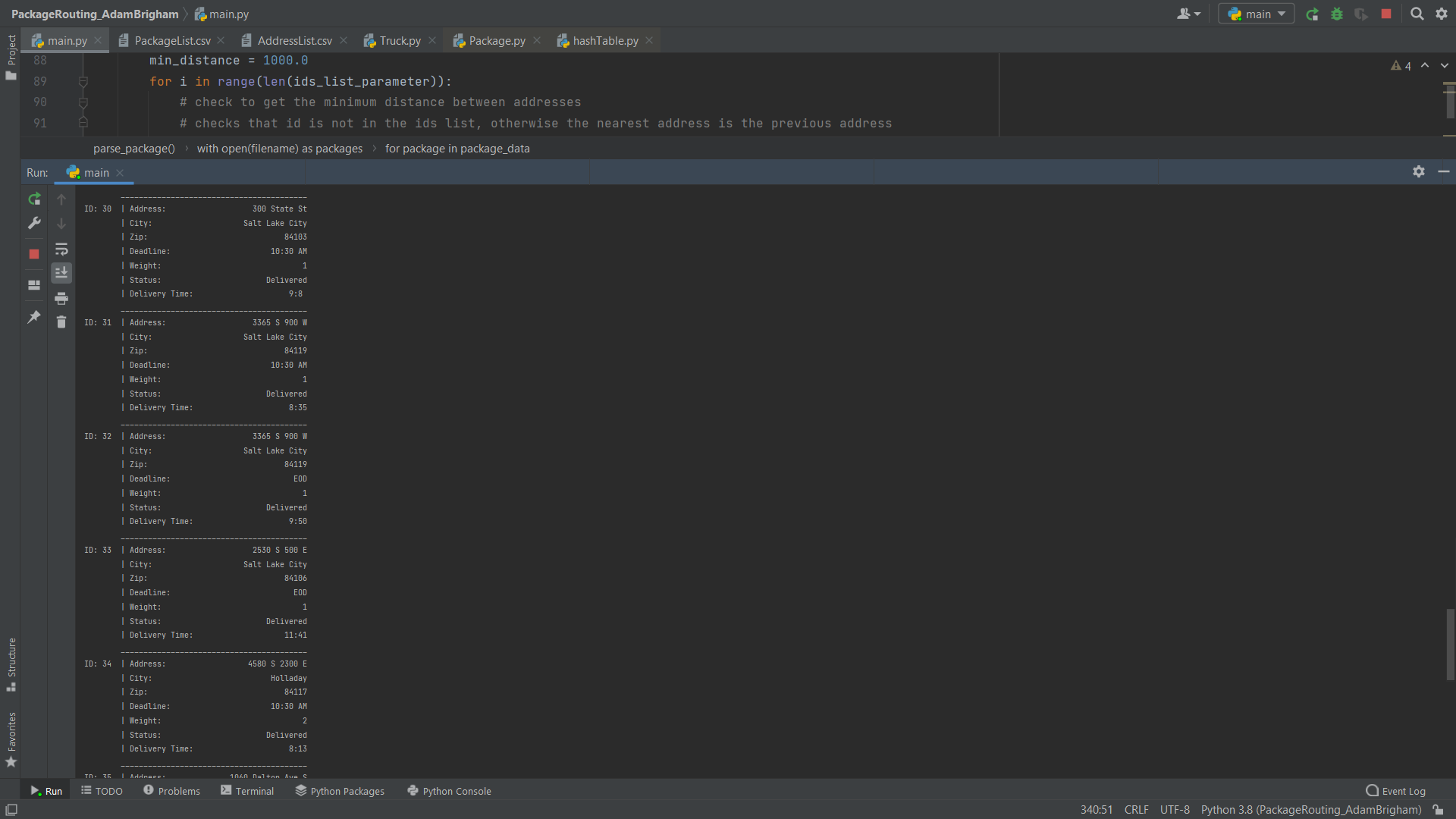


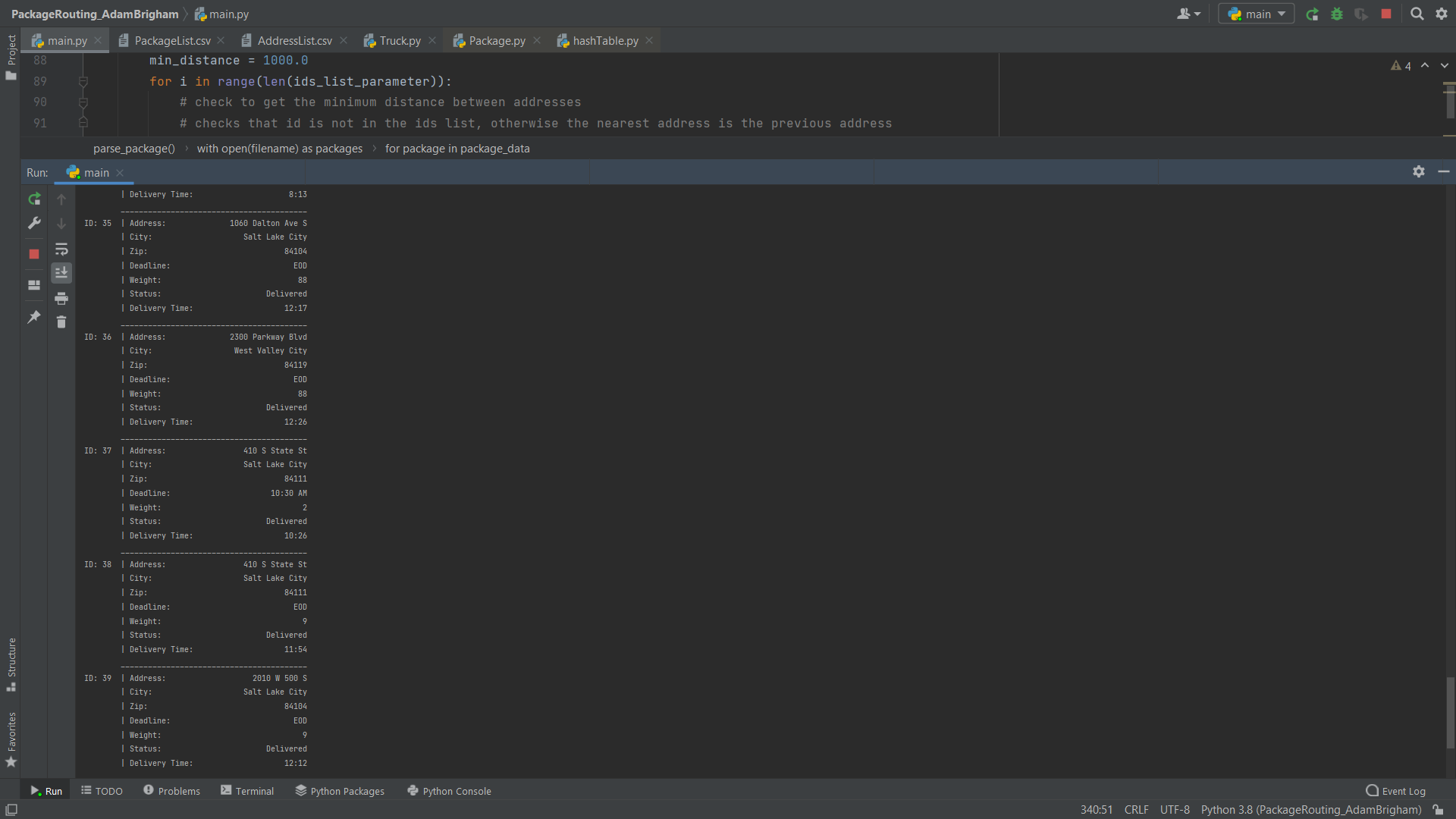


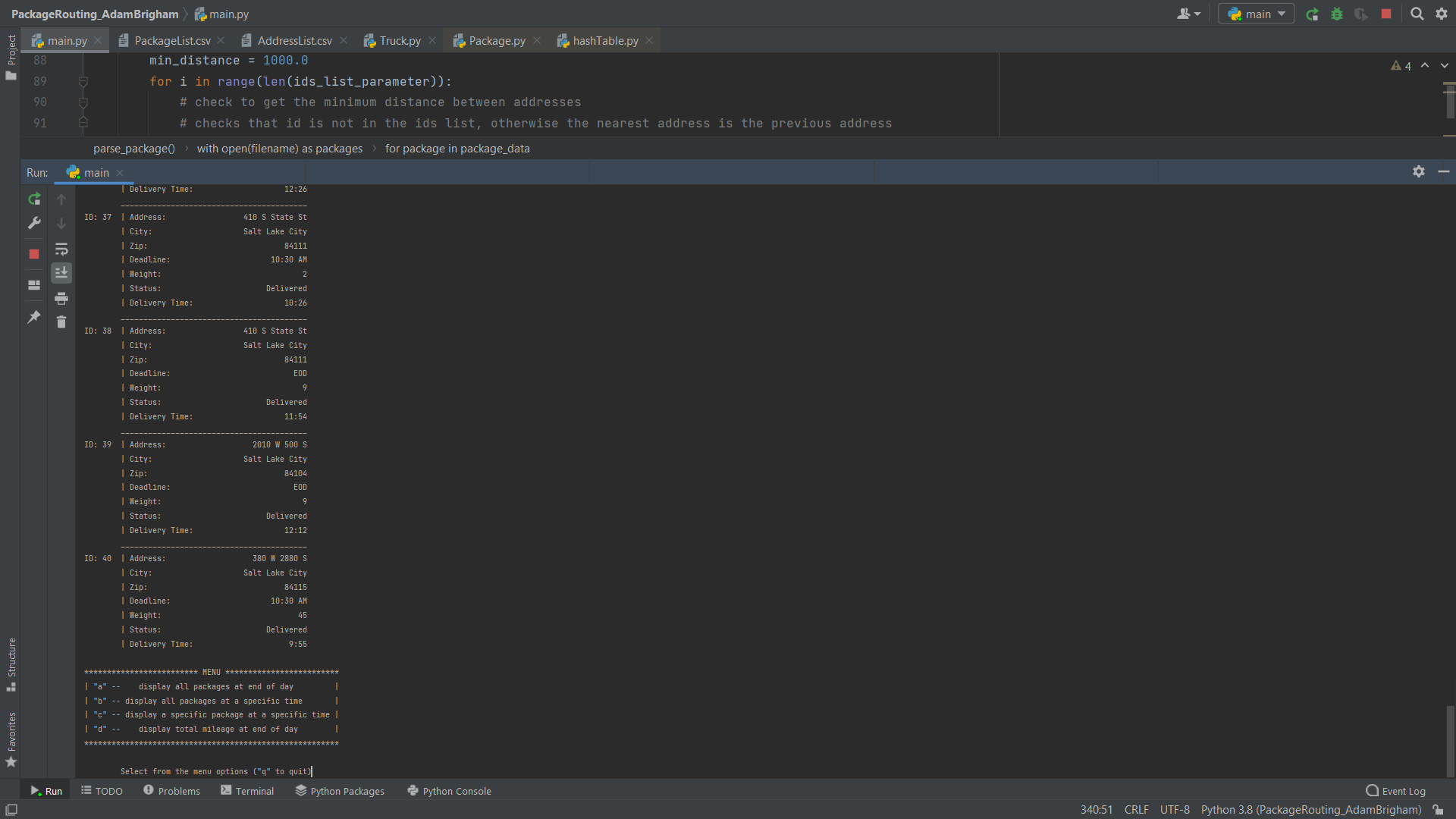












Distances travelled by all trucks / code execution:

