***PROGRAM OVERVIEW- MITCHELL BLAKE***

***DATA STRUCTURES 2***

***WGUPS ROUTING APP***

**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.**

The self-adjusting algorithm that I chose to use for this program is the Nearest Neighbor Algorithm. In this algorithm the closest vertex to the current vertex is visited until all vertices

in a circuit have been visited.

The code which generates the shortest distance from a delivery address of a package to the next delivery address of a package is self-adjusting. It adjusts to the new location and provides a new shortest distance to the next closest delivery address. This self-adjusting algorithm can be found within the get\_path function of my program. The algorithm also adjusts automatically to the number of vertices/delivery addresses whether increasing or decreasing. In addition, the function which incorporates the algorithm adjusts to packages which are delivered at the same address by grouping them together and removing them from the truck together and provides a path back to the starting hub after all other vertices have been visited.

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

**1.  Explain the algorithm’s logic using pseudocode.**

ShortestDistance = 100

currentVertex = startVertex

pathList = empty

While verticesList is not empty:

For each vertex in verticesList:

If vertex.Distance < shortestDistance:

shortestDistance = vertex.Distance

currentVertex = vertex

add vertex to pathList

remove vertex from verticesList

add distance from last vertex to startVertex

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

The programming environment I used was Pycharm, and the programming language used was python. The operating system was windows 10.

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

Evaluation of space-time complexity is found in comments inside the program.

The code blocks which have been commented on within the program are the hash table class within HashTable.py, the csv distance hash table building/loading, the csv package hash table build/load, the simulation block for truck deliveries and the entire program at the beginning of the main.py file.

Distance csv space-time complexity is: O(m\*n) multi-linear

Package csv space-time complexity is: O(m\*n) multi-linear

Simulation space-time complexity is: time: O(n^2) quadratic

space: O(m^2\*n^2) quadratic multi-linear

Entire program space-time complexity: time: O(m^2) quadratic

space: O(m^2\*n^2) quadratic

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

The hash table within my solution can scale to include an unlimited number of packages and distance data. However, I did not automate the sorting of the packages. The packages were sorted manually and if the number of packages were to increase substantially it would require a large amount of time to sort these packages. My solution does not scale well at all. If I were to automate the package sorting it would be a much better solution which would adapt to a larger number of packages more easily.

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

The software is efficient because it does not take too much computational power to derive a solution. The packages per truck are sorted manually and hard-coded into the solution which prevents the program from having to exert resources to calculate those values itself. The program is rather small as well, being under 500 lines with comments included. Maintenance is easy as the hash tables are built in a class and modifying the class would affect all hash tables if a change were necessary. The code blocks which read the csv data can easily be modified if the format of the csv were to change because they load data into

the hash table by row and column instead of key/value names.

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

The self-adjusting data structure that I chose is a chained hash table. The strengths of this

Data-structure are that it deals with collisions efficiently by using the same index for several different keys. The chaining of keys within a hash table index also allows the hash table to adjust to ever-increasing size without having to expand the number of buckets. The weaknesses of this data-structure are that search operations are no longer constant because a hash table index must be iterated to find the appropriate key. There is also no easy way to find an entry which was recently inputted in reference to a previous entry because the hashing function assigns indexes almost at random.

**D.  Identify a self-adjusting data structure, such as a hash table, that can be used with the algorithm identified in part A to store the package data.**

1. **Explain how your data structure accounts for the relationship between the data points you are storing.**

The hash table stores information in buckets which are stored in a list. Each bucket is itself a list and within each list a chain of lists containing key-value pairs. The hash table stores information about packages by taking an input, which in the case of the package is a tuple which includes all the package’s information and uses the hash function on the input to generate an index to store the information. After an index is generated, a list containing both the tuple as a key and another list with package information as the value is added to the hash table index which extends the chain or list of lists stored at that index. If a search function is called the input is again put into a hash function which will generate an index where the appropriate element is stored. In the case of a search for a package Id the appropriate index is generated through the hash function and the chain found at that index is iterated. The comparison of a key with matching package Id is done in constant time because the index of the package Id property within the tuple key is already known. After a comparison fails to yield a match the next link in the chain is visited until either a match is found, or the iteration has reached the end of the chain which is done in linear time.

A chained hash table stores and accesses information more accurately and faster than a simple linear search because elements are grouped together by the hashing function. Instead of iterating through every element in the hash table a search need only iterate through the elements found at a specific index. Concerning the package Id and a search for that specific property the accuracy is defined by the fact that the index of the package Id within the tuple key is already known. If an elements key does not possess the id at the specific index there is no match, and because each package has a unique Id there can be no inaccuracies when retrieving that property.

**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.**

Two strengths of the nearest neighbor algorithm are that it is easy to implement, and a solution can be found at worst O(n^2) time complexity (Weru, 2019). It is easy to implement because a list of vertices is merely iterated, and the shortest value is repeatedly taken until the list is empty. The reason it is easy to implement is also the reason it can generate a solution quickly.

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

Total combined miles traveled by all trucks: 109.9

All packages delivered on time.

All packages were delivered according to delivery specifications.

The above points are verifiable through the user interface by running a query of all packages at 16:00 hours. All package data will load which includes special notes for each package, the time it was delivered and its deadline, and on which truck it was transported, etc. at the end of the query is a statement of how many miles all trucks traveled.

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

Cheapest Insertion and Nearest Insertion

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

Cheapest Insertion is different from Nearest Neighbor because it finds the next vertex by finding the shortest distance to the next vertex which falls between to vertices already in the sub-path. Nearest Neighbor only compares the current vertex with vertices that are not in the sub-path (Weru, 2019).

Nearest Insertion is different from Nearest Neighbor because instead of taking the vertex with shortest distance from the current vertex, it compares all vertices in the sub-path with neighboring vertices and takes the shortest distance path out of all possibilities (Weru, 2019).

**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 load the csv data into objects instead of lists, this would allow me to deal with named properties instead of indexes. I would also add a hashing function within the chained part of my hash table. That way I could have a search function which can locate a hash tables index in constant time and locate the specific key within the chained list at that index in constant time as well. I would also create a class for input validation that I could import in from a separate file. This would help clean up my main program file and make it easier to maintain.

**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.**

Total combined miles traveled by all trucks: 109.9

All packages delivered on time.

All packages were delivered according to delivery specifications.

An efficient hash-table with a look-up function is present.

Reporting of package statuses and information can be verified through the user interface

And all information is accurate.

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

Time needed to complete the look-up function is affected by the change in the number of packages because it may extend the chain which is stored at each index within the hash table. If the chain is extended to account for an additional element the look-up must iterate over one more element when searching a hash tables index. If the number of packages is small the look-up function iterates less objects, if the number of packages is large the opposite is true.

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

When the number of packages increases the data structure uses the same amount of buckets. If the number of packages increases to the point that each bucket within the hash table has elements, the length of each chain within the bucket will increase

accordingly. Although the bucket number remains the same the chains within each bucket can become increasingly larger. If the number of packages is small enough that each bucket does not contain elements the size of the data structure is substantially smaller and the length of the chains within each bucket also reflects that fact. The space required by the data structure both increases or decreases based on the number of packages and buckets necessary to store those packages.

**c.  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.**

Changes to the number of trucks would have zero affect on the look-up time or space usage of the data structure because the trucks are not included within the data structure at all only distances and packages are included. Increasing the number of cities however would create a larger data structure with larger chains within each hash table index. The look-up time would also increase as the number of cities increases and the number of keys within each hash table index that a search must iterate over increases. If this were a very large set of data, I would create a hash table that had a greater number of buckets to reduce the chance that a single bucket has an abnormally large chain.

**2.  Identify two other data structures that could meet the same requirements in the scenario.** A Binary Search Tree and 2D array.

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

A Binary Search Tree is different from the hash table because it consists of a root node where all searches originate from. From the root node the value of the search can be found by comparing values stored at each leaf node. The hash table has specific indexes generated that can be accessed in constant time based on the hashing functions output.

A 2D array is different from a hash table because it does not incorporate a hashing function. In order to find a value, the list must be iterated without the aid of knowing which index to look for a value. A 2D array however does allow easier association between elements in relation to each other. If I know that I just added an element to the array and I am searching for the most recently added element I can just calculate the length of the array and return the last element.

ALL PACKAGES AT 9:00 am and mileage of trucks

Text

Description automatically generated

Graphical user interface, text

Description automatically generatedA picture containing text, electronics, screenshot, computer

Description automatically generated

ALL PACKAGES AT 10:00 and mileage of trucks

Text

Description automatically generatedA computer screen with a black background

Description automatically generated with low confidenceA picture containing text, electronics, screenshot, computer

Description automatically generated

ALL PACKAGES AT 1:00 pm and mileage of all trucksText

Description automatically generated

A picture containing text, electronics, screenshot, computer

Description automatically generatedA picture containing text, screenshot, electronics, computer

Description automatically generated

**Screenshot of all packages delivered at 16:00 hours or 4:00p.m. with total mileage of all trucks included free from run-time errors**

Text

Description automatically generated

REFERENCES:

Weru, L. (2019, December 28). 11 animated algorithms for the traveling salesman problem. Retrieved March 20, 2021, from https://stemlounge.com/animated-algorithms-for-the-traveling-salesman-problem/