Joshua Corpuz

C950 – Data Structures and Algorithms II

Western Governors University

C950 Documentation

**A. Identify the algorithm that will be used to create a program to deliver the packages and meets all requirements specified in the scenario.**

I used the Nearest Neighbor Algorithm to calculate the best route for the packages to be delivered. The nature of the algorithm looks for each of the current vertex’s “nearest neighbor” or nearest other node that has yet to be visited. Before the algorithm is optimized, each package is assigned to each of the trucks and loaded depending on each of the package’s notes and dependencies like delivery deadlines, trucks they need to be on and other packages they need to be delivered with. Once all the packages have been loaded, the algorithm will be called to create an optimized route for each of the truck.

The algorithm successfully delivered all the packages on time and below the project limit of 140 aggregate miles travelled between the three trucks provided. See the screenshots on section **I2** as evidence of the deliveries with the package details, and the summary of the results.

**B. Write a core algorithm overview, using the sample given, in which you do the following:**

1. Comment using pseudocode to show the logic of the algorithm applied to this software solution

Function nearest\_neighbor(truck, graph\_data\_to\_reference)

Step 1: initialize the following variables:

Minimum\_distance = 999999.99 (a ridiculously high value)

Nearest\_neighbor = [] #a list to hold the values for the nearest neighbor

vertex\_a = ‘HUB’ initialize vertex\_a to be first vertex

address\_list\_temp = [] # temporary holder for the addresses, will be used for error handling

adjacency\_list = graph\_data\_to\_reference.get\_edge\_weights(vertex\_a) #initialize the first adjacency list to the values for vertex\_a

truck.route.clear() # always clears the truck’s current route, allows for the function to be called again without causing any errors

Step 2: Find the nearest neighbor for the current vertex\_a from the adjacency list initialized in the start of the function. Looping through all the value pairs of vertex\_b and edge\_weight and returning the one with the shortest distance to the current vertex. Once found, the algorithm will append the nearest\_neighbor to the truck’s route, remove the found address from the truck’s address list and append it to the address\_list\_temp. The nearest\_neighbor will then be assigned as vertex\_a, and minimum\_distance will be re-assigned to its initial value. The algorithm will continue the loop until the truck’s address\_list has a length of 0.

while (len(truck.address\_list) > 0):

for location in adjacency\_list:

if location[0] not in truck.address\_list:

pass

elif location[0] == vertex\_a:

# removes the found nearest\_neighbor from address\_list

truck.address\_list.remove(location[0])

# adds the found nearest\_neighbor to temp\_address\_list, to reset the truck's address\_list when the algorithm is over

address\_list\_temp.append(location[0])

pass

else:

distance = float(location[1])

if (distance < min\_distance) and distance != 0:

min\_distance = distance

nearest\_neighbor = location

# removes the found nearest\_neighbor from address\_list

truck.address\_list.remove(nearest\_neighbor[0])

# adds the found nearest\_neighbor to temp\_address\_list, to reset the truck's address\_list when the algorithm is over

address\_list\_temp.append(nearest\_neighbor[0])

# adds the nearest\_neighbor and it's distance from vertex\_a to the route

truck.route.append([nearest\_neighbor[1], nearest\_neighbor[0]])

# sets vertex\_a to the current nearest\_neighbor

vertex\_a = nearest\_neighbor[0]

min\_distance = 99999.99 # resets min\_distance to an absurdly high number

# resets the adjacency list using the new vertex\_a

adjacency\_list = distance\_graph.get\_edge\_weights(vertex\_a)

Step 3: Complete the route, assign the route and re-assign address\_list

Prepend(‘0’, ‘HUB’)

distance\_to\_hub = distance\_graph.get\_edge\_weight(vertex\_a, "HUB")

Truck.route.append(distance\_to\_hub)

Truck.address\_list = address\_list\_temp

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

The program utilizes Python 3.10 with Visual Studio Code as the IDE. The program runs locally with the data coming from Excel files converted into CSV files in the root directory of the program, this data is then read by the program using Python’s csv reader module.

3. Evaluate space-time complexity using Big O notation throughout the coding and for the entire program.

The Nearest Neighbor Algorithm has the space-time complexity of O{n^2). The other functions utilized in the program have their time-complexities commented above the function declaration.

4. Discuss the ability of your solution to adapt to a changing market and to scalability

- Issue with data source

- Changes should be handled as they come in.

- The program will scale well enough because it leveraged OOP design which allows for more

trucks and more packages to be used

One of the most glaring issues with the solution as it currently stands is its reliance on excel files as its data source. Combine the need to manually convert the data set into a csv file and then load them to the program would be a rather complicated and tedious task. This would prevent it from scaling up as market and business needs change.

The other potential source of problem would be the handling of changes in the package data. In the real world, changes to package data could be constant and should be instantaneous. There should be a way to handle such changes when inputted into the system.

5. Discuss the efficiency and maintainability of the software.

The design of the project leverages OOP as much as possible. Given that, the program would be able to take on more trucks and packages to an extent. Because each part of the business process has been abstracted out to their own individual classes, the core logic and process of the program will be able to handle an increase in trucks and packages.

However, some aspects of the program like loading the package data, distance matrix, and assigning which packages go to which trucks are all custom built with the project requirements in mind. Given that, those parts of the program need to be adjusted as the data source, its format and dependencies change.

6. Discuss the self-adjusting data structures chosen and their strengths and weaknesses based on the scenario. TODO: Research self-adjusting data structures

There are two self-adjusting data structure in the project: graph.py and hash\_map.py

Strengths:

The data structure in hash\_map.py was used to store a master list of packages, this allowed for insertions, deletions, updates, and lookups to be quicker than a traditional list would provide. The data structure in the graph.py allowed the storage of the adjacency matrix provided from wgups\_distance\_table as an adjacency list which also allowed for insertion, deletion, updates and lookups to be quicker than a traditional list would provide.

**C. Write an original code to solve and to meet the requirements of lowest mileage usage and having all packages delivered on time**

All of the project’s code is original and results meet major project requirements: All packages were delivered on time and total miles travelled by all trucks was 125.8 miles.

1. Create a comment within the first line of your code that includes your first name, last name and student ID.

- Each file was labeled with first name, last name, and student id

2. Include comments at each block of code to explain the process and flow of the coding

- Each code block has comments to explain the process and flow of the coding

D. Identify a data structure that can be used with your chosen algorithm to store the package data.

1. Explain how your data structure includes the relationship between the data points you are storing.

I used a graph data structure for my distance data and a hash\_table for my package data.

The graph is loaded with data using the add\_vertex() and add\_edge\_weights() functions. First the list of the vertices is created from the addresses/locations in the wgups\_distance\_table.csv and then each of those locations are used as keys. The corresponding vertex and distance are then assigned as the values for each of those keys.

The hash\_table is loaded with package data in the beginning of the program using the insert() function of the hash\_table. First, each data set is used to create a package object which is then inserted into the hash\_table using the package id as the key and the package object as the value.

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

* Package ID Number
* Delivery address
* Delivery city
* Delivery zip code
* Package weight
* Delivery status

First, with the use of object-oriented design, I made a representation of a package by defining a class. The components stated above are then assigned as the class’s properties. The package id, delivery address, city, zip and weight will be retrieved from the WGUPS Package File while delivery status will be assigned by the program. With this class, the program will have an abstracted version which could be used in a hash table data structure as the value component.

Second, I created a chaining hash table to store these package classes. The hash table was made to have a configurable size by passing an integer in its constructor. The number tells the program how many “buckets” or “containers” the hash table will have. Next, to ensure that each key will be limited to the number of buckets, the hash key will be calculated by dividing the integer value of the chosen key by finding the remainder (modulo) against the size of the hash table (key % hash table size).

The next step would be to prepare the data given by cleaning up the format of the WGUPS Package Excel file by removing unnecessary spacings, headings and/or images from the file. It is then saved as a CSV file. With the data inculcated into a manageable format, it can then be loaded into the program.

I decided to use the package Id number as the key and used the created package object as the value that will be stored. A simple function was declared to consume this data by looping through all the rows of the CSV, reading each of the item in that row and setting the package object properties to those read values. Then, with the use of the package id number, the package is inserted into the hash\_table.

Also, since the delivery status is not part of the data set, all the packages were initialized to a default value of “AT HUB”.

**G. Provide an interface for the insert and look-up functions to view the status of any package at any time. This function should return all information about each package including delivery status.**

* The Look-up functionality, or the hash\_table’s search() function is implemented in the beginning of the program. See screenshot: (INSERT SCREENSHOT HERE)
* The Insert functionality of the hash table was implemented in a loop as the program went through each row of the package data CSV file, created a package object, and inserted each of those objects into the hash table. See screenshot: (INSERT SCREENSHOT HERE)

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

- An arbitrary time between 8:35 A.M. and 9:25 A.M was chosen, please note the different statuses of the packages. See screenshot:

2. Provide screenshots to show package status of all packages at a between 9:35 A.M. and 10:25 A.M

- An arbitrary time between 9:35 A.M. and 10:25 A.M was chosen, please note the different statuses of the packages. See screenshot:

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

- An arbitrary time between 12:03 P.M. and 1:12 P.M was chosen, please note the different statuses of the packages. See screenshot:

H. Run your code and provide screenshots to capture the complete execution of your code.

Show screen shots of parts that make sense or are necessary for the project

Final results:

**I. Justify your choice of algorithm by doing the following:**

1. Describe at least two strengths of the algorithm you choose

2. Verify that the algorithm you chose meets all the criteria and requirements given in the scenario

3. Identify two other algorithms that could be used and would have met the criteria and requirements given in the scenario

a. Describe how each algorithm identified in part I3 is different from the algorithm you chose to use in the solution

**J. Describe what you would do differently if you did this project again**

**K. Justify your choice of data structure by doing the following:**

1. Verify that the data structure you chose meets all the criteria and requirements given in the scenario.

a. Describe the efficiency of the data structure chosen

b. Explain the expected overhead when linking to the next data item

c. Describe the implications of when more package data is added to the system or other changes in scale occur

2. Identify two other data structures that can meet the same criteria and requirements given in the scenario

A list and Dictionary

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

**M. Demonstrate professional communication in the content and presentation of your submission**