C950 Task Directions summary

Scenario

The Western Governors University Parcel Service (WGUPS) needs to determine the best route and delivery distribution for their Daily Local Deliveries. The Salt Lake City DLD route has three trucks, two drivers, and an average of 40 packages to deliver each day; each package has specific criteria and delivery requirements.

Your task is to write code that determines and presents a solution delivering all 40 packages, listed in the attached "WGUPS Package File," on time, according to their criteria while reducing the total number of miles traveled by the trucks. The "Salt Lake City Downtown Map," provides the location of each address, and the "WGUPS Distance Table" provides the distance between each address.

The supervisor (user) should be able to check the status of any given package at any given time using package IDs, including the delivery times, which packages are at the hub, and which are en route. The intent is to use this program for this specific location and to use the same program in different cities as WGUPS expands its business. As such, you will need to include detailed comments, following the industry-standards, to make your code easy to read and justifying the decisions you made while writing your program.

Project Summary

You will write a program in Python, which does the following:

- Stores the package information into a hash table.
- Uses a *self-adjusting* heuristic algorithm to find a solution that delivers all packages using under 145 miles and according to the provided requirements (e.g., delivery deadlines, addresses, number of trucks, special notes, etc.).

    Note: Non self-adjusting methods can also be used, but there must be at least one part that uses a self-adjusting algorithm (identified in Part A) with one scalable element (discussed in B.4).
- Allows the 'user' to check the status (at the hub, en route, or delivery time) of any package at any given time.

Using code comments and a separate document, complete the following:

- Provide code comments for every major portion of code that explains the logic and time-complexity.
- Provide documentation on algorithms used, alternative methods to, efficiency, and scalability of the methods used to find a solution.
- Provide documentation on the data structure(s) describing their use, alternative methods, efficiency, and scalability of processes used to store data.

Technical Requirements:

- Evaluators need to successfully run your code using only submitted files and the most recent version of Python. Submitted code may use anything from Python's standard library, including

the built-in data structures (e.g., lists, tuples, sets, and dictionaries), except for the hash table where only the use of dictionaries is prohibited.

- The project must use the package and distance data provided in files WGUPS Distance Table.xlsx WGUPS Package Table.xlsx found in 'View Task>Task Overview>Supporting Documents' on the C950 COS page. You may make organizational changes and convert these documents into other formats, e.g., '.csv'.
- Code downloaded from the internet or acquired from another student or any other source may not be submitted and will result in automatic failure of this assessment.

Recommendations:

- The IDE PyCharm is recommended. You can simply zip and upload your project folder when you submit your project.
- Organize your document so that it specifically addresses the requirements marked less than competent. For example:

  Typically 2-3 sentences suffice per section. This way makes it easier to write, grade, and fix if necessary. Mention everything in the rubric requirements even if it's redundant or superfluous. The coding/annotations sections of the directions are only suggestions.
- Avoid using the provided Sample Algorithm Overview.docx as a template for your submitted document. It provides a pseudocode example for one way (but not the only way) to satisfy Part B1 (see below).
- The supporting data files, WGUPS Distance Table.xlsx and WGUPS Package Table.xlsx are best viewed in MS Excel. However, there are many alternatives for those without Excel. LibreOffice is freely available and runs in Windows, Mac, and Linux.
- Watch this C950 webinar: Getting Started on the project
- Make an appointment with a CI, send us an email, and read the course chatter.

Assumptions

- Two drivers and three trucks are available. So no more than two trucks can be away from the hub at the same time.
- The trucks move at a constant speed of 18 miles per hour.
- Trucks can carry a maximum of 16 packages.
- Trucks can leave the hub no sooner than 8:00 a.m.
- Packages can only be loaded onto a truck at the hub.
- You only need to account for the time spent driving. You can ignore the time spent on all other activities, such as loading trucks and dropping off packages.

- The wrong delivery address for package #9, Third District Juvenile Court, will be corrected at 10:20 a.m. The correct address is “410 S State St., Salt Lake City, UT 84111”. You may assume that WGUPS knows the address is incorrect and when it will be corrected.
- Packages #13, #14, #15. #16, #19, and #20 must go out for delivery on the same truck at the same time.
- Packages #3, #18, #36 and #38 can only be loaded on truck 2.
- #6, #25, #28, #32 cannot leave the hub before 9:05 a.m.

Rubric Requirements Summary

Note: The sections below have paraphrased or restated some of the official rubric requirements for better readability and alignment with general evaluation practices. The official rubric requirements are located on your C950 COS page.

A: ALGORITHM SELECTION

Identify the name of the algorithm used to create a program to deliver the packages and meet all requirements specified in the scenario.

You must identify the self-adjusting part of your algorithm by name. Though you may have written your own algorithm using a variety of approaches, you should be able to connect or generalize part of your algorithm to a commonly recognizable method, e.g., a greedy algorithm, farthest neighbor algorithm, etc.

The identified (also called chosen or core) algorithm must be *self-adjusting* (inferred from the scenario). Non self-adjusting methods can also be used to find a solution, but there must be at least one part that uses a self-adjusting algorithm.

B1: LOGIC COMMENTS

The comments accurately explain the logic applied to the solution.

Provide a “what, how, and why“ summary explanation of how your code finds a solution, i.e., explain how your core algorithm works. You can do this in your document or code though the former is recommended. Pseudocode (following the example found here sample algorithm document) is one way to fulfill this requirement, but you certainly need a step by step explanation (via pseudocode or bullet points) outlining how your code finds a solution.

B2: APPLICATION OF PROGRAMMING MODELS

Provide a description of the communications protocol used to exchange data, the target host environment used to host the application server application, and the interaction semantics defined by the application to control connect, data exchange, and disconnect sequences.

It appears this project originally intended to read data from a server (see programming models). However, the current project requires all data files to be stored on the same machine hosting the application, i.e., the .csv files should be stored in your project folder.

You should include a description of the programming language and environment used. Because everything is run on the same local machine, there is no communication protocol or defined interaction semantics. However, as both are listed in the rubric, you should explain why they are not needed.

B3: SPACE-TIME AND BIG-O

The evaluation shows the space-time complexity using Big-O notation for each major block of code and the entire program.

B4: ADAPTABILITY

The discussion includes the chosen algorithm's ability to handle a growing number of packages (scalability).

The identified (chosen) algorithm from Part A must be self-adjusting. Hence it will have at least one scalable element. This discussion can also include shortcomings, e.g., "X will not scale well because…"

B5: SOFTWARE EFFICIENCY AND MAINTAINABILITY

The discussion addresses how the software's efficiency and why it is efficient and easy to maintain.

By "efficiency", they mean the Big-O time complexity of your entire program. Your program must run in polynomial time or better to be efficient.

By "maintainability", they mean how the entire program can be easily understood, repaired, and enhanced by developers other than the author -*software* maintainability.

B6: SELF-ADJUSTING DATA STRUCTURES

The discussion of the self-adjusting data structure(s) includes the ability of the data structure(s) to adapt when data is inserted, removed, or accessed; and how that adaptation affects running time.

At least self-adjusting one data structure needs to be discussed. The hash table discussed in D, D.1, and E can fulfill this requirement.

C: ORIGINAL CODE

The original code runs properly and delivers all packages within their requirements while adding the least number to the combined mileage total of all trucks in less than 145 miles.

It should be made clear that all the requirements are met, and the total mileage is less than 145. For the milage, the program should provide the user with the total mileage.

Per requirement F and G, the user should be able to check the status (at the hub, en route, or delivered at time X) of all the packages at any given time. For example, the user should be able to provide the time 10:47 and see a printout of every packages' status and info (listed in part F). Using this functionality, the evaluator can verify that your delivery solution is valid.

C1: IDENTIFICATION INFORMATION

The initial comment is located within the first line of code and includes the candidate's first name, last name, and student ID.

Include these comments in whatever is most easily identifiable as your "main" file. Easy to satisfy, but easy to forget!

C2: PROCESS AND FLOW COMMENTS

Include comments at each major block of code explaining the process and flow of the code.

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: DATA STRUCTURE

Identify a data structure written by the student using only primitive data structures, lists, tuples, or sets, used by your program to store and retrieve package data.

This data structure must be same the hash table in parts E and F. The official task directions include a note:

> "Do NOT use any existing data structures..."

This should read, "use only built-in data structures." Submitted code may use anything from Python's standard library, including the built-in data structures (e.g., lists, tuples, sets, and dictionaries). The only exception is the hash table, where the use of dictionaries is prohibited.

Per parts E and F, the hash table is required to have the following:

- E: an insertion function that includes as input all a package's info (see below).
- F: a look-up function that includes as input a package's ID and returns the corresponding package's info (see below).

The abilities to store and retrieve package info (via the package's ID) are the only requirements. The information can be stored in an object and can include additional parameters, e.g., special notes, time the package left the hub, etc.

The insert function (Part E) and look-up function (Part F) must respectively store and retrieve the following information:

- package ID number
- delivery address
- delivery deadline
- delivery city
- delivery zip code
- package weight
- delivery status (at the hub, en route, or delivery time)

D1: EXPLANATION OF DATA STRUCTURE

The submission accurately explains the data structure and how that data structure accounts for the relationship between the data points to be stored.

Provide an explanation that describes the logic of the hash table and how it is used in the context of solving the problem.

E: HASH TABLE

The hash table has an insertion function that includes, as input, all of the given components.

The provided hash table must include an insertion function which can insert all the packages info (see part D) into the hash table.

NOTE: Submitted code may use anything from Python's standard library, including the builtin data structures (e.g., lists, tuples, sets, and dictionaries), except for the hash table where only the use of dictionaries is prohibited.

F: LOOK-UP FUNCTION

The look-up function includes all of the given data elements, completes searches via package ID, returns the data corresponding to the provided ID including the package' status (at the hub, en route, or delivery time).

The provided hash table should include a look-up function which can use a package's ID to retrieve all the packages info (see part D) from the hash table.

NOTE: Submitted code may use anything from Python's standard library, including the builtin data structures (e.g., lists, tuples, sets, and dictionaries); except for the hash table where only the use of dictionaries is prohibited.

G: INTERFACE

Provide an interface for the user to view the status and info of any package at any time.

For example, the user should be able to look up package #19 at 10:43 am and check the info and status. Having the user provide a time and printing the info and status of all the packages will meet this requirement. It is acceptable to use the command line, but it needs to clarify how the user can view the package info and statuses for a given time.

G1-G3: 1st, 2nd, and 3rd status checks.

Provide screenshots showing the info (outlined in part F) and statuses at a time between:

- *8:35 a.m. and 9:25 a.m.*
- *9:35 a.m. and 10:25 a.m.*

- *12:03 p.m. and 1:12 p.m.*

Provide one screenshot within each of the time intervals above. The screenshot can be included anywhere in your submission, e.g., the document, separately, in the project folder.

H: SCREENSHOTS OF CODE EXECUTION

Provide a screenshot or screenshots showing successful completion of the code free from runtime errors or warnings.

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. The screenshot can be added anywhere in your submission, e.g., the document, separately, in the project folder.

I1: STRENGTHS OF THE CHOSEN ALGORITHM

The description includes at least two specific strengths of the chosen algorithm as they apply to the scenario.

"Chosen algorithm" refers to the core algorithm identified in Part A.

I2: VERIFICATION OF ALGORITHM

The verification includes the total miles added to all trucks, and it states that all packages were delivered on time.

You must verify the total miles traveled by all the trucks is under 145 miles and state that all packages were delivered on time and according to their constraints. Evaluators should be able to verify the mileage and deliveries via the user interface. So the total mileage must be provided via the user interface or console output.

I3: OTHER POSSIBLE ALGORITHMS

The submission identifies two other algorithms that could meet the requirements of the scenario.

The two alternative algorithms should be different than the algorithm identified in Part A

I3A: ALGORITHM DIFFERENCES

The description includes attributes of each algorithm identified in part I3 and how the identified attributes compare to the attributes of the algorithm used in the solution.

The two alternative algorithms should be compared to the algorithm identified in Part A.

J: DIFFERENT APPROACH

The description includes at least one aspect of the process that the candidate would do differently and includes how the candidate would modify the process.

K1: VERIFICATION OF DATA STRUCTURE

The verification shows all the criteria have been met: the least number of total miles added to all trucks, all packages were delivered on time, the hash table with look-up function is present, and the reporting needed is accurate and efficient.

This section should be titled "Verification of Data Structure and Solution." Provide evidence that the hash table (identified in Part D) and delivery solution meets the following criteria:

- Completes in 145 miles or less.
- Delivers all packages on time and according to their constraints outlined in the package notes.
- Uses a hash table with a look-up function as described in Part F.
- Reports on package statuses and mileage accurately and in a user-friendly manner.

This requirement is redundant to previous requirements. Reciting evidence or citing previously written sections is acceptable.

K1A: EFFICIENCY

The description of the efficiency of the data structure (hash table) used in the solution includes what type of data is being used and how that data is being used.

The "data structure used in the solution" refers to the hash table identified in Part D. Describe the type of data used and how your program uses that data. Provide a justification that the hash table makes the program more efficient.

K1B: OVERHEAD

The explanation of the data structure (hash table) includes the computational time, memory, and bandwidth aspects when handling data in this program.

Computational time means time-complexity for handling data. This will include reading, storing, and finding package info from your hash-table (there will be some redundancy here from Part K1A). Bandwidth and memory aren't much of a concern as everything is run from a local machine. However, just as with "communication protocol" in Part B2, you need to mention it and explain why as it's in the rubric.

K1C: IMPLICATIONS

In regards to the data structure (hash table), describe the implications when more packages are added to the system or other changes in scale occur.

Describe how adding more packages, cities, or trucks impacts your hash table performance (scaling). The discussion can include possible shortcomings.

K2: OTHER DATA STRUCTURES

The submission identifies two data structures other than the one used in the solution that meets the criteria and requirements in the scenario.

"one used in the solution" refers to the data structure in Part D. Identify two alternative data structures and justify that they could be used for the project.

K2A: DATA STRUCTURES DIFFERENCES

The description includes the attributes of each data structure identified in part K2 and compares these attributes to the attributes of the data structure used in the solution.

L: SOURCES

The submission includes in-text citations for sources that are properly quoted, paraphrased, or summarized and a reference list that accurately identifies the author, date, title, and source location as available.

You must follow (APA standards). Contact the writing center for questions or help with this portion. If no sources were used, include a sources section in your documentation explaining that no sources were used. Every source listed must have a matching in-text citation; sources not warranting an in-text citation should be excluded. Code sources should be included in code comments near their citation.

M: PROFESSIONAL COMMUNICATION

The content reflects an attention to detail, is organized, and focuses on the main ideas as prescribed in the task or chosen by the candidate. Terminology is pertinent, is used correctly, and effectively conveys the intended meaning. Mechanics, usage, and grammar promote accurate interpretation and understanding.

The submitted document should be grammatically correct and easy to read. Make use of one of the many freely available grammar checkers and/or the writing center.

C950 Performance Assessment FAQ

Do I have to use Python?
Yes. The project requires that you use a version compatible with the latest version of Python 3.

Am I required to use a specific IDE?
You can use whatever IDE you like, but we recommend the free community version of PyCharm.

Do all packages have to be delivered on time?
Yes. Packages must be delivered on or before their delivery time. Packages with an "EOD" deadline must be delivered before the "End Of Day." "EOD" is not specified, but it's reasonable to assume that it's 5:00 pm. However, solutions are almost always completed before 3:00 pm.

Why are there three trucks?
Only two drivers are available, and trucks have unlimited gas and load instantly. Perhaps, a third truck makes instantaneous load times realistic (truck 3 can be loaded while the others are out). However, it is no required to use truck 3 (or truck 1).

What is the maximum total mileage allowed for trucks?
Less than 145 miles. This is what the official task directions mean by "optimal." Both heuristics and optimizing algorithms are important to get a good solution.

Does the solution have to be absolutely optimal?

The delivery problem is a variation of the traveling salesperson problem (TSP), which is known to be NP-Hard. So even if you found the optimal solution, we couldn't verify it without checking every possible route (and we don't have enough time for that!). Furthermore, the task requires your algorithm to be "efficient," meaning it must run in polynomial time. If you find a polynomial-time algorithm for solving the TSP, please let us know as this would answer the P versus NP problem -a millennium problem with a one million dollar prize.

Therefore, you should use an algorithm that finds a solution that performs sufficient optimization. For this, there are many approaches resulting in far less than 145 miles.

Do I need to use any algorithms not covered in the learning materials?
No, but you certainly can. However, a program using only the covered algorithms (plus some heuristics) can find an adequately optimal solution.

Can I use external third-party libraries?
No. Only submit code that you've written yourself or from the standard built-in Python library. Evaluators will need to run your code on their machines without installing additional resources.

Can I use the built-in Python dictionary construct for my hash table?
No. Your hash table can use lists, sets, or tuples. You can use dictionaries elsewhere in your project.

Does my program have to parse the excel data straight from excel?
No. You can export the data from excel to text or CSV file. Include these files with your submitted project. You can clean up the file manually, making it easier to import, e.g., removing headers.

The Excel mileage matrix only has the lower half filled in, so it only has mileage entries in one direction, for example from point A to point B, but not from point B to point A. Does that mean a truck has to find an alternate route?
No. The distance table is bi-directional, i.e., the distance from A to B equals the distance from B to A. Furthermore, the distance table is a fully connected graph, i.e., it provides a direct path between every address. So using a graph data structure is not necessary.

However, the triangle inequality does not hold. Meaning the provided distance from A to B mileage might not always be the shortest path! For example, from the distance table: hub-->Taylorsville is 11.0, but hub--> Valley Regional--> Taylorsville is 6.4+0.6 =7 miles. You can optimize the distance table via a pathfinding algorithm, such as Dijkstra's. However, this step is not necessary to find a solution under 145 miles.

Does my program have to use any specific "communications protocol"?
It appears this project originally intended to read data from a server (see programming models). However, the current project requires all data files to be stored on the same machine hosting the application, i.e., store the source data files in your project folder.

So in the traditional sense, your program will not need a communication protocol as it does not communicate with another machine (see B2 above). However, your program will communicate with different parts of your program, e.g., using method parameters and global shared data structures. These processes can be interpreted as a communication protocol.

Does my program need to have an interactive external user interface?
Yes. See Part G above. The 'user' needs to conveniently be able to check the status of any package at any time. For example, the user should be able to look up package #19 at 10:43 am and check the info and status. Having the user provide a time and printing the info and status of all the packages will meet this requirement. We recommend a simple command-line interface.

Does the user need to be able to update the package's address?
There is no requirement for this, and the code should be able to complete the delivery simulation using only the provided data.

How do we handle the package with the wrong address (#9)?
The wrong delivery address for package #9, Third District Juvenile Court, will be corrected at 10:20 a.m. The correct address is "410 S State St., Salt Lake City, UT 84111". There is no specification for handling this situation other than package #9 cannot be delivered to the correct address before 10:20 a.m. It is acceptable to assume WGUPS knows the address is incorrect and when it will be corrected. Hence effectively treating package #9 like a delayed package.

How do we handle packages with the special notes:

- Package #14: "Must be delivered with 15, 19"
- Package #16: "Must be delivered with 13, 19"
- Package #20: "Must be delivered with 13, 15"

Packages #13, #14, #15. #16, #19, and #20 must go out for delivery on the same truck at the same time, e.g., those packages all leave the hub at 9:30 on truck 2.