Bilal Sayed

C950: Data Structures & Algorithms II

**CORE ALGORITHM OVERVIEW**

**Stated Problem:**

The purpose of this project is to determine the best route for delivery distribution for the Western Governors University Parcel Service Daily Local Deliveries. The intent is to use this program for this specific location and to use the same program in different cities as WGUPS expands its business. The Salt Lake City DLD route has 3 trucks, 2 drivers, and an average of 40 packages to deliver each day. The task is to write code that determines and presents a solution delivering all 40 packages on time according to their criteria while reducing the total number of miles traveled by the trucks. The solution must include a hash table, self-adjusting heuristic algorithm, and all packages delivered in under 145 miles. Trucks move at a constant speed of 18 mph, can carry a maximum load of 16 packages, and can only be loaded at the hub. The earliest a truck can leave is 8:00 AM. All special instructions must be followed.

**Section 1:**

1. **Algorithm Identification**

The algorithm design chosen was a greedy algorithm. This algorithm will find a heuristic solution and is easily able to adjust to different input sizes. It works by finding the best option at each step. The algorithm does not find the optimal solution but finds a good enough solution with the advantage of saved computing time.

1. **Algorithm Overview**
2. **Algorithm Pseudocode**

function alg(graph, inital)

set nodes to graph.nodes

set edges to graph.edges

set distances equal to graph.distances

set cursor to inital

declare lowest dictionary and add cursor and 0

remove inital from nodes

for key in edges

if the cursor is in the values of the key

remove cursor from values

while continue is true

try

initialize temp dictionary

for edge in edges[cursor]

add the distances of each edge to temp dictionary

select the vertex with the minimum distance in the temp

dictionary and add to the lowest dictionary

set the cursor to the found vertex

for key in edges

if the cursor is in edges[key]

remove the cursor

remove cursor from nodes

except ValueError

set continue to false

add the distance from the final vertex to the hub

set sum\_of\_distances equal to sum of lowest.values()

return sum\_of\_distances, lowest

1. **Programming Models**

The program is written in Python and developed in PyCharm. The program is run on the local machine, so no servers, nor any connect or disconnect sequences are needed. The program does read information from csv files with UTF-8 encoding stored in the project directory.

1. **Space-Time Complexity**

HashTable.py Package.py

|  |  |  |
| --- | --- | --- |
| **Method** | **Line Number** | **Space-Time Complexity** |
| \_\_init\_\_ | 8 | O(1) |
| \_\_setitem\_\_ | 11 | O(1) |
| \_\_getitem\_\_ | 14 | O(1) |
| \_\_iter\_\_ | 17 | O(1) |
| \_\_next\_\_ | 20 | O(1) |
| \_\_str\_\_ | 27 | O(1) |
| \_\_hash | 31 | O(1) |
| set | 37 | O(1) |
| get | 44 | O(1) |

|  |  |  |
| --- | --- | --- |
| **Method** | **Line Number** | **Space-Time Complexity** |
| \_\_init\_\_ | 6 | O(1) |
| \_\_str\_\_ | 20 | O(1) |
| get\_package\_id | 29 | O(1) |
| set\_package\_id | 32 | O(1) |
| get\_address | 35 | O(1) |
| set\_address | 38 | O(1) |
| get\_city | 41 | O(1) |
| set\_city | 44 | O(1) |
| get\_state | 47 | O(1) |
| set\_state | 50 | O(1) |
| get\_zip | 53 | O(1) |
| set\_zip | 56 | O(1) |
| get\_deadline | 59 | O(1) |
| set\_deadline | 62 | O(1) |
| get\_weight | 65 | O(1) |
| set\_weight | 68 | O(1) |
| get\_notes | 71 | O(1) |
| set\_notes | 74 | O(1) |
| get\_truck\_load\_time | 83 | O(1) |
| set\_truck\_load\_time | 86 | O(1) |
| get\_status | 89 | O(1) |
| set\_status | 92 | O(1) |
| get\_delivery\_time | 95 | O(1) |
| set\_delivery\_time | 98 | O(1) |

HashTable.py

|  |  |  |
| --- | --- | --- |
| **Method** | **Line Number** | **Space-Time Complexity** |
| - | 12 | O(n) |

Graph.py

|  |  |  |
| --- | --- | --- |
| **Method** | **Line Number** | **Space-Time Complexity** |
| \_\_init\_\_ | 9 | O(1) |
| add\_node | 15 | O(1) |
| add\_edge | 19 | O(1) |

Greedy.py Truck.py

|  |  |  |
| --- | --- | --- |
| **Method** | **Line Number** | **Space-Time Complexity** |
| alg | 5 | O(n2) |

|  |  |  |
| --- | --- | --- |
| **Method** | **Line Number** | **Space-Time Complexity** |
| \_\_init\_\_ | 15 | O(n) |

Main.py

|  |  |  |
| --- | --- | --- |
| **Method** | **Line Number** | **Space-Time Complexity** |
| main | 9 | O(n2) |

1. **Adaptability and Scalability**

The algorithm’s adaptability to handle a growing number of packages is excellent due to its self-adjusting capability. It can easily scale and handle any number of packages. However, the limiting factor is the space-time complexity because the algorithm time grows at an exponential rate. Just doubling the number of packages increases the time taken by four times.

1. **Efficiency and Maintainability**

Overall, the program as a whole is efficient, with a space-time complexity of O(n2). The program is fairly simple with the most complex part being the algorithm, which itself is still straightforward. It is split up into categorized files which make it very easy to understand when accompanied by the inline comments.

1. **Self-Adjusting Data Structures**

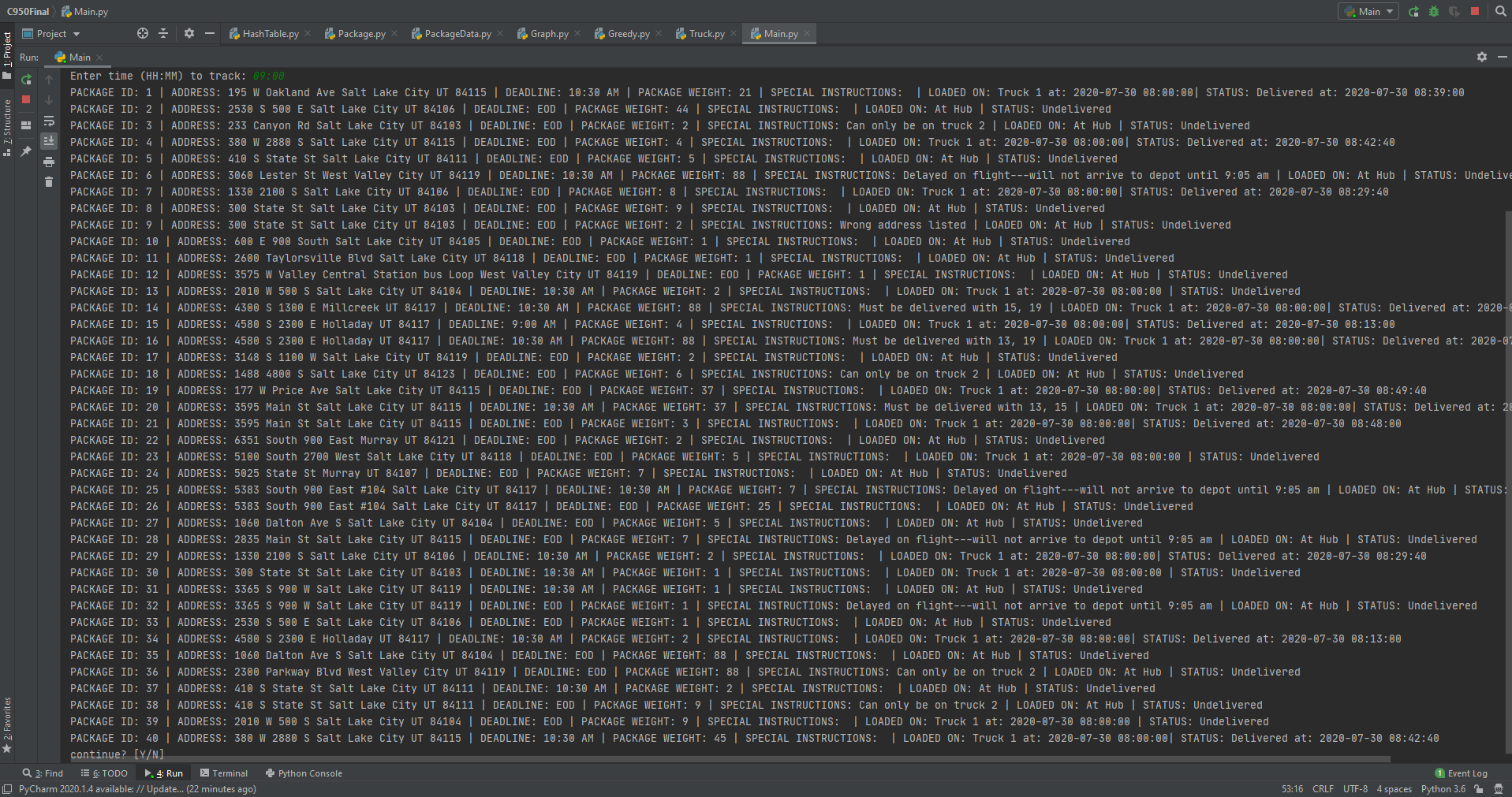
The self-adjusting data structure used is a hash table. It can handle any number of packages. It has methods to set and get items from the hash table. The methods run in constant space-time complexity, so the running time is not affected.

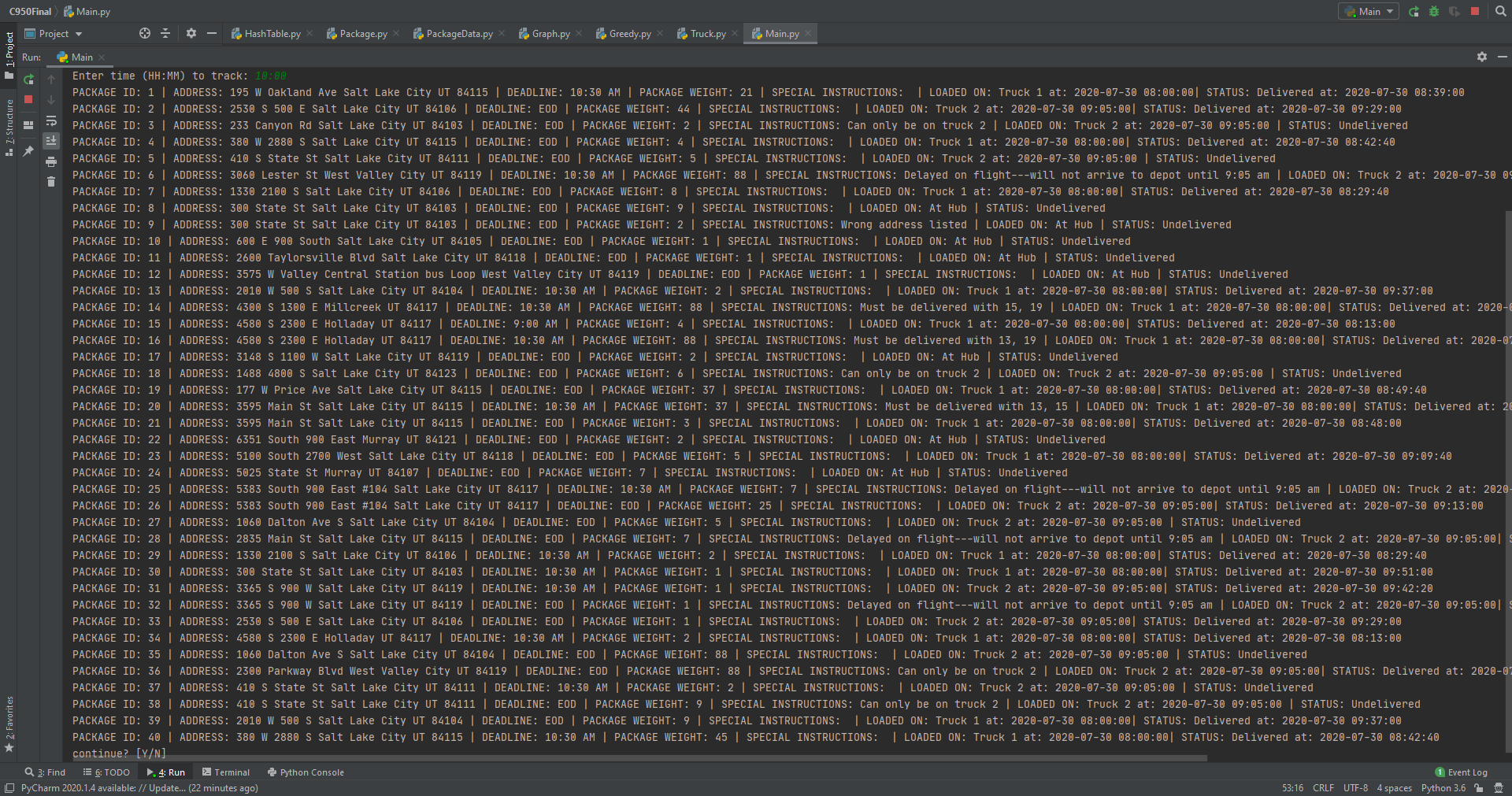
1. **Original Code**
2. See attached code
3. See attached code
4. **Data Structure**
5. **Hash Table** The data structure used to store the package data is a hash table. The package data is first read from the file. All the package data is put into a package object, and then the package id is hashed and added to the hash table. The hash table is direct mapped, so there is no need for buckets or collision mitigation. The hash function works by first getting the length of the array, then getting the modulus of the key(package id) and length of the array. It then returns the index in which the data will be inserted or where it is stored in the array.

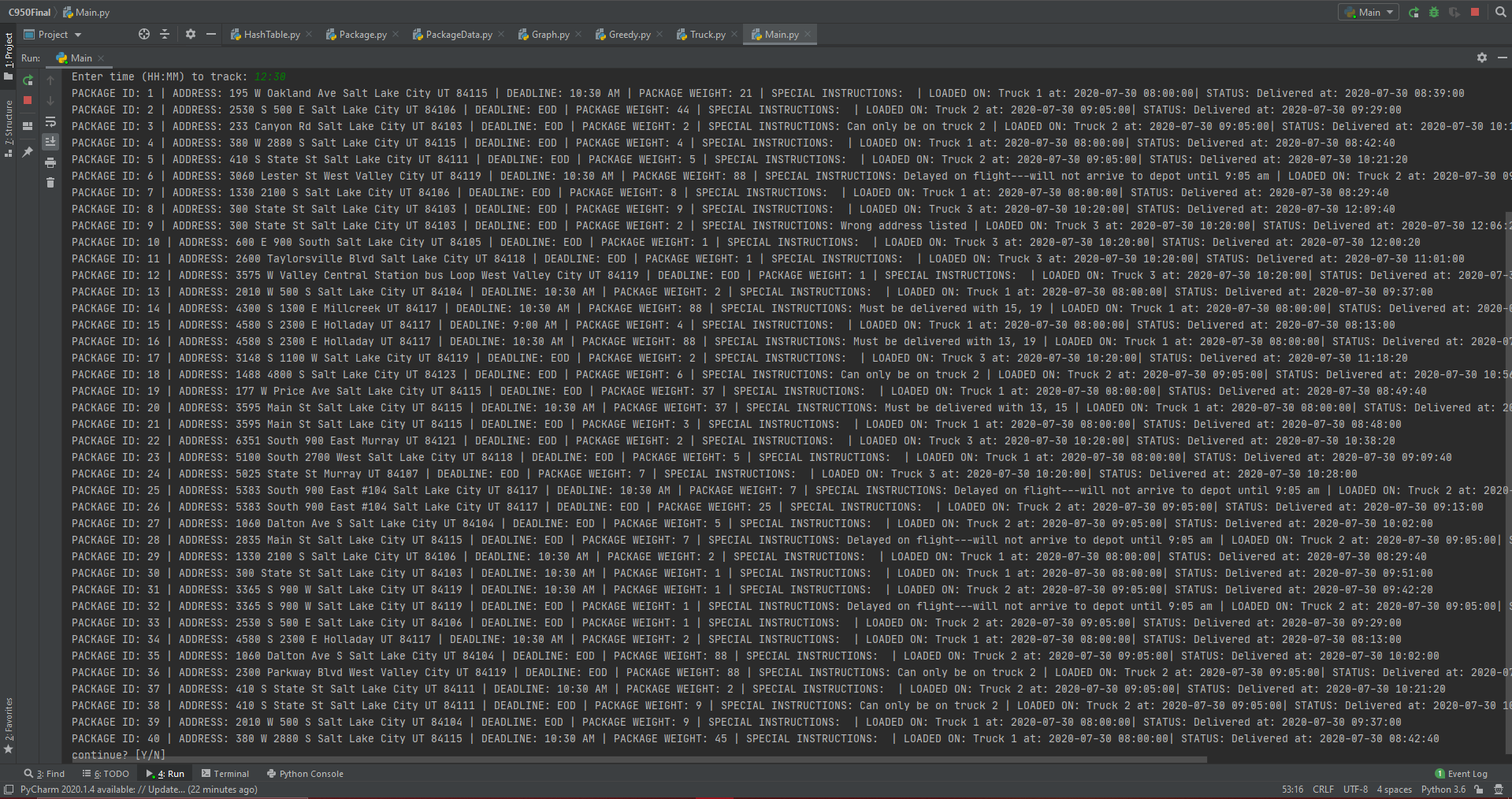
The hash table includes an insertion function (getter) as well as a lookup function (setter). The setter takes a key and a value as input. It then hashes the key and creates an array at the returned index. It then adds the key and value to the nested array.

The getter takes a key as input. It first runs the hash function on the key to find the index at which the value is stored. It then returns the value at the index.

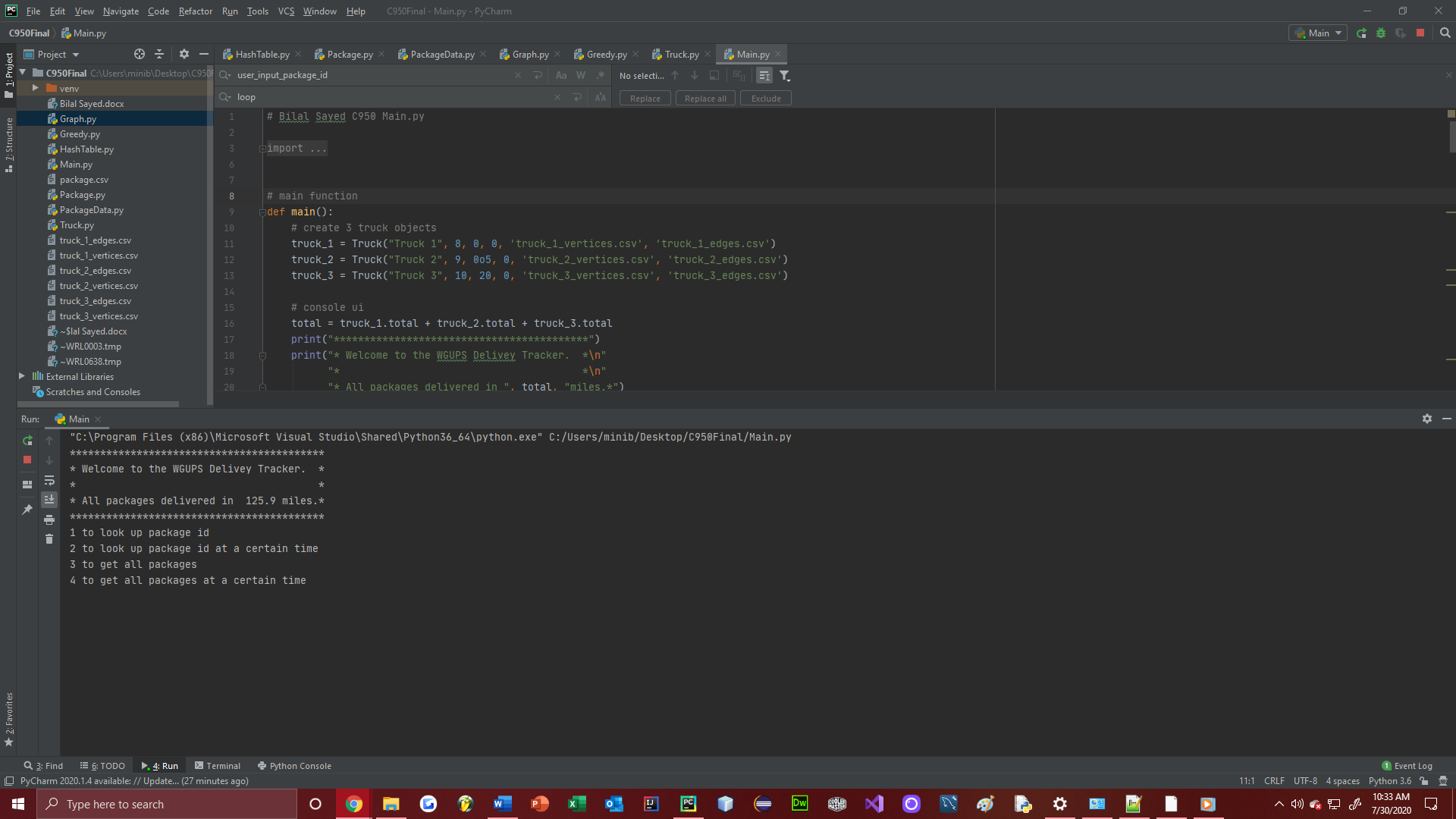
1. **Hash Table**
2. See HashTable.py
3. **Hash Table Look-Up Function**
4. See HashTable.py Line 44
5. **Interface**
6. **Status report at 09:00 am**



1. **Status report at 10:00 am**
2. **Status report at 12:30 pm**



1. **Code Execution**
2. **Code Execution Screenshot**



1. **Strengths of the Greedy algorithm**
2. **Two Strengths of the Chosen Algorithm**

The Greedy Algorithm used has many advantages over an algorithm that might provide a more optimal result. For one, it is much cheaper than doing an exhaustive search for the optimal solution. The Greedy Algorithm finds a heuristic solution while saving computing time. Another advantage of the Greedy Algorithm is that it can scale to any number of packages and still provide a good enough result.

1. **Criteria**

The Greedy Algorithm delivers all packages in 125.9 miles, which is under the 145-mile requirement. The algorithm is also heuristic and self-adjusting. All special instructions are also considered, and packages are delivered on time.

1. **Alternate Algorithms**

The Greedy Algorithm is not the only way to solve this problem. Other algorithms that can be used include the 2-opt algorithm or a branch and bound algorithm.

The 2-opt algorithm is a heuristic algorithm that uses the fact straight edges are shorter than diagonal edges. If 2 edges cross, their 2nd vertices are swapped so that they do not cross, therefore shortening the route. This process is repeated until the route length reaches an optimum, and no more improvements can be made (Henrique Fonseca da Silva Diniz, 2017). It has a worst-case space-time complexity of O(n2), but usually converges quickly, making a fairly popular algorithm. This is similar to the greedy algorithm in the way that it produces a heuristic result. However, the greedy algorithm loops through once and returns a result. The 2-opt, on the other hand, loops through multiple times before reaching an acceptable result. The 2-opt may be applied to the greedy result, perhaps finding an even more optimal solution.

Branch and bound works by setting an upper and lower bound. It then starts exploring every possible solution. However, as it is traversing the tree, if the lower bound is higher than the upper bound, it cancels the search and moves to the next possible solution, as that solution is not worth exploring (Stanford University, (n.d.), slide 3). This will find the optimal solution and could be faster than an exhaustive approach, depending on the search order. The greedy algorithm, on the other hand, finds a heuristic solution and does not explore the data points in a tree like fashion.

1. **Potential Project Revisions**
2. **What I Would Do Differently**

If I were to re-do this project, I would eliminate the hash table and use a graph data structure to hold the package data. I would create a graph class to hold package data. I would then create an adjacency matrix for each package. Next, I would modify the algorithm so that it would accept a graph as a variable and use the data in the graph to loop through the algorithm. This would make it easier to apply algorithms to the data and find the best route. It would also make it easier to access and store data while running the algorithm. The reason for not doing this is due to the hash table requirement.

1. **Choice of Data Structure**
2. **Verification of Data Structure**
   1. The hash table used is very fast, because it uses direct mapping. When looking for a value, there is no need to search through buckets. The hash table uses 2 nested arrays to store the data. It has a hash function, and a getter and setter method. These are used to insert and lookup key-value pairs.
   2. The hash table insert and lookup functions run in O(n2) space-time due to direct mapping, therefore they are constantly fast. Bandwidth and memory are not a concern, as the program runs on a local machine.
   3. Adding more packages, cities, or trucks to the hash table does not affect run time, because of direct mapping and its self-adjusting nature. The lookup and insert methods run at a constant space-time complexity keeping computation time low. To add another city, all that is needed is to create an instance of the hash table for that city and then add the package data. To add trucks, another truck instance can be created, with low impact. Each truck class runs the algorithm on the packages loaded on the truck and has a worst-case space-time complexity of O(n2).

Because of the program’s object-oriented approach, changes in structure are not necessary. To add more packages, cities, or trucks, all that is needed is to create a new instance and add data. To add more packages, all that is needed is to instantiate the hash table with the number of packages. To add more cities, all that is needed is to create a new instance of the hash table and use that to represent the city. To add more trucks, all that is needed is to create a new instance of the truck object and pass in the packages that are on the truck.

1. **Alternate Data Structures**

Other data structures that could be used to store the package data could be a record data type or a graph data structure.

A record stores sub items with a name associated with each sub item (Lysecky & Vahid, 2018*,* 3.2). This would make it easy to store package data because all packages consist of the same fields. This is different from the hash table, which stores data based on a key.

A graph is a data structure for representing connections among items, and consists of vertices connected by edges (Lysecky & Vahid, 2018, 6.1). This would work well with the project because all packages must be delivered to different locations, which could be represented by vertices. The path between the vertices would be the edges and could be weighted with the distances between them. This is very different then a hash table because the graph does not store values base on a key. Searching the graph could also take longer to find the value needed.

Overall, different data structures have different advantages, and a combination of different ones would provide the best result.

1. **Sources**

Henrique Fonseca da Silva Diniz, P. (2017, August 9). Optimization with 2-OPT - Part 1. Retrieved from <http://pedrohfsd.com/2017/08/09/2opt-part1.html>

Stanford University. (n.d.). Branch and bound methods.

Retrieved from <https://web.stanford.edu/class/ee364b/lectures/bb_slides.pdf>

Lysecky, R., & Vahid, F. (2018, June). 6.1 Graphs: Introduction. zyBooks.

Retrieved from <https://learn.zybooks.com/zybook/WGUC950AY20182019/chapter/6/section/1>

Lysecky, R., & Vahid, F. (2018, June). 3.2 Data structures. zyBooks.

Retrieved from <https://learn.zybooks.com/zybook/WGUC9492018/chapter/3/section/2>