Damon Heard

Student ID: 001670334

C950, Data Structures and Algorithms II

**Program Overview and Details**

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

This project implements a Nearest Neighbor algorithm. This algorithm starts at a point, surveys a list of points to determine the next closest point, travels to that closest point, and repeats the process until all points are visited.

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

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

*Distance Search(current location, destination):*

*Time: O(N)*

*Space: O(N)*

*Survey a row of data in list of lists (2D array):*

*If the current location/given location is in first index of the array*

*Set the first index to the current location’s index in the array*

*Increment through the rows*

*Reset the counter*

*Next, repeat the above actions for the destination*

*Find where the two points intersect in the array. The value in that intersection is the distance between the two points*

*Return the found distance*

*Get Shortest Route(list of addresses, current location)*

*Time: O(N)*

*Space: O(N)*

*Initialize a large maximum distance to be used for later comparison*

*Surveying the list of ID numbers associated with the truck object:*

*Iterate through the list and look up the destination of each package*

*Calculate distance to every destination using Distance Search*

*If the distance to the destination is less than the distance initialized, set the initialized distance to the value of shorter distance*

*Repeat iteration until the smallest possible distance is found*

*Return the smallest distance and its index in the 2D distance array*

*Deliver Packages*

*Time: O(N)*

*Space: O(N)*

*While the truck has packages in its list (size is greater than zero):*

*Iterate through the packages, performing the following for each package:*

*Find the next closes point using the above Get Shortest Route function*

*Calculate the delivery time by dividing the distance by average speed*

*Timestamp each package with its delivery time and append each time to a list*

*Append mileage to each destination to a list*

*Set the truck’s new location to the delivery address*

*Update the package status*

*When every package has been delivered:*

*Return the truck back to its starting point, recording the miles traveled and time*

*Append miles and distance to list for final calculations*

*Clear the miles list to record for the next truck*

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

Project was created in IntelliJ PyCharm IDE, v 2021.2.2 using Python 3.10.

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

Identified complexity in section B.1 with additional information in code comments and docstrings.

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

As the trucks were manually loaded according to observed requirements and constraints, the program can scale but will require an automatic sorting method according to delivery constraints. A faster, automated solution would have to be implemented if the total number of packages were to exceed much further past the 40 in this project. Packages could be sorted into different trucks depending on deadline and any amplifying data annotated in the package’s notes.

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

The software is less than 500 lines of code with easy-to-read functions. PEP8 naming and style conventions were referenced as well. In addition, the Python programming language is closer to the English language than most other programming languages, lending itself well for human interpretation. The distance information is housed in .csv files which makes parsing easy. Functions are also written in smaller blocks when possible so that they are easy to follow and implement in larger functions more central to the running of the program.

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

The chaining hash table is straightforward to implement, and the functions are very fast in terms of time complexity. Because it also uses a list for each index, multiple objects can be inserted in one index which prevents collisions. However, the benefit of fewer collisions is slightly offset by increased spatial usage which becomes a bigger concern as the program scales up.

The Nearest Neighbor algorithm is also short and easy to implement and read, making it easy to maintain. However, as a greedy algorithm it looks for the optimal solution at a singular point in time, rather than surveying an entire list of points and finding an optimal solution for the entire route.

**C. Write an original program to deliver all the packages, meeting all requirements, using the attached supporting documents “Salt Lake City Downtown Map,” “WGUPS Distance Table,” and the “WGUPS Package File.”**

See attached .zip folder.

**1. Create an identifying comment within the first line of a file named “main.py” that includes your first name, last name, and student ID.**

See lines 1 and 2 of Main.py.

**2. Include comments in your code to explain the process and the flow of the program.**

See code files which contain comments and docstrings for all major functions which identify process flow, parameters, output, etc.

**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.**

As outlined in the project requirements, a hash table was implemented to store package data.

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

The hash table easily stores all associated data to each package object stored in it. The lookup function is also very simple and any information can be found with very few lines of code.

**E. Develop a hash table, without using any additional libraries or classes, that has an insertion function that takes the following components as input and inserts the components into the hash table:**

Please see HashTable.py, and line 22 (**def insert**) for the insert function.

**F. Develop a look-up function that takes the following components as input and returns the corresponding data elements:**

Please see HashTable.py, line 35 (**def search**) for the lookup function. Option 1 when running the program prints the package details.

**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.)**

Please see Main.py, option 1 for package info, option 2 for info by time, and option 3 for truck mileage.

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

Please see C952\_PA\Project\Screenshots\Time Interval Queries\0835-0925.

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

Please see C952\_PA\Project\Screenshots\Time Interval Queries\0935-1025.

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

Please see C952\_PA\Project\Screenshots\Time Interval Queries\1203-1312.

**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.**

Please see Successful\_Run.png in C952\_PA\Project\Screenshots

**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.**

As stated above, the Nearest Neighbor algorithm is short, easy to read, and therefore easy to maintain. Combined with a two-dimensional array for distance data, locating distances by index is very fast and straightforward, and it returns the best choice in a given moment.

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

All packages with delivery constraints were delivered within those constraints and deadlines specified. The solution comes in at 107.0 miles, well under the 140-mile requirements.

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

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

Prim’s algorithm, using a weighted and undirected graph, would be able to meet the requirements provided the conditions are the same. Prim’s is another type of greedy algorithm which builds a solution one node at a time but can also survey visited nodes. Nearest neighbor typically only looks at unvisited nodes. (“Prim’s Spanning Tree Algorithm,” n.d.)

Dijkstra’s algorithm surveys an entire graph and finds the shortest path from the start to every node in the graph, producing the best solution after considering all data. Nearest neighbor only surveys nodes one at a time.

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

I would implement an automated sorting function based on annotated delivery requirements instead of sorting manually. This function would account for deadlines, physical location of the destination, and other constraints such as the need to deliver multiple packages at the same time.

**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 hash table is designed to include the insert and lookup functions as per the requirements. The hash table stores the required objects with their associated data so that lookup is fast and only requires the package ID.

The Nearest Neighbor algorithm and delivery functions worked together to survey each node as the simulation truck ran, dynamically plotting a path until all nodes were visited. This kept the total distance under the 140-mile limit by a good margin.

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

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

The lookup function runs at O(N) complexity so an increase in package amounts would have negligible effect on this function. With regard to spatial complexity, scaling up the number of packages would most likely result in larger storage needs as some buckets would go unused and others will probably have multiple lists stored in them. However, using a hash table that is not chaining will almost certainly lead to collisions and therefore, misinterpreted data.

**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.**

The lookup function is a hash table function. Adding more trucks and/or cities also implies an increase in the number of packages. This would require more packages to be hashed which increases the risk for data collisions. More space can be allocated to reduce collision risk, but that will come at the expense of larger space requirements.

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

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

A weighted, undirected graph would have met the same requirements, does not come with the risk of collisions as the hash table does, and can be written to exclude data duplication. It is also scalable, though depending on the implementation, may increase time complexity to as much as O(N^2).

An AVL tree is dynamic in that it rebalances the tree as nodes are added and deleted. This self-adjusting data structure meets the requirements of this exercise and, if this program were to scale, could potentially be optimal in a real-world scenario which requires constant adjusting of routes depending on external conditions such as weather and traffic. It also lends itself to better pre-planning of routes.

**References:**

TutorialsPoint. (n.d.) *Prim’s Spanning Tree Algorithm.* Retrieved from https://www.tutorialspoint.com/data\_structures\_algorithms/prims\_spanning\_tree\_algorithm.htm