WGUPS Routing Program

Section 1 – Programming/Coding

1. Algorithm Selection:

The Greedy algorithm is utilized.

B1. Logic Comments:

Found in comments of “algorithm.py.”

B2. Application of Programming Models:

The scenario looks to find the shortest distance for trucks to deliver packages. Each truck is only allowed to carry a finite number of packages at once, there are varying distances between each address, and some packages have special requirements. I believe the greedy algorithm is the best option for traveling the minimum distance because it can find the next shortest distance at any moment. The program was also built to allow earlier delivery for packages with a deadline before end of day. The program also sorts packages with special loading requirements and loads them on to specific trucks as necessary. The rest are loaded onto the trucks to assure that no truck has more than sixteen packages.

The program was built using the IDE Pycharm Professional 2020.3 from Jet Brains on a late 2013 Macbook Pro.

B3. Space-Time and Big O:

The Big O notation for the program is O(N^2) with evaluations for each individual block of code throughout the files of the program.

B4. Adaptability:

This program can scale and adapt to any city as long as associated distances between addresses in the new city are provided.

B5. Software Efficiency and Maintainability:

Although the shortest path may not always be the most efficient, this program goes a step beyond. Not only does it find the shortest path to travel for the truck, but it also loads the packages with early delivery times on the first truck. It also gives the user the freedom to manually load the packages with special requirements onto specific trucks. As far as maintainability, the program is simple enough to adapt to other environments as long as address location and distance data is provided, and faults can be corrected by ensuring the data given is correct within the csv files. The program is efficient on small scale but must be adapted to a faster runtime complexity if it is to be fed large amounts of package or address data.

B6. Self-Adjusting Data Structures:

A nested dictionary type was used as its capable of holding all of the package information with key value pairs. A nested dictionary is a simple but powerful data structure. Lists and dictionaries are forms of data structures but nesting such containers provides a programmer with much more flexibility in the way that data can be organized. Another benefit is that the code tends to be more readable. A weakness of nested dictionaries is that it can be clumsy to search and manipulate subsequent contents. Nested dictionary keys must also be unique and immutable but that’s not an issue for this program (Lysecky & Vahid, 2018.)

C1. Identification of Information:

Found in the first two lines of ‘main.py.’

C2. Process and Flow Comments:

Found throughout all of the files within the program.

D. Data Structure:

The nested dictionary is great for storing data. All of the package data was stored in the nested dictionary, making it easy to retrieve the data since all important information was stored in a key value pair (Lysecky & Vahid, 2018.)

D1. Explanation of Data Structure:

The package data is stored within a data structure called a nested dictionary. They are sorted in the table, using the package ID as a key, with other important package information linked to a key value pair within the nested portion of the dictionary. These package ID keys are unique to every package so there will be no collisions.

E. Hash Table:

The hash table created is a nested dictionary found in “packages.py.” It uses a nested dictionary with the package ID as the key and the corresponding row as the value within the csv file.

F. Look-Up Function:

(Lines 107-114 in “packages.py” and called in “main.py”) Lets the user input the package ID and find the information involving the associated package.

G. Interface:

Screenshots are located within the screenshots folder of the program as a pdf from Pycharm.

H. Screenshots of Code Execution:

Screenshots are located within the screenshots folder of the program as a pdf from Pycharm.

Section 2 – Annotations

I1. Strengths of the Chosen Algorithm:

The greedy algorithm is simple to implement and easily scalable (Lysecky & Vahid, 2018.) Because it only cares about the shortest route, using it is easy. Inserting a new address and the distances to and from this address are all that’s necessary in regard to the algorithm’s computations.

I2. Verification of Algorithm:

With the greedy algorithm, the program can choose packages with short delivery distances. It can also prioritize packages with special requirements and early delivery times. The greedy algorithm meets all requirements for the scenario.

I3. Other Possible Algorithms:

Dijkstra’s Shortest Path and Bellman Ford’s algorithm are two other algorithms that could have been utilized.

I3a. Algorithm Differences:

Dijkstra’s Shortest Path is very similar to the greedy algorithm used in this program; however, Dijkstra’s also considers future destinations rather than the greedy algorithm’s best choice at the moment. Dijkstra’s Shortest Path starts at the ending destination and finds the shortest path between vertexes until it reaches the starting point, essentially working backwards (Lysecky & Vahid, 2018.)

Bellman Ford’s algorithm uses a dynamic programming approach to find the shortest path by utilizing a graph and a source vertex (Geeks for Geeks, 2020.) Because it can also account for negative values in a graph with weighted paths, it is different from the greedy algorithm (Geeks for Geeks, 2020.)

J. Different Approach:

If I were to redo this project, I would most likely use Dijkstra’s shortest path. Not only because it appears to be similar to the greedy algorithm’s shortest path approach, but because it also takes into account future paths. This would seem to be more efficient. Other changes would be the utilization of more getters and setters. It would reduce clutter.

K1. Verification of Data Structure:

I used Nested Dictionaries as the main data structure to host the Distance Table and all of the Package Information.

K1a. Efficiency:

The package information was stored with the package ID as a key with the nested portion of the package key containing key value pairs for all subsequent data (keys = type of data, values = specific data for that package.) The distance table was utilized in a similar manner with the location and destination being a key and the distance being the value. This information could be called easily to find the distances between two points.

K1b. Overhead:

The programs runtime complexity is O(N^2) so as the program grows in scale it will eventually become too slow to be considered efficient. As the company stands on a smaller scale currently, the overhead is manageable but if the company were to ever grow in massive scale, the program would need to be changed to accommodate.

K1c. Implications:

New package data or changing locations would not be a problem. All that is required to adding a new address is the distances between other addresses and the algorithm can take it from there. Although it has been stated, as the runtime complexity is O(N^2), as more data is added overtime the slower the program will become.

K2. Other Data Structures:

Balanced trees and graphs are two other data structures that could be used to meet all requirements.

K2a. Data Structure Differences:

Graphs focus on how data points are connected to each other while hash tables are only concerned on presenting the data as vertices that the algorithm can utilize (Lysecky & Vahid, 2018.)

Balanced trees work around the connection between data points in the memory much like a graph (Lysecky & Vahid, 2018.) The algorithm can utilize them to find the most efficient route to a data point.

L. Sources:

Lysecky, R. and Vahid F. (2018). *C950: Data Structures and Algorithms II.* Zybooks.

Geeks for Geeks. (2020, April 23). *Bellman Ford’s Algorithm* | *DP 23.* https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/.

M. Professional Communication:

Comments are spread throughout the program but some things to note are that the time is based on a 24-hour clock. When the program begins it will automatically display the number of miles it will take to deliver all packages. A prompt will also appear that gives the user the option to display package information or input a time to see all package whereabout at that current time. The user may also choose the third option to exit the program. The end result came out to be 91.0 miles.

|  |
| --- |
|  |
|  |