C950 Task-2 FAQ

(Task-2: The implementation phase of the WGUPS Routing Program).

(Zip your source code and upload it with this file)

C950 Data Structures and Algorithms II

# A. Hash Table

Develop a hash table, without using any additional libraries or classes, that has an insertion function that takes the package ID as input and inserts each of the following data components into the hash table:

* 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

You need to have HashTable data structure implemented as part of your solution. Then, you can use dictionaries and or other built-in data structures. Avoid using libraries other than CSV and DateTime Python libraries.

Provide screenshot of your HashTable.

# B. Look-Up Functions

Develop a look-up function that takes the package ID as input and returns each of the following corresponding data components:

* 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

# A computer screen shot of a program code Description automatically generated

The above screenshot is a search/lookup function for the hash table. It returns the data requested if successful else a simple string asking to try again.

# C. Original Code

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

Major code blocks screenshots go here showing implementation

A computer screen shot of white text

Description automatically generated

The above code block shows a few global variables used to allow modifications at a later date and to make expressions more readable. There is also the beginning of the function that parses CSV data and loads it into the has function.

A computer screen with text

Description automatically generated

The above block includes a majority of the variables and objects assigned in the main program. All 3 delivery trucks, the call to calculate their routes, two of the three csv’s being formatted, as well as the beginning of a rudimentary CLI for demonstration purposes.

# C1. Identification Information

1. Create an identifying comment within the first line of a file named “main.py” that includes your student ID. main.py screenshot goes here showing Student ID

A screen shot of a computer

Description automatically generated

# C[[1]](#footnote-1). Process and Flow Comments

Some code blocks screenshots go here showing comments

By explaining the intent and decisions of each “major” block of code, i.e., the “why, what, and how,” the comments should improve readability.

Provide a little more detail for any process that is unusual or complicated.

# D. Interface

Provide an intuitive interface for the user to view the delivery status (including the delivery time) 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.)

Interface screenshot goes here.

# A screenshot of a computer Description automatically generated

The initial text displayed to the user when starting the main script is shown above.

# D1. First Status Check

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

A computer screen with white text

Description automatically generated

The information for all packages at 8:45 am shows the various states the packages are in. Including a few packages that are already delivered, package #34, for instance.

# D2. Second Status Check

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

A computer screen with white text

Description automatically generated

This screenshot continues to show the interface displayed as packages are delivered. Displayed here is the package status at 10:00 am.

D3. Third Status Check

1. Provide screenshots to show the status of all packages loaded onto each truck at a time between 12:03 p.m. and 1:12 p.m.

A screen shot of a computer screen

Description automatically generated

Finally, this screenshot displays the package status for every package at 12:59 pm. Showing all packages delivered, and at what time.

# E. Screenshot of Code Execution

Provide screenshots showing successful completion of the code that includes the total mileage traveled by all trucks.

A screenshot of a computer

Description automatically generated

The above block shows the total mileage of the trucks being well below the 140 mile maximum.

Provide a screenshot or screenshots so that the evaluator can check that your code ran on your machine successfully to completion. The screenshot(s) should include a view of the console output, the project files, etc.

A computer screen with white text

Description automatically generated

The CLI working on my machine.

A computer screen shot of a computer code

Description automatically generated

A screenshot that includes part of the main.py file as well as the entire directory my project is in.

A screenshot of a computer program

Description automatically generated

Another screenshot of the main.py file with the terminal in view showing the working interface as well.

# F1. Strengths of the Chosen Algorithm

Justify the package delivery algorithm used in the solution as written in the original program by doing the following:

1. Describe two or more strengths of the algorithm used in the solution.

The first major strength of the nearest neighbor algorithm is its simplicity. It only takes a few lines of code to implement. This gives the algorithm a “good enough” solution to a problem. It also avoids a training step, like some other algorithms. This allows it to interpret data without any assumptions or needing any additional inputs before running.

# F2. Verification of Algorithm

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

The algorithm finds a near shortest path to deliver the packages on each truck. It uses the distance between points to find a route and sorts the packages on the truck by their package ID in that short order. All package constraints were noted during loading, which was done manually. The algorithm was completely responsible for finding the shortest route and updating times/destinations/mileage accordingly.

# F3. Other Possible Algorithms

3. Identify two other named algorithms that are different from the algorithm implemented in the solution and would meet all requirements in the scenario.

See above

Two other named algorithms include a Greedy algorithm and a Branch and Bound algorithm. The Greedy algorithm is very similar to the nearest-neighbor algorithm. It uses a method of finding the closest point from an arbitrary starting position. The Branch and Bound algorithm, on the other hand, uses a check system to compare the current optimal path with the next route, exchanging quicker version while throwing slower versions away.

# F3a. Algorithm Differences

a. Describe how both algorithms identified in part F3 are different from the algorithm used in the solution

The Greedy algorithm is similar to the nearest-neighbor approach. The main difference between the two is their scopes. A Greedy algorithm simply finds the next closest point from the current point and records it. The nearest neighbor algorithm can bolster its decision-making process with many additional inputs.

The Branch and Bound method creates an initial route and then explores many different versions of the route. Each time the algorithm calculates a new path it checks it against the current most optimal route. Whenever it finds the current path to be longer than the recorded optimal path it removes it.

G. Different Approach

Describe what you would do differently, other than the two algorithms identified in part F3, if you did this project again, including details of the modifications that would be made.

If I were to do this project again, I would pull more methods out of the main.py file to create a program that better aligns with an OOP mindset. When I initially started working on the project, I wanted to manipulate all of the package data within the package class. In practice, I was unable to find a method for printing the content of packages instead of their memory addresses. I also would like to include a GUI. I have lots of experience with Tkinter and feel I could create something much more appealing to the average user. A fresh take on the code might lend itself to faster approaches as well. In an attempt to clean the code up, I found myself looping within methods more times than I was sure I needed. Some specific things that I feel could be cleaned up and made more efficient:

• Line 96 of main.py, in the distance\_finder function. Instead of checking for an empty cell with the if statement, a better format to store that information could avoid it.

• Line 24 of package.py, in the status\_update method could include a Python equivalent to a switch case (I believe these are called match functions?).

• Lines 35 and 41 include both of the functions I made to extract CSV data. I’m not sure if they are needed but wanted to include means of future-proofing/expanding later. This is also why I used global variables for package number and truck speed. I thought the flexibility to use different trucks and package amounts was important.

# H. Verification of Data Structure

Verify that the data structure used in the solution meets all requirements in the scenario.

Address above questions with 4-5 sentences.

The hash map I’ve created correctly uses the packageID as a key and the entire package contents as its value. You can insert, delete, update, and search for entries by those keys. The package data includes all information found in the package CSV. The table uses a hash function to find a unique number for each key. That number is then divided by a determined amount, the length of our table in this project, and the remainder is recorded. That remainder is the index for the bucket in which the key and it’s data will be stored.

# H1. Other Data Structures

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

See above

Two other data structures that could meet the same requirements include a linked list and an array list. The array list is similar to a hash map, in which they both use keys as identifiers to later find stored data. A linked list does not user a (key, data) system. But rather a series of nonadjacent pieces of memory that connect to each other with identifiers called pointers.

# H1a. Data Structure Differences

a. Describe how each data structure identified in H1 is different from the data structure used in the solution.

See above

A linked list uses a system that doesn’t require data to be held in concurrent memory. Each piece of data is allocated memory that includes pointers from the previous node and to the next node. As the data is manipulated all that changes from node to node is the pointer’s destination.

An array list, like a hash map, uses keys that can be used to quickly locate data. Arrays use sequential indices to represent their key values, unlike the hash map, which allows for any value to represent a key.

1. . Include comments in your code to explain both the process and the flow of the program. [↑](#footnote-ref-1)