C950 WGUPS

David Sainvilus

ID #001049272

WGU Email: dsainvi@wgu.edu

10/10/2021

C950 Data Structures and Algorithms II

# Introduction

The purpose of this project is to determine the best route and delivery distribution for the Western Governors University Parcel Service (WGUPS) using a common high-level programming language Python. 40 packages must be split up across three trucks with two different drivers. Some packages have delivery constraints, and some have been delayed. There’s a package that has the wrong packages and address it must go with the second delivery truck. For this problem, I must use conditional commands to load the trucks given the current data that is provided. Then I must use the greedy algorithm to optimize delivery of each package on a truck during their route. This article will analyze the use of the greedy algorithm and provide a descriptive overview of the software methods.

# Algorithm Identification

The Greedy approach it determines the shortest available path from its current location then continues to do this until no additional packages remain.

It uses the recursive technique in order to determine the location with the shortest distance to visit next based on the current location.

The greedy algorithm contains three parameters. A list of packages, the number of trucks, and the current location of the truck the first for loop finds the shortest distance to the next location. A value will continually change until a minimum value is found.

Then the for loop determines what happens when the value with the lowest number has been determined. Conditionally statements check which truck the package is associated with. The values are added to the proper truck list. The package is then removed off the list, the location moves to the next minimal distance location determined from the original loop.

Finally, the recursive call asks for the next location and the list is shortened. The recursive calls will continue until the base case is called, this will end the loops and return an empty list.

# Logic Comments Greedy Algorithm

def get\_shortest\_route(\_list, num, curr\_location):

if not len(\_list):

return \_list

lowest\_value = 50.0

location = 0

for i in \_list:

value = int(i[1])

if get\_current\_distance(curr\_location, value) <= lowest\_value:

lowest\_value = get\_current\_distance(

curr\_location, value)

location = value

for i in \_list:

if get\_current\_distance(curr\_location, int(i[1])) == lowest\_value:

if num == 1:

first\_truck.append(i)

first\_truck\_indices.append(i[1])

\_list.pop(\_list.index(i))

curr\_location = location

get\_shortest\_route(\_list, 1, curr\_location)

elif num == 2:

second\_truck.append(i)

second\_truck\_indices.append(i[1])

\_list.pop(\_list.index(i))

curr\_location = location

get\_shortest\_route(\_list, 2, curr\_location)

elif num == 3:

third\_truck.append(i)

third\_truck\_indices.append(i[1])

\_list.pop(\_list.index(i))

curr\_location = location

get\_shortest\_route(\_list, 3, curr\_location)

# Insert 0 for first index of each index list

first\_truck\_indices.insert(0, '0')

second\_truck\_indices.insert(0, '0')

third\_truck\_indices.insert(0, '0')

# Development Environment

PyCharm and Python 3.7

# Space-Time and Big-O

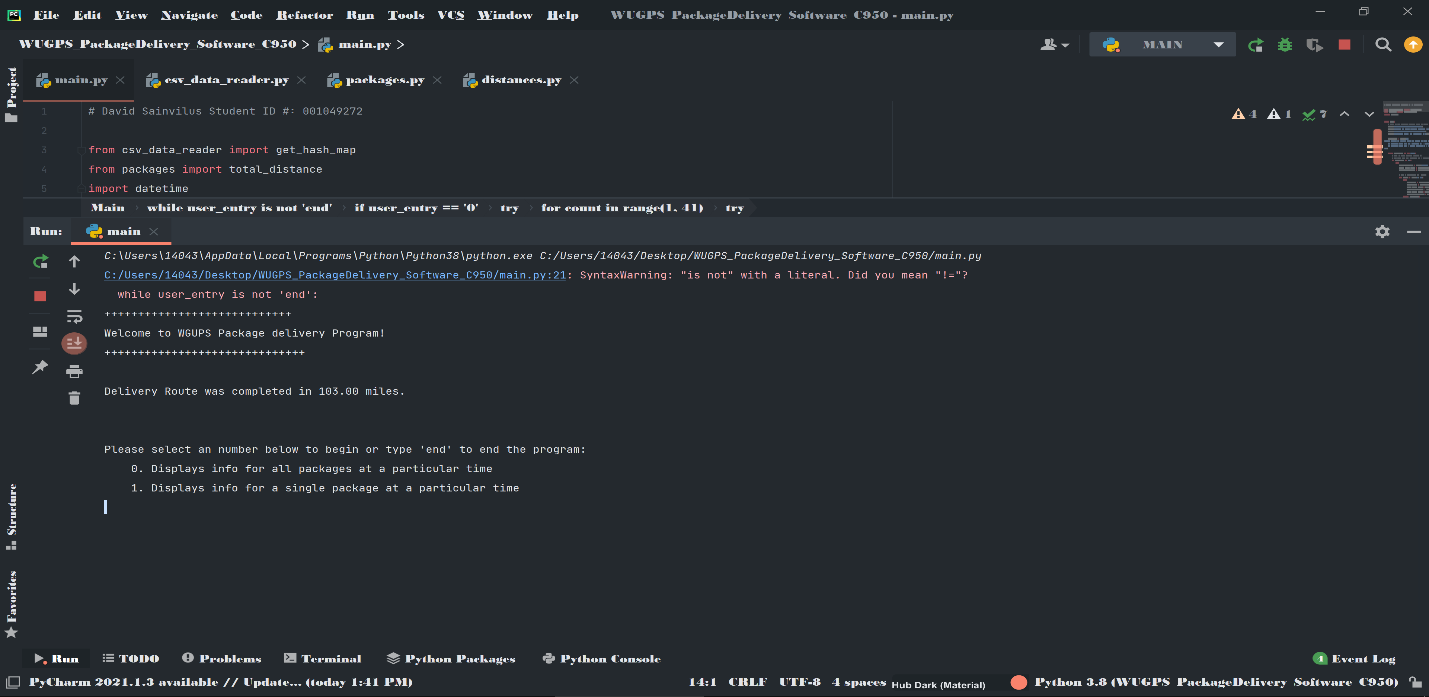
O(n^2) Memory and computational time remain nearly linear throughout the entire application. This allows the available set of inputs to scale without being overburdened by memory availability constraints. Bandwidth is not a factor in the current implementation as the application is run and managed on a local machine that does not require network resources. B4. Scalability and Adaptability

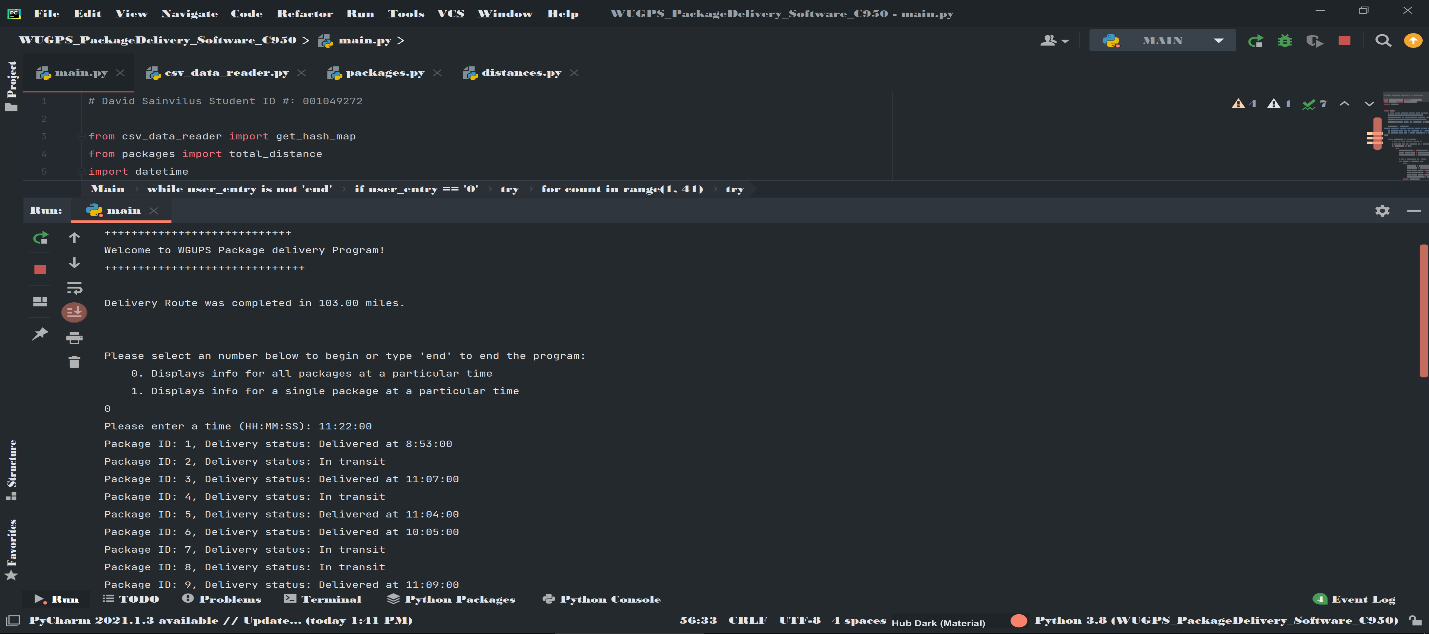
The core functions of the application are designed to be able to scale and addresses changes in the number of packages, the number of trucks, and the number of locations. Minor changes are required to scale with any of these components. For example, another set of location CSV files can be inputted into the application to calculate a new route and determine an optimal path. Additional packages can be inserted, and the program will determine where to place the packages. This approach also allows a great deal of control when implementing numerous sub-applications as the design allows the input set to change freely. A potential problem with future scaling in this application is the approach I take to load the packages into the truck. It is currently done manually to meet all the package constraints and deadlines. Given the opportunity to improve this project, I would instead work on developing a heuristic approach to determining what packages go onto which truck. Having this process automated would benefit both the ability for the application to adapt to new environments and potentially provide a better path set for the greedy algorithm to work with. However, the core functions of the application have great potential to scale with changing markets as the greedy algorithm handles limited data sets very quickly.

# Data Structure

The primary data structure I implemented throughout the application is a list of lists. I choose this data structure because it is flexible and easy to work with. It also works well with a hash table and data retrieval. Additionally, the project required to be able to have quick access to many elements of a package. Using a list allowed me to create a hash table with chaining resulting in very fast lookup, insertion, deletion, and access to specific data elements. Most operations involving the data structure had a constant time complexity of O(1). The biggest advantage for me was it allowed me to verify the correctness of my trucks and packages along the route with very little additional overhead.

# Screenshots of Code Execution





# Strengths of Chosen Algorithm

The greedy algorithm preforms all required functions given the project constraints and delivers the packages 100 miles. The biggest strength of this algorithm is its ability to quickly find the optimal path for a given truck (defined above). Another huge advantage of the greedy algorithm approach is that it can scale with any set of data and addresses provided to it. Additionally, the user interface provides the ability for a user to see all package details and their delivery status at a given time. It can also search for an individual package based on its package ID.

# Other possible Algorithms

Another algorithmic approach I could have used to optimize the packages was a dynamic programming approach. “Dynamic programming is a problem-solving technique that splits a problem into smaller subproblems” (Zybooks, 3.5). I use an exhaustive approach instead of trying to break the program down into smaller functions. The advantage of using a dynamic approach is that I can store paths along a route and check if there is a faster way to get to a location by first traveling to another location. Storing these different paths may have created a large space complexity but it could have yielded a shorter path

# Different Approach

Another approach I could have implemented for the optimal route is a self-adjusting heuristic. This approach would start at the hub then determine the closest path to the hub. It would then determine all packages that need to be delivered to that truck and load them into the truck. From there it would start at the new location and determine the shortest path from that location. If a location was already visited by that truck than it would move on the to the next closest location until all 40 packages had been allocated to a truck and a path was set. My approach and the self-adjusting heuristic share a similar attribute in the sense that the shortest available path is always chosen.

# Verification of Data Structure

The primary data structure I implemented throughout the application is a list of lists. I choose this data structure because it is flexible and easy to work with. It also works well with a hash table and data retrieval. Additionally, the project required to be able to have quick access to many elements of a package. Using a list allowed me to create a hash table with chaining resulting in very fast lookup, insertion, deletion, and access to specific data elements. In fact, most operations involving the data structure had a constant time complexity of O(1). The largest advantage was the ability to verify the correctness of my trucks and packages along the route with very little additional overhead. One weakness I identified while working through this project using a list of lists is that it is hard to utilize the lists in an object-oriented way. For example, I had to apply the same operations to each list in order to deliver the packages for a given truck. Having a package factory class and a truck object would have simplified the coding and made it easier to manage scaling.

# Efficiency

The software is extremely efficient, with two comparisons that have a time efficiency of O(n^2). While this is not the best time complexity, it scales well for the 16-package limit per truck. It is also very maintainable as much of the software is the same core functions that have been modified for use case. Debugging is easier because you can always refer to another instance of the function to determine where potential errors might be.

# K2. Other Data Structures

One weakness I identified while working through this project using a list of lists is that it is hard to utilize the lists in an object-oriented way. For example, I had to apply the same operations to each list to deliver the packages for a given truck. Having a package factory class and a truck object would have simplified the coding and made it easier to manage scaling.

# Data Structure Differences

I could have used many different data structures to fit the requirements of this project. One popular data structure would be a binary search tree. The BST would be different in the sense that I can presort the packages based on an attribute and quickly access them through a tree. Another data structure I could have implemented was a graph. A huge advantage of using a graph is that I could have grouped similar packages together as adjacent vertices. Then I could have traversed the graphs until a maximum traversal length of 16 was reached. This would also be a good method for future scaling.

# Sources - Works Cited

The primary resource I used when modeling this overview was the reading material provided by Zybooks. The quote in the section, **Advantages of chosen Algorithm** is cited below:

Learn.zybooks.com. (n.d.). *zyBooks*. [online] Available at: https://learn.zybooks.com/zybook/WGUC950AY20182019/chapter/3/section/5 [Accessed 10 Nov. 2019].

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

<https://learn.zybooks.com/zybook/WGUC950AY20182019/>