Project Overview

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.

The self-adjusting algorithm that will be used to create the program to deliver the packages is "Dijkstra's Shortest Path Algorithm."

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

1. Explain the algorithm’s logic using pseudocode.

1. hubs\_to\_deliver := Read hub data and Initialize hubs List

2. indexed\_packages := Read package data and initialize indexed\_packages HashMap

3. graph := Initialize graph

4. trucks := Initialize trucks List with Truck objects.

5. business\_rules := Initialize business\_rules HashMap

6. For each loaded truck with packages as long as there is sufficient time to deliver.

7. truck.packages := Sort truck’s packages by package.delivery\_time

8. For each package loaded in the truck:

9. Hubs\_to\_deliver := Get the hub associated to the packages address

10. start\_loc := get starting location—WGU address—from graph.adjacency\_list.

11. current\_location := initialize with start\_loc

12. current\_time := initialize with a provided begin\_time or default\_start\_time for a shift (i.e. 8:00 am)

13. retry\_hubs := initialize retry queue with empty List[Hub]

14. packages\_delivered := initialize list of delivered packages List[Hub]

15. remaining\_time := initialize by calculating the end\_time – start\_time

16. While there are hubs to deliver and the remaining time is greater than 0:

17. next\_hub, distance := find the nearest hub to deliver the package to

18. seconds\_to\_drive\_to\_next\_hub := calculate the seconds to drive the next hub.

19. If the seconds\_to\_drive\_to\_next\_hub is less than the remaining\_time:

20. For each package in the truck that is yet to be delivered:

21. If the package is deliverable according to its business rules

22. Update miles driven by truck

23. Update the truck’s delivery clock

24. Update remaining time

25. Delivery package

26. Else validation flag is updated to false

27. If validation passes update the current\_location with the next\_hub

28. Else append the failed address hub, to the retry\_hubs list and reset validation flag to True.

29. If retry\_hubs list is not empty and there are no more hubs in hubs\_to\_deliver list

30. Reload the hubs\_to\_deliver list with the retry\_hubs list.

31. Else update truck.status and return current\_time

32. Update truck.status and reset the packages\_delivered List.

33. Return current time.

34. ­­­­­­­­­Print delivery report with package status and truck status.

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

The programming environment used to create the Python application is the **PyCharm CE** editor with **Python 3.8** on an **Apple MacBook Pro 2021** with an **M1 Apple silicon chip** and **32 GB** of memory running **macOS Ventura 13.3.1**

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

The space-time complexity of each major segment of the program and the entire program using big-O notation is as follows:

1. The time complexity of Dijkstra's algorithm is O((N + E) log N), where N is the number of hubs/vertices and E is the number of edges in the graph.
2. The time complexity of reading the package data file is O(n).
3. The space complexity of storing the package data in the hash table is O(n).
4. Explain the capability of your solution to scale and adapt to a growing number of packages.

The solution can scale and adapt to many packages because Dijkstra's algorithm can handle large graphs efficiently. As the number of packages increases, the algorithm will still find the shortest paths to deliver the packages efficiently.

1. Discuss why the software is efficient and easy to maintain.
2. The program is efficient due to using hash tables for fast package lookup, reducing the time complexity of package retrieval operations. Additionally, the algorithm optimizes package delivery by considering nearest neighbor locations, minimizing the distance traveled by the truck. Considering the worst-case scenario, where each hub is connected to every other hub, the overall time complexity of the algorithm becomes O(H^2). This means that the execution time of the algorithm grows quadratically with the number of hubs in the graph where H is a hub in the graph. The algorithm's efficiency remains regardless of the number of hubs added to the `Distances.csv` file.
3. The program is easily maintainable in several aspects; it uses a constants.py file that can be regarded as a configuration file for the project. This file can be updated to address an increase in speed on the trucks, a change in package capacity, and even the package numbers to be delivered by each truck and the business rules to follow before delivering a package to a hub. The program is highly modular, well commented, and aims to separate concerns to encourage the addition of more business logic if required. Furthermore, the program is executable in several operating systems as it uses native python3.8, and it’s not bound to a specific platform making it highly portable.
4. Discuss the strengths and weaknesses of the self-adjusting data structures (e.g., the hash table).

Strengths of this HashMap data structure:

1. Efficient Key-Value Storage: The **HashMap** class provides an efficient way to store key-value pairs, allowing for fast retrieval and insertion of elements.
2. Operations are average Constant-Time: In an ideal scenario with a well-distributed hash function and a proper load factor, the average time complexity for operations like **add**, **get**, **remove**, and **contains** is constant-time (O(1)), providing efficient access to data.
3. It has flexible key types: The **HashMap** class supports various keys, including objects, primitives, and custom data types, allowing for diverse usage.

Weaknesses:

1. No dynamic resizing: The current implementation of the **HashMap** class does not include dynamic resizing. If the number of elements exceeds the initial capacity, the load factor increases and may result in more collisions and degraded performance.
2. No load management: Without dynamic resizing or load factor management, the **HashMap** class does not automatically resize or rehash the elements to maintain a balanced factor. This results in inefficient storage and retrieval operations when the number of elements grows significantly.

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.

A picture containing text, font, screenshot, algebra

Description automatically generated

1. Include comments in your code to explain the program's process and flow.

*This is the starting comment/entry point for the main.py file where comments begin. (please see the program’s code for further comments).*

A picture containing text, screenshot, font, document

Description automatically generated

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.

The self-adjusting data structure used in the program is a hash table to store the package data. The package ID is used as the key to access the package data. Package data comprises the package ID, address, city, state, zip code, delivery time, weight, status, and relevant notes. These notes will be used in a separate hash table to define the `business rules` to dictate requirements before a package can be delivered. The hash table enables fast access to package data with a constant time complexity of O(1).

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)

**(please see program)**

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

**(please see program)**

G.  Provide an interface for the user to view any package's status and info (as listed in part F) 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.

"screenshots/EOD\_report\_option1.png" is a screenshot depicting choice 1, "Display report of the end-of-day," where the trucks are loaded and sent on their routes to deliver all packages to their respective addresses and ultimately arrive back to the parent hub.

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

"screenshots/EOD\_report\_option2\_840.png, screenshots/EOD\_report\_option2\_1015.png, screenshots/EOD\_report\_option2\_1312.png" are screenshots displaying, "Display summary report for a given time," where the user is asked to enter a specific time between 8 and 5 pm.

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

"screenshots/EOD\_report\_option3\_830\_930\_1130\_package\_9.png" is the last screenshot for option 3, where a user can pick a specific time and package. The input is validated, and the output results in the full status of the package at the provided time by the user.

H.  Provide a screenshot or screenshots showing the successful completion of the code, free from runtime errors or warnings, that includes the total mileage traveled by *all* trucks.

A screenshot of a computer screen

Description automatically generated with low confidence

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.

Strengths of Dijkstra's Shortest Path Algorithm:

**a**. Optimal Solution: Dijkstra's algorithm guarantees to find the shortest path from a given source node to all other nodes in the graph. It ensures that the delivery trucks will take the most efficient routes to deliver the packages, minimizing the total distance traveled.

**b**. Scalability: Dijkstra's algorithm can handle large graphs efficiently. As the number of packages or delivery locations increases, the algorithm's time complexity remains relatively stable, making it suitable for scaling to accommodate a growing number of packages.

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

Dijkstra's algorithm meets all the requirements in the scenario. It calculates the shortest paths from the hub (source node) to all delivery hubs (other nodes) in the graph, ensuring efficient routes for the trucks. The algorithm considers the distances provided in the distance table, taking into account each package's specific delivery constraints and deadlines. Finding the shortest paths allows the trucks to deliver all 40 packages on time while keeping the combined total distance traveled under 140 miles for both trucks. In this particular program, the total distance traveled was 106.5 miles which can be observed in the console output upon running the program.

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

**a**. A\* Search Algorithm: A\* search is an informed search algorithm that combines elements of Dijkstra's algorithm and heuristic functions. It uses heuristics to guide the search toward the goal node, improving the algorithm's efficiency by considering estimated distances. In the scenario, A\* search could be applied to find the optimal routes for the delivery trucks while considering the actual and estimated distances based on heuristics.

**b**. Uniform Cost Search (UCS) is another search algorithm that’s similar to A\* in which the search is based on the cost of reaching each node/hub. UCS could be applied to this program, similarly, by locating our starting hub and goal hub for which we want the shortest path. Upon finalization of UCS, we should have the shortest path between starting hub and the goal hub.

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

1. One of the main differences between Dijkstra and A\* is that A\* uses a heuristic function. This heuristic guides the search by prioritizing nodes that are likely to lead to the goal. On the other hand, Dijkstra's algorithm does not consider any heuristic and treats all nodes equally. In short, A\* is an improvement upon Dijkstra’s as per the aforementioned.

2. One of the key differences between Dijkstra’s and UCS is that Dijkstra’s assumes we use only non-negative weights, while UCS doesn’t assume non-negative weights. Instead, It considers the cumulative path cost from the start node to each node and selects the node with the lowest path cost at each step.

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

There are definitely many things that I would consider doing differently.

* 1. I would add a priority queue to Dijkstra’s algorithm rather than an array to hold the unvisited queue. This would allow the algorithm to become more performant.
  2. I would also want to take advantage of external libraries such as dataclasses to create less opinionated models, pandas to be able to extract information directly from the .xslx files that were provided initially, and graviz to be able to visualize graphs paths and convert them to a pdf.

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.

It provides fast insertion and lookup operations with a constant time complexity of O(1). The package ID is the key to efficiently accessing the corresponding package data.

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

The time needed to complete the lookup function is unaffected by changes in the number of packages to be delivered. Regardless of the number of packages, the hash table allows direct access to the desired package information using the package ID as the key.

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

The data structure space usage increases linearly with changes in the number of packages to be delivered. As each package is stored as a key-value pair in the hash table, the space required will grow proportionally to the number of packages.

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.

Changes in the number of trucks or cities would not significantly affect the lookup time or the space usage of the hash table data structure. The hash table's constant time complexity ensures efficient lookup regardless of the number of trucks or cities. The space usage will still scale linearly with the number of packages, as each package requires storage in the hash table.

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

a. Binary Search Tree (BST): A BST is a sorted binary tree data structure where each node's key is greater than all keys in its left subtree and smaller than all keys in its right subtree. In the scenario, a BST could be used to store the package data sorted by package ID. This would allow efficient lookup operations with a time complexity of O(log n), where n is the number of packages. However, the space usage would still be linear, like the hash table.

b. Linked List: A linked list can also be used to store key-value pairs, where each node in the list represents a key-value pair. In this case, we would also be using the package ID to map to the package data. We would have to traverse this structure linearly in order to access its data. At best, O(1) and at worst, O(n), where n is the number of packages

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

1. BST vs. HashMap implementation:

\* BST has ordered format storage, while a HashMap relies on a hashing function to determine the bucket it’ll reside on.

\* BST has comparison-based operations for searching, insertion, and removal, as these rely on comparing keys using BST’s ordering properly while

\* BST relies on being balanced. An unbalanced tree affects look-up performance, deletion, and insertion.

2. Linked List vs. HashMap implementation:

\* Linked List is organized sequentially from node reference to a node

\* Linked list access time is linear, where we start at the beginning and traverse until we reach the desired node. This could result in O(n), where n is the number of elements in the list. Granted, a Hash table that doesn’t have dynamic resizing implementation can end up with a high number of collisions where its time complexity can increase to O(n).