Marc T. Rios

ID#787989

C950 – Data Structures & Algorithms II

**WGUPS Routing Program Write-up**

**I1, I2.** The algorithm I decided to use for this project was Nearest Neighbor (NN). The two reasons I chose this algorithm was because it is intuitive and simple. Since the steps in NN are straightforward, it is easy to implement; moreover, the number of delivery addresses is low, so using a simple algorithm makes sense for this scenario. The packages were pre-sorted outside of the program on MS Excel using different sections from the provided package file as the guide. In Excel I was able to sort the packages in delivery priority based upon some of the column data, like delivery deadline, address, special notes. The packages that have a delivery deadline time in the AM hours were placed in the delivery truck that left the earliest, truck one. Any packages loaded into any truck that contained the special instructions “Must be delivered with,” were coupled with their associated packages. The packages that contained “Can only be on truck 2” in the special notes, were placed in truck two’s cargo. If a package was “Delayed,” the delayed package was placed in truck two’s cargo. Truck two was scheduled to leave right when the delayed packages arrived, at 9:05AM. All other packages that do not have a deadline or special condition were placed in truck three. When the package file with the details is read into the program, the hash table is generated automatically; to find and create the most optimal route for each truck, only the package IDs within each truck’s cargo is needed. Since the packages were presorted in Excel, the program code does have a sorting packages section that needs to be maintained. The special notes section of the package details file is structured in a way that allows the person creating the file to use their interpretation on which words best describe the situation. For example, one may use “Must be delivered with” to describe the package’s dependence on another, while another person may use “Should be delivered with” to describe the same codependency. Excel cuts out this ambiguity and the need to always maintain and test that the presort code is working properly. The company is small, and packages delivered in a day are not high; therefore, using Excel to decide which truck will deliver which packages is the most economical choice for the fledgling company because code maintenance can be costly. If the company branches out into other cities and the amount of packages to deliver increases, the Nearest Neighbor algorithm used in the program would work best if the program was used by each WGUPS hub separately; meaning every individual hub would run its own program for its packages.  In other words, using this algorithm nationally where only one program would run the entire logistics of the packages would not work for this algorithm. If the company wants to eventually use one copy of program to manage the logistics nationally, the algorithm will need to be changed to something more like the Genetic algorithm. For now, the NN algorithm works fine if a copy of the program would be used separately to manage its packages for each regional hub.

The algorithm is then used on each individual truck’s cargo, so each data point is only compared against at most fifteen other data points. Every time a data point (or address) is visited, it is marked as visited and is no longer used in the future comparisons. The first truck’s route was 39 miles, the second truck traveled 38.4 miles, and the third truck traveled 50 miles. The vehicles traveled a total of 119 miles using this method, and all the packages were delivered on time.

**I3.** Two other algorithms that could have been applied to this scenario are Genetic algorithm and Greedy Heuristic algorithm. The Genetic solution is known for solving complex problems; therefore, the algorithm itself is far more complex than the NN method. When using this approach, there are some limitations that need to be adhered to for it to be successful. For example, in every route, each address should not be repeated because loops will appear. Only valid routes can be considered for this algorithm; moreover, an address on the far east side of the city is not part of the route of an address on the complete other side of the city. This algorithm has far more iterations than the Nearest Neighbor algorithm and takes longer, timewise, to find a solution. (Abdulkarim, 76-77)

The Greedy Heuristic finds the best local route, and then it optimizes that route to find the best global optimal solution. To begin it sorts all the edges with the minimum distance, and it selects the best choices if no loops are formed in the process. As for all heuristic approaches, there is no guarantee that the optimal solution will be found. The complexity of this algorithm is high when compared to the Nearest Neighbor method; moreover, the execution time of the greedy approach is longer than the NN algorithm. For the Nearest Neighbor, Genetic, and the Greedy Heuristic algorithms, the time-complexity is O(N2log2(N)). (Abdulkarim, 76-77)

**J.** There are a couple of things I would have done differently if I had to do this project all over again. When I started this project, I created the hash table using a tuple to bind the package ID number with the dictionary of all the package details. Because I used a tuple, it did not allow me to update or change any of the information, like delivery status. I had to work around this by writing more code. Towards the end of completing the program, I realized I could have used a list to house the package ID and its details, instead of a tuple. The advantage of the lost, is it allows me to alter the information within it, the delivery status to be precise. I did change the hash map to use the list feature, but I was so far along in the project that it was too late to use the list to its full potential. Most of the important pieces of code were written with a tuple in mind; consequently, I would have to rewrite the entire program to take full advantage of the list feature. In hindsight I would have created the hash map with only lists and dictionaries, instead of lists, tuples, and dictionaries. The second thing I would have done differently is to start my project with the creation of the user interface. When I did this project, I decided to create the user interface as one of the last components. The reason I believe building the interface sooner would have been beneficial is because I could have used the user input as triggers in my code, which would have made solving the problem easier for me. I was still able to overcome both issues discussed in this paragraph with creativity and willingness to try different solutions until one worked to my satisfaction.

**K1.**  I used two different data structures for this project. To hold the packages information, I used a hash table where the package ID was used to get a package’s details. There are three functions that interact with the hash table in my project. The add\_data function inserts a package’s complete information into the hash table without any possibility of collision. The get\_data function allows for retrieval of all of the package details with the use of the package’s ID as the lookup key. The get\_data\_with\_address function allows for retrieval and manipulation of desired data with the address as the lookup key. The efficiency of the hash table structure is O(1) for both the add\_data and get\_data functions. The third function is slower at O(n). As more packages are added to this data structure, more buckets for the hash table will need to be created. There will be a point where a delete function will need to be added to the hash table to make sure the amount of memory allocated does not get too high. In other words, once certain packages are delivered, the data within a bucket of the hash table, would be deleted to make room for future deliveries.

I used an adjacency matrix to create a two-dimensional view of the distances between each address. The map matrix is supplemented with another two-dimensional list with the address location as the first element in each list and a 0 or 1 as the second element in each list; the 0 means unvisited and the 1 means visited. These data structures have a worst-case complexity of O(n). The self-adjustment part of this data structure allows for a faster run-time because every time an address is visited, the address is no longer checked against other addresses because it is considered visited. In other words, every time a node location is considered visited (aka 1 in the structure), there are fewer locations to do comparisons within the algorithm, which speeds up the execution time. The amount of memory needed to use a map matrix is high because it uses double the amount of addresses loaded into the program. In order to ensure the memory allocation is kept at a manageable amount, a map matrix should be created daily based on that day’s delivery addresses; the way the program was built, a new map matrix is created every time a new distance file is read into the program.

  When adding more packages and addresses to the data structures, the number of buckets within the hash table buckets will need to increase, and the size of the map matrix will increase as well. The increase in the amount of memory allocated for the hash map is not significant, however, the map matrix structure uses a lot of memory since every address is linked to every other address on the map. Due to the way the code is set up in my program, when the new data files are read into the program, both the hash table and map matrix data structures will be created automatically.

**K1, K2.**  Two other data structures that would have met the same criteria and requirements for this given scenario are linked list and adjacency list. There are a few differences between an adjacency matrix, an adjacency list, and a linked list. “A linked list is a linear data structure where each element is a separate object. Each element (we will call it a node) of a list is comprised of two items - the data and a reference to the next node. The last node has a reference to null. The entry point into a linked list is called the head of the list.” (Adamchik) The adjacency matrix takes up more space in memory than the adjacency list, regardless if all addresses or vertices are connected or not; the matrix takes up the number of vertices squared. For dense graphs and weighted graphs (of which this scenario is), the matrix works better than other data structures. The adjacency list is an array of linked lists; this combination of data structures works better with unweighted graphs and are slower with dense graphs. Also, the adjacency list and linked list structures use less space for sparse graphs, vertices that are only connected to a few other vertices, and the execution time of the adjacency list is faster than the adjacency matrix. (Anonymous)

**Works Cited**

1. Abdulkarim, Haider A, and Ibrahim F Alshammari. “International Journal of Engineering and Advanced Technology (IJEAT) .” *Comparison of Algorithms for Solving Traveling Salesman Problem*, vol. 4, no. 6, Aug. 2015, pp. 76–79.
2. Adamchik, Victor S. “Linked Lists.” *Linked Lists*, 2009, www.cs.cmu.edu/~adamchik/15-121/lectures/Linked%20Lists/linked%20lists.html.
3. Anonymous. “Graph Representation - Adjacency Matrix and Adjacency List.” *Algorithms*, 28 Jan. 2019, algorithms.tutorialhorizon.com/graph-representation-adjacency-matrix-and-adjacency-list/.