Name: Michael Wormley

Student ID: 012418005

DSA 2 Task 1 Paper

1. Identify a named self-adjusting algorithm that could be used to create your program to deliver the packages.

I selected the **nearest neighbor algorithm**. The nearest neighbor algorithm will take your current position and then find the nearest target destination and go there next. It is used for the traveling salesman problem because it works reasonably well (15% longer than the optimal tour), but doesn’t suffer from the NP-hard exponentially increasing complexity to find the optimal path, but is instead quadratic complexity(Sindhukumari, 2022; Weru, n.d.).

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

The self-adjusting data structure that I chose was a **hash table**. The hash table will store data by creating a list of items, and each of those items will be given a key that will identify that item uniquely, and each of those items will contain the rest of the data about each package it represents. It is self-adjusting in that it will dynamically resize when it’s too “full,” so when its load factor exceeds a preset limit it will expand so that it can decrease the likelihood of collisions.

1. Write an overview of your program in which you do the following:
2. **Explain the algorithm’s logic using pseudocode**

define function nearest\_neighbor\_algo(distance\_matrix, start\_city, list\_of\_cities):

# Get the number of cities

num\_cities = length (distance\_matrix)

# This is going to be the path

visited\_cities = [start\_city]

# These are the cities that haven’t been visited yet

unvisited\_cities = list\_of\_cities.remove(start\_city)

# this variable will store what city we’re currently at

current\_city = start\_city

# loop to visit each city

While len(visited\_cities) < (num\_cities):

nearest\_city = None

min\_distance = infinity

# find which city is closest to current city to be visited next

for each city in range(num\_cities):

if city is not in visited\_cities AND distance\_matrix[current\_city][city] < min\_distance:

nearest\_city = city

min\_distance = distance\_matrix[current\_city][city]

# Add the nearest city to the tour and then remove it from unvisited cities

visited\_cities.append(nearest\_city)

unvisited\_cities.remove(nearest\_city)

# move to the next city

current\_city = nearest\_city

# Return to the starting point to complete the tour

visited\_cities.append(start\_city)

return visited\_cities

1. **Describe the programming environment you will use to create the Python application, including both the software and hardware you will use.**

I will be creating this application using Python 3.11 in the IDE VS Code on my computer HP desktop computer that is running Windows 10.

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

The dominant time complexity is O(n^2), because the nearest neighbor algorithm and processing a full distance matrix are both O(n^2).

| Segment | Function | Complexity |
| --- | --- | --- |
| Package | init | O(1) |
|  | package\_lookup | O(1) |
| Truck | init | O(1) |
| import\_csv | import\_packages | O(n) |
|  | import\_distances | **O(n^2)** |
| hash\_table | insert | O(n) |
|  | search | O(n) |
|  | resize | O(n) |
| delivery | nearest\_neighbor\_algo | **O(n^2)** |

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

This program is capable of scaling and adapting to a growing number of packages by utilizing a chaining hash table that efficiently retrieves items and stores items regardless of how large the input size grows by utilizing dynamic resizing of the table to prevent collisions.

The nearest neighbor algorithm also scales more optimally to the traveling salesman problem than other algorithms, with a big O complexity of n^2.

1. **Discuss why the software design would be efficient and easy to maintain.**

This software design is efficient and easy to maintain because it utilizes object oriented programming, which means that as different functions are needed or as the existing functions need to be adjusted, the overall function of the program will not be disturbed. The difficulty of understanding the functioning of the program is also diminished by dividing the program because within an individual module, which is based on an intuitive understanding of the objects the modules represent and the actions that can perform.

1. **Describe both the strengths and weaknesses of the self-adjusting data structure (hash table).**

The strengths of the hash table are that it has fast lookup and retrieval due to each item having a key. Because my implementation of the hashtable automatically resizes itself, it can handle larger data inputs without decreasing performance.The chaining method allows for handling collisions.

The weaknesses of this hash table are that if too many keys hash to the same bucket, the lookup performance will degrade, the linked-list will require extra memory, and hash tables aren’t as good for range-based queries as other data structures.

1. **Justify the choice of a key for efficient delivery management form the following components.**

The key that was chosen was the package ID. It is unique to each package, which will prevent collisions, allow for fast lookups and updates, and is scalable even with a large number of possible packages.

**D. Cite Sources**

References

GeeksforGeeks. (2024, September 15). *Hash table data structure*. https://www.geeksforgeeks.org/hash-table-data-structure/

P, S. (2022, December 25). *Traveling salesman problem*. Medium. https://sindhukumari.medium.com/traveling-salesman-problem-9deb6853ac6

*Webinar 1*. Panopto. (n.d.). https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=f08d7871-d57a-496e-a6a1-ac7601308c71

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

YouTube. (n.d.). *Creating a Hash Map with Lists*. YouTube. https://www.youtube.com/watch?v=9HFbhPscPU0