**C950 Algorithm Overview**

**Nicholas Reynolds**

**06/18/23**

**Stated Problem:**

This assignment’s purpose is to choose and implement an algorithm that will ensure the delivery of 40 packages on time while also considering various special requirements some packages may have and keeping the total mileage of the trucks being used under 140 miles.

A.  Identify a named self-adjusting algorithm (e.g., “Nearest Neighbor algorithm,” “Greedy algorithm”) that you used to create your program to deliver the packages.

**I used a greedy nearest neighbor algorithm to complete my project.**

B.  Write an overview of your program, in which you do the following:

1.  Explain the algorithm’s logic using pseudocode.

**Nearest Neighbor**

**CalculateDistance(loc1, loc2)**

**Check distance table for distance between loc1 and loc2**

**nearestNeighbor(loc1){**

**nearest = null**

**for each loc x{**

**calculateDistance(loc1, x)**

**if nearest == null**

**{nearest = x}**

**Else if calculateDistance(loc1, nearest) > calculateDistance(loc1,x)**

**{nearest = x}**

**Else**

**{continue loop}**

**}**

**return nearest**

**}**

2.  Describe the programming environment you used to create the Python application.

**I used PyCharm as my IDE and used Python 3.11 for my project. My hardware was a desktop PC running Windows 10.**

3.  Evaluate the space-time complexity of each major segment of the program, and the entire program, using big-O notation.

**The major segments of my program have an analysis of their time and space complexities present directly above them in their source code as comments. My programs overall time complexity is O(n^3\*d), with the d representing the number of locations and n representing the number of packages, which is an area that I would need to work on improving if I revisited this project.**

4.  Explain the capability of your solution to scale and adapt to a growing number of packages.

**My current solution would have trouble scaling to a larger number of packages because I manually created the package lists for each truck. This could be fixed by creating a function that automatically creates the lists based on defined rules but from my understanding that wasn’t required by the rubric, so I haven’t done so. My current algorithm is also not as efficient as some others I could have used and so as the number of packages increases the time that my program takes would increase exponentially.**

5.  Discuss why the software is efficient and easy to maintain.

**My software is well commented, and the variable names and function names are descriptive so anyone else looking at the software should be able to easily understand what my code is doing. My program is efficient because it uses a simple algorithm which has a quick run time, and my code could be efficiently maintained or added to because I believe it’s well organized into different classes and functions which would allow another developer to quickly find whatever part of my code they needed to change.**

6.  Discuss the strengths and weaknesses of the self-adjusting data structures (e.g., the hash table).

**My chaining hash table allows for quick insertion, deletion, and update of data contained in the hash table and is designed to stop any collisions from happening. A potential weakness of my chaining hash table is that, depending on how it is initialized, it could end up using more memory than is needed because many of the chains could be empty.**

G.  Provide an interface for the user to view the status and info (as listed in part F) of any package at any time, and the total mileage traveled by all trucks. (The delivery status should report the package as at the hub, en route, or delivered. Delivery status must include the time.)

1.  Provide screenshots to show the status of all packages at a time between 8:35 a.m. and 9:25 a.m.

A screenshot of a computer

Description automatically generated

2.  Provide screenshots to show the status of all packages at a time between 9:35 a.m. and 10:25 a.m.

A screenshot of a computer

Description automatically generated

3.  Provide screenshots to show the status of all packages at a time between 12:03 p.m. and 1:12 p.m.

A screenshot of a computer

Description automatically generated

H.  Provide a screenshot or screenshots showing successful completion of the code, free from runtime errors or warnings, that includes the total mileage traveled by all trucks.

A screen shot of a computer

Description automatically generated

I. Justify the core algorithm you identified in part A and used in the solution by doing the following:

1. Describe at least **two** strengths of the algorithm used in the solution.

**The first strength of my nearest neighbor algorithm is that it is fairly simple and thus easy to implement and understand. This is beneficial if other people will be working on this software over time because it would take them less time to understand what is being done and it would be easy to update if needed. The second strength of this algorithm is that it is also easily adapted to various situations as well as additional data. If the number of packages were to be increased, while it wouldn’t be the most efficient solution, it would be able to adapt to the increase in packages easily.**

1. Verify that the algorithm used in the solution meets all requirements in the scenario.

**My algorithm meets all of the requirements in the scenario by loading all packages based on their specific notes and delivering all of the packages by their deadlines with a combined milage of less than 140 miles.**

1. Identify **two** other named algorithms, different from the algorithm implemented in the solution, that would meet the requirements in the scenario.

**The two additional algorithms that I could see being used for this scenario are Dijkstra’s shortest path algorithm or Floyd-Warshall’s algorithm.**

1. Describe how each algorithm identified in part I3 is different from the algorithm used in the solution.

**Dijkstra’s shortest path: My algorithm calculates the distance from a current location to all other locations and picks the shortest distance and repeats. Dijkstra’s algorithm calculates the shortest path between a location and any other point based on the distances between other locations. As long as the graph that Dijkstra’s shortest path is run on doesn’t change it only needs to be run once and saved. In cases where locations aren’t being changed much Dijkstra’s shortest path can be incredibly efficient as a result. It is a bit more complicated to implement than a basic nearest neighbor algorithm as well.**

**Floyd-Warshall: This algorithm is similar to Dijkstra’s algorithm but, instead of calculating the shortest distance between a single point and a destination point, it calculates the distances between each pair of locations and then can be used to calculate the shortest distance to a destination point based on the distances between individual locations as well as which locations need to be stopped at. It provides for more customizability of routes with multiple or changing stopping points. A weakness of this algorithm is that it is very resource intensive to run.**

J.  Describe what you would do differently, other than the two algorithms identified in I3, if you did this project again.

**If I were to do this project again, I would try to implement the Floyd-Warshall algorithm because it seems like it would be particularly suited to this project. My other main change would be to design a way for my packages to be loaded automatically, which would improve the scalability of my program. If it wasn’t taken care of as a part of the above changes, I would also like to work on improving the time complexity of my program overall.**

K.  Justify the data structure you identified in part D by doing the following:

1.  Verify that the data structure used in the solution meets all requirements in the scenario.

**The data structure I used was able to effectively load and unload packages as needed efficiently, and the packages were easily accessed using a search function. Because hash tables perform searches in O(1) time on average this data structure is a great choice for a situation where a large amount of lookups need to be performed. Using this data structure, I was able to deliver all packages on time based on their provided constraints while keeping my total mileage under 140 miles.**

a.  Explain how the time needed to complete the look-up function is affected by changes in the number of packages to be delivered.

**As a worst-case scenario, if all of the packages were somehow added to the same chain in my chaining hash table, my search function would have to look through n packages thus would be O(n). On average though it should be O(1) regardless of how many packages are added.**

1. Explain how the data structure space usage is affected by changes in the number of packages to be delivered.

**Hash tables generally have a space complexity of O(n) which applies to the data structure I used. As more packages are added the space required would increase proportionally at a rate of n.**

1. Describe how changes to the number of trucks or the number of cities would affect the look-up time and the space usage of the data structure.

**Look-up time:**

**The look up time would remain the same regardless of the number of cities or truck that were added.**

**Space usage: As more trucks are added a proportional amount of space would be required to account for those additional trucks. Each new city would require a new instance of my chaining hash table so the space required would increase proportionally to the number of cities deliveries were being made in.**

2.  Identify **two** other data structures that could meet the same requirements in the scenario.

**Two other data structures that could be used for this project could be a binary search tree or a queue.**

1. Describe how each data structure identified in part K2 is different from the data structure used in the solution.

**Binary search trees are different from hash tables in that they do not reserve more memory than they need when in use while hash tables can sometimes waste memory if it’s uncertain how many entries they will have. Binary search trees are also less efficient than hash tables when it comes to insertion, deletion, and search operations (O(1) in hash tables but O(log(n)) in binary search trees). Hash tables run into a problem if they end up in a situation where they have to keep being resized which can result in then performing operations in O(n) time which is worse than binary search trees.**

**While most operations performed on queues are similar in speed to operations performed on hash tables, they are less efficient when it comes to looking up entries because the entire queue has to be looped through until the entry is found. Queues also operate in a first-in-first-out pattern which could be made to work for our package deliveries, but additional work would need to be done to ensure that packages were added to the queue in the order they were going to be delivered.**

L.  Acknowledge sources, using in-text citations and references, for content that is quoted, paraphrased, or summarized.

**Other than the *C950 – Webinar – 1 – Let’s Go Hashing* which I used to help create my chaining hash table and the python docs for the tkinter class which I used to create my UI, I only used the class resources in zybooks.**