**WGU C950 PA: WGUPS Routing Program**

Nikolas Butalid

Western Governors University

C950: Data Structures and Algorithms II

**Introduction**

This document contains information pertaining to the Performance Assessment (PA) for WGU C950: Data Structures and Algorithms II. A program written in Python was developed to implement an algorithm that calculates an efficient route for the delivery of 40 packages. The project was designed according to the assumptions and restraints given in the PA Task Overview.

**Development Environment**

The program was written using JetBrain’s PyCharm version 2021.1.3. It was written using Python 3.9. The device the program was developed and tested on was a Dell G5 Laptop running Windows 10. No external libraries were used during the development of this project.

**Algorithm Identification**

The algorithm utilized in this project was the “Nearest Neighbor Algorithm”. The algorithm uses a weighted adjacency matrix to represent the distances between different locations. The overall program’s time complexity is at worst is O(n3 ) and the space complexity is O(n2). More detailed information about the program’s space-time complexity can be found in the comments of the code.

**General steps for Nearest Neighbor Algorithm** (TLMaths, 2014)

1. Create a list of all vertices to be visited and called it ‘unvisited’ or ‘u’
2. Start at a vertex
3. Find the edge with the smallest value in the unvisited list connected to the current vertex. Call the connecting vertex ‘v’
4. Remove the current vertex and ‘v’ from the unvisited list. Record the edge’s value
5. Repeat steps 2 - 4 until the unvisited list is empty
6. Add up all the values together and add the edge from the last ‘v’ to the original vertex

**Pseudocode for Nearest Neighbors Algorithm**

while there are unvisited addresses in a list:

min\_dist = first address in unvisited list

for j in unvisited list:

if equal to min\_dist:

continue through loop

\*if distance between starting address and j is less than or equal to

to the distance between the starting address and min\_dist:

min\_dist is now equal to j

at end of for loop:

next\_address = min\_dist

distance between the starting address and next\_address is calculated and saved

\*distance\_between function takes O(n) time because of the list index() function

**Scalability and Adaptability**

This algorithm would be able to scale with almost any number of packages. If new addresses are added, the distance data spreadsheet must be updated in the same way it was originally created. The .csv reader functions should be able to easily incorporate new addresses and new distance data. The trucks, however, would need to be loaded manually because their loading process is not entirely automated. There may be space-time complexity issues if there are a very large number of addresses. The algorithm currently, however, relies on the addresses being entered into the distance spreadsheet and the package list spreadsheet in the same format. If they are not, they would have to be handled manually.

**Software Efficiency and Maintainability**

Because the program was mostly developed following the functional programming model, any changes to the program would just require modifying existing functions. The software runs at polynomial time O(n3) in worst case scenario and the space complexity is O(n2). The comments located within the code are well detailed and show the flow of the program to a potential developer who could add additional functionality. Related functions and methods are grouped together so that they are easy to access and modify. They divided into five .py files: one for the GUI, one for Truck objects and methods, one for .csv file handling, one for the hash map data structure and its methods, and one for distance-related functions. The program also meets the given requirements of the project to have the total miles traveled by all trucks under 140 miles. The program does so in 120 miles.

**Strengths of the Nearest Neighbor Algorithm**

1. A strength of the nearest neighbor algorithm is that it is easy to follow and easy to implement. It may not be the most optimal solution to the problem, but it gives a solution that can be used for almost any kind of data.
2. Another strength is that it is scalable and adding additional data points to the algorithm is easy to do. There would be no change in accuracy if more data is added to the problem. In this case, adding more packages to a truck would not affect the accuracy of the algorithm.

**Other Possible Algorithms and their Differences**

**Christofide’s Algorithm**

The Christofide’s algorithm is an algorithm that uses a minimal spanning tree (Christofide’s). A big difference between Christofide’s and Nearest Neighbor is that Christofide’s algorithm has a 3/2 approximation, meaning that its worst case is about 3/2 of the optimal solution (Weru, 2021). It also requires that the vertices follow the triangle inequality. This means that the distance between two cities is the shortest possible distance with no other shortcuts. This kind of restriction does not affect the Nearest Neighbor algorithm. The time complexity of Christofide’s is also slower than Nearest Neighbor on average but may have more optimal solutions.

**2-Opt**

The 2-Opt algorithm is another algorithm that could have been used. It starts with an initial solution that has visited all the vertices. It then tries to improve the solution by going through every 2-edge combination and updating the new original solution if an improvement has been found (Venhuis, 2019). The algorithm does this until no improvement are found. The difference between 2-Opt and Nearest Neighbor is that 2-Opt starts with an arbitrary solution that may be optimal or not. Using the NN algorithm could actually be the first step of the 2-Opt process. There is no sort of swapping edges in the NN algorithm like there is in 2-Opt. Because of this, 2-Opt is more complex than the Nearest Neighbor algorithm.

**Explanation of Data Structure**

For this project, a hash map or hash table data structure was created. It was created by using Python lists to store data. When an item is inserted into the hash map, the insert function requires key and value parameters. The key is hashed using Python’s hash function, and the key and value pair is inserted into a list using the hashed value as the index. If there is already an item at the hashed address, the item is appended at that index. On average, accessing, inserting, and updating items in the hash map takes O(1) time. If there are multiple items stored at the same hashed index, then the worst-case time complexity is instead O(n). The worst-case space complexity of the hash map structure would be O(n2) but on average is O(n). The hash map is initially created with a size of 64 indexes. Because Python lists are dynamic, however, the hash map capacity can also be increased dynamically. The size can also be easily increased by changing one line in the code for the hash map.

**Look-up Function Details**

The hash map has a look-up function that allows the user to retrieve a package’s information by using the package ID. The hash map takes the ID, hashes it, and uses that hashed value to find the location of the package. If there are multiple packages at the same address, it loops through until the package with the correct ID is found. The user can also access and update various attributes of the package using a similar process.

The time complexity of the look-up function is O(1) on average and at worst is O(n). The function can reach O(n) if the hash function hashes all keys to the same address. If this happened, the look-up function would have to loop through the items found at the hashed index until it found the correct key. This, however, is unlikely to happen because of the usage of Python’s hash function.

**Data Structure Space Usage**

The space complexity of the hash map on average is O(n), but at worst it could be O(n2). This would be due to the 2D lists that are used in the data structure. It would reach O(n2) if all the packages were hashed to the same index. If the number of packages increases, the hash map should scale okay and not take up too much space; however, there could be an issue if many of the packages are hashed to the same key address which would require up to O(n2) space.

**Effects of increased Trucks or Cities**

If the number of cities or addresses increase, there would not be a significant impact on time complexity because the creation of the distance graph would take O(n) time. If a large number of addresses are added, there could be issues with space and memory because the distance graph has a space complexity of O(n2). A large change to the number of trucks would not have a significant impact to space or time complexity. All functions related to the trucks all have a linear or constant space and time complexity.

**Strengths and Weaknesses of the Hash Map**

The hash map can be excellent at retrieving and updating items in the hash map. On average, the hash map is faster than a list at searching for an object. Even though there is not a delete function implemented in this hash map, it would be more efficient than deleting an item in a list/array.

It would not be the best data structure for an ordered set of values because of the hashing function. If there are many collisions during the hashing process, there could an issue with both time and space complexity. Retrieving, updating, and deleting items could take O(n) time, similar to a list. The hash map could also potentially take O(n2) space. The efficiency of the hash map heavily relies on the reliability of the hash function to create unique hash values which can be a weakness.

**Alternate Data Structures**

**Linked Lists**

One data structure that could have been used is a linked list. For this scenario it would be best to use a doubly linked list that can be traversed by either side. Linked lists uses nodes that hold information and point to either the node after it, before it, or both. Linked lists are good for insertion and deletion. They are not the most efficient for memory usage. They are much slower than hash maps for searching for an item by a key value. Linked lists use nodes instead of lists like a hash map. They also don’t use any sort of hash function and its items are ordered.

**Lists/Arrays**

Another data structure that could have been use is an array or a list. Lists are very good are accessing items. They take constant time to retrieve an object using an index. Popping and pushing items onto a list also only take O(1) time. One advantage a hash map has over a list is the time it takes to search for an item. Searching for an item in a list takes O(n) time while a hash map can, on average, do it in constant time if there are minimal collisions. Lists also use indexes instead of keys like a dictionary.

**Different Approach/Improvements**

If I had to do this project again, I would be more mindful of the time complexity that the program required. Because of certain functions, the overall project takes O(n3) time in the worst-case scenario. This was because some list functions used in the program were not the most optimal choices. I could have tried to use more sets instead of lists because their searching function only takes constant time. I would also add some more functionality to the GUI and allow the user to add, remove, or edit packages and their information. I would also try to fully automate the loading of the trucks and determining which packages go in what truck. There is also plenty of room for the total milage of the entire route to decrease. This would probably, however, require a different algorithm implementation instead of the Nearest Neighbor algorithm.

**References**

*Christoﬁdes’s -approximation for Metric TSP*. (n.d.). Retrieved February 11, 2022, from https://www.cs.cornell.edu/courses/cs681/2007fa/Handouts/christofides.pdf

Venhuis, M. (2019, April 22). *Around the world in 90.414 kilometers*. Medium. Retrieved February 11, 2022, from https://towardsdatascience.com/around-the-world-in-90-414-kilometers-ce84c03b8552

TLMaths. (2014, April 8). *AQA Decision 1 8.04 The Travelling Salesperson Problem: The Nearest Neighbour Algorithm* [Video]. Youtube. <https://www.youtube.com/watch?v=b7wW5mCPh5U&t=275s>

Weru, L. (2021, August 24). *11 animated algorithms for the traveling salesman problem*. STEM Lounge. Retrieved February 11, 2022, from https://stemlounge.com/animated-algorithms-for-the-traveling-salesman-problem/