NHP2 – NHP2 TASK 1: WGUPS ROUTING PROGRAM

C950: DATA STRUCTURES AND ALGORITHMS II

PRFA – NHP2

Jared Hines

Program Overview

jhine85@wgu.edu

WGU ID: #001497856

# **Part 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 was used within my program was a greedy algorithm. This can be located within distance.py, named determine\_best\_path.

**Part B.**

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

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

The greedy algorithm is located in file distance.py and is defined as determine\_best\_path. Pseudocode for greedy algorithm:

1. Input:
   1. List for all packages currently on trucks named truck\_packages
2. Start:
   1. Hash table is made for package\_data
   2. best\_path set to 1 (WGUPS hub)
   3. packages list is set to be copy of truck\_packages
   4. Create empty list for packages on truck
3. For all packages on truck:
   1. Retrieve info from package\_data
   2. Appends info to list created for packages on truck
4. For all packages on Truck:
   1. Find address for most\_recent\_stop
   2. Find next\_stop in list
      1. Retrieves all addresses for packages still on truck
      2. Find distance between each address and most\_recent\_stop
      3. Determines and returns closest address to most\_recent\_stop
   3. Append best\_path with entry from next\_stop
   4. When package is delivered to address, remove from list
5. If value is None in best\_path
   1. Remove None value
6. Append address for package ID 1 for truck to return to WGUPS hub
7. Return best\_path
8. **Describe the programming environment you used to create the Python application.**

The programming environment that I used to create the Python application was PyCharm Community Edition 2021.3.3. Python programming language version 3.10.0 was used. The hardware environment includes an AMD Ryzen 5800X processor, NVIDIA RTX 2070 graphics card, M.2 SSD for storage, 16 GB of RAM; and peripherals such as a mouse, keyboard, and monitor.

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

I have all big-O notations for space-time complexity in program but I will list them here. Numeric numbers indicate .py files and alphabetic symbols represent functions within the files. All big-O notations for time complexity are highlighted in blue. All big-O notations for space complexity are highlighted in yellow.

1. chaininghashtable.py – O(n) – O(n)
   1. \_\_init\_\_ - O(1) – O(n)
   2. insert – O(n) – O(1)
   3. search – O(n) – O(1)
   4. remove – O(n) – O(1)
2. distance.py – O(n^2) – O(n)
   1. retrieve\_distance\_data – O(1) – O(1)
   2. retrieve\_address\_info – O(1) – O(1)
   3. distance\_from\_first\_address\_to\_second\_address – O(n) – O(n)
   4. find\_next\_delivery\_address – O(n) – O(n)
   5. determine\_best\_path – O(n^2) – O(n)
   6. retrieve\_address\_index – O(n) – O(1)
   7. retrieve\_address\_from\_id – O(n) – O(1)
   8. retrieve\_total\_distance – O(n) – O(1)
   9. truck\_delivery\_times – O(n) – O(n)
   10. package\_status\_update – O(n^2) – O(n)
3. main.py – O(n)
   1. user\_interface – O(n) – O(1)
4. package.py – O(n) – O(n)
   1. package\_chaining\_hash\_table – O(n) – O(n)
   2. retrieve\_package\_details – O(n) -O(1)

The entire program has a time complexity of O(n^2) and a space complexity of O(n).

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

The program is able to adapt to a large amount of packages and can even add more trucks if needed. The hash table can change to allow more or less packages to be added or removed. Each truck and it’s path is calculated independently, so packages would just need to be added to the trucks. More trucks can also be added if too many packages are at the hub. Since the time complexity of the entire program is O(n^2), a growing number of packages would increase the runtime.

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

The software is efficient and easy maintain due to the development environment. Python is an easier programming language to follow than many. The program also contains many comments that will allow for easier reading and understanding. If using Pycharm or a similar IDE, things within the code are color-coded to allow for easier differentiation of parts of a function. Maintenance is easy thanks to version control and the revert option. I ran into many instances where I needed to go back, many due to my cats jumping onto my keyboard, and was able to revert to previous local saves. The code can be saved locally or to a repository and edited from there if needed. It is also easy to maintain due to the simplicity of Python as a programming language. The code is easy to follow even without the comments. Also, many parts of the code have been broken up and simplified to create a simpler way to read and edit if necessary. The program itself is not the most efficient due to the way it was written but is a tradeoff for maintenance feasibility. Efficiency of the program itself could be better if packages were organized onto trucks better or different algorithms are set into place to decrease overall mileage.

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

Hash tables can be adjusted to the amount of packages and/or trucks that are available. They are typically easy to update and search. A weakness corresponding directly to this strength is that if more packages or truck are added, the longer it will take to create the hash table and search it. The best\_path function is able to determine if multiple packages can be dropped off at one address, eliminating the need to revisit the address. It is weak due to the fact that it only calculates from stop to stop, not calculating all stops to find the shortest possible path to deliver all packages.

**Part 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.”**

**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.**

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

All parts for this are located within the code and begin with #.

**Part 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.**

Located within chaininghashtable.py, there is a class named HashTableChaining that stores information for the packages. Figure 7.8.2 in Zybooks is referenced because that is where the chaining hash table is derived from. Once all information is entered into the hash table, it is easily searched and updated. A hash table is a data structure that maps keys to values. Within the program, ID numbers are given to all packages and values are stored within the table matching the key. When called upon using the ID, the hash table can find all values associated with a particular ID by retrieving the information from the table.

**Part 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:**

**•   package ID number**

**•   delivery address**

**•   delivery deadline**

**•   delivery city**

**•   delivery zip code**

**•   package weight**

**•   delivery status (e.g., delivered, en route)**

All parts for this section is located within the code in chaininghashtable.py.

**Part F.**

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

**•   package ID number**

**•   delivery address**

**•   delivery deadline**

**•   delivery city**

**•   delivery zip code**

**•   package weight**

**•   delivery status (i.e., “at the hub,” “en route,” or “delivered”), including the delivery time**

All parts for this section is located within the code in chaininghastable.py. This part is defined as the function search.

**Part 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.**

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

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

Screenshots will be on the following pages and I will also include them in a folder within the file named “screenshots.”

1. Screenshot of package statuses at 8:45 a.m

A screenshot of a computer

Description automatically generated with medium confidence

1. Screenshot of package statuses at 10:20 a.m.

A screenshot of a computer

Description automatically generated with medium confidence

1. Screenshot of package statuses at 1:05 p.m. 24 hour clock is used within the program and user is prompted to use that. That is why 13:05 is shown.

A screenshot of a computer

Description automatically generated with medium confidence

**Part 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.**

**Text

Description automatically generated**

**Part 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.**

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

**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.**

1. The first strength is that number of total packages and trucks do not matter. Algorithm is run so that trucks are independent of each other and only the packages located within one truck matters. The second strength is that if multiple packages need to be delivered to the same address, it accounts for that so there is no need for multiple visits.
2. I believe that all requirements are met, located within the code.
3. Within the supplemental resource page and textbook for the course, there are two other algorithms that are mentioned that could be used. The first is Dijkstra’s algorithm. According to Tepe (n.d), Dijkstra is different because rather than looking at just the next distance, it looks ahead and creates values for the remaining distances between packages (Tepe, n.d) . The algorithm that I chose only looks to find the next closest stop. Another algorithm that could be used would be a dynamic programming method. Dynamic programming creates smaller subsets of problems using recursion to try to simplify a problem. Dynamic programming is different because it will look to solve the problem in the most efficient manner for all possibilities, and then choose the best outcome (Lysecky & Vahid, 2018). The greedy algorithm grabs the nearest address without accounting for travel to other stops, while dynamic programming would.

**Part 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 reattempt this project, I would have just one instance of the hash table. Instead of having a new one rebuilt every time the path is initiated, I would just have the initial has table updated. Another thing that I would do is use the resources that my course instructor provided (including himself) to help solve some issues that I had. Lastly, I would use an algorithm that searches best path from start to finish before loading all trucks. That would help lower my overall mileage.

**Part 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.**

**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.**

**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.**

**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.**

1. The hash table that I used (chaining) seems to meet the requirements.
   1. Big-O notation for the search function is O(n), so this means that if there are more packages, the time to complete the look-up increases.
   2. The buckets located within the hash table are able to hold multiple objects. The data structure space should not be affected because adding more packages and their information could just be chained in the buckets. The number of buckets would not change but the amount of information within the buckets would.
   3. Adding more trucks to the problem would mean more calculations for the best path function and updating the status of the packages. Adding more cities would mean changing which packages are added to which truck, assuming there is still a mileage requirement that must be met.
2. Two other structures that would work are dictionaries and heaps. Both of these are mentioned within Zybooks (Lysecky & Vahid, 2018).
   1. A dictionary would allow for the information of packages to be accessed quicker than the hash table. Both hash tables and dictionaries use keys to and values. Hash tables need the info stored within the buckets to be converted to a readable text, while a dictionary can be directly retrieved (Lysecky & Vahid, 2018). A heap would still create objects for the packages and store them based on ID. Priority for all packages would also be given through the ID. A hash table is much faster when being updated compared to a heap. Heaps are typically sorted in a way to access data in a certain way, such as min heap and max heap, while a hash table can have data entered into buckets and does not need to be sorted because of key-value pairs (Lysecky & Vahid, 2018).

**Part L.**

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

**Sources:**

**eText from zyBooks:**

Lysecky, R., & Vahid, F. (2018, June). *C949: Data Structures and Algorithms I*. zyBooks.

Retrieved May 14th, 2022 from

<https://learn.zybooks.com/zybook/WGUC9492018/>

Lysecky, R., & Vahid, F. (2018, June). *C950: Data Structures and Algorithms II*. zyBooks.

Retrieved May 14th, 2022, from <https://learn.zybooks.com/zybook/WGUC950AY20182019/>

**Supplemental Resources:**

Tepe, C. “C950 – Webinar-1 – Let’s Go Hashing”. C950 Supplemental Material.

Retrieved May 14th, 2022, from

[**https://westerngovernorsuniversity-my.sharepoint.com/personal/cemal\_tepe\_wgu\_edu/\_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fcemal%5Ftepe%5Fwgu%5Fedu%2FDocuments%2FMyDocs%2FC950%2FWebinar%2FC950%20%2D%20Webinar%2D1%20%2D%20Let%27s%20Go%20Hashing%2Epdf&parent=%2Fpersonal%2Fcemal%5Ftepe%5Fwgu%5Fedu%2FDocuments%2FMyDocs%2FC950%2FWebinar&ga=1**](https://westerngovernorsuniversity-my.sharepoint.com/personal/cemal_tepe_wgu_edu/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fcemal%5Ftepe%5Fwgu%5Fedu%2FDocuments%2FMyDocs%2FC950%2FWebinar%2FC950%20%2D%20Webinar%2D1%20%2D%20Let%27s%20Go%20Hashing%2Epdf&parent=%2Fpersonal%2Fcemal%5Ftepe%5Fwgu%5Fedu%2FDocuments%2FMyDocs%2FC950%2FWebinar&ga=1)

Tepe, C. “C950 – Webinar-2 – Getting Greedy, who moved my data”. C950 Supplemental Material.

Retrieved May 14th, 2022, from

<https://westerngovernorsuniversity-my.sharepoint.com/personal/cemal_tepe_wgu_edu/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fcemal%5Ftepe%5Fwgu%5Fedu%2FDocuments%2FMyDocs%2FC950%2FWebinar%2FC950%20%2D%20Webinar%2D2%20%2D%20Getting%20Greedy%2C%20who%20moved%20my%20data%5F%2Epdf&parent=%2Fpersonal%2Fcemal%5Ftepe%5Fwgu%5Fedu%2FDocuments%2FMyDocs%2FC950%2FWebinar&ga=1>

Tepe, C. “C950 – Webinar-3 – How to Dijkstra?”. C950 Supplemental Material.

Retrieved May 14th, 2022

<https://westerngovernorsuniversity-my.sharepoint.com/personal/cemal_tepe_wgu_edu/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fcemal%5Ftepe%5Fwgu%5Fedu%2FDocuments%2FMyDocs%2FC950%2FWebinar%2FC950%20%2D%20Webinar%2D3%20%2D%20How%20to%20Dijkstra%5F%2Epdf&parent=%2Fpersonal%2Fcemal%5Ftepe%5Fwgu%5Fedu%2FDocuments%2FMyDocs%2FC950%2FWebinar&ga=1>