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

🡪 The deliver\_packages function uses the nearest neighbor algorithm, which offers several strengths in the context of package delivery:

**Simplicity and Intuitiveness:**

The nearest neighbor algorithm is conceptually simple to understand and implement. At each step, the algorithm just selects the closest unvisited node (in this case, the nearest undelivered package) as its next target.

This simplicity makes it easier to debug, modify, and maintain as compared to some more complex optimization algorithms.

**Fast Execution for Smaller Input Sizes:**

When dealing with a relatively small number of packages (or nodes in general), the nearest neighbor algorithm can be quite fast and efficient. This is because, for each package, it only needs to evaluate distances to the remaining undelivered packages, and as deliveries progress, the number of undelivered packages decreases.

This can be particularly useful for real-world applications where the number of packages on a truck isn't very large, ensuring timely delivery decisions.

**Locally Optimal Choices:**

By always selecting the nearest package, the algorithm minimizes local distances. This could lead to relatively short routes for many scenarios, ensuring that trucks don't make long detours when nearby deliveries are available.

This local optimization can often result in good (though not always optimal) solutions for the Package Delivery system especially when a quick solution is needed.

**Adaptability:**

The nearest neighbor algorithm can be easily adapted to changing conditions. If a package is added or removed, or if certain roads become inaccessible, the algorithm can be rerun without significant overhead, providing a new delivery order based on the updated information.

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

🡪 The algorithm used viz Nearest neighbor does take care of all the requirements in the scenario and also works with the specified assumption for the project.

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

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

🡪 **1. Dijkstra's Algorithm**

*a.* *Description:* Dijkstra's algorithm is primarily designed to find the shortest path between a starting node and all other nodes in a weighted graph. It starts with the source node and iteratively selects the next unvisited node with the smallest known distance from the source. The algorithm then updates the distances of the neighboring nodes based on the newly visited node.

*b. Difference from Nearest Neighbor Algorithm:*

*Objective:* While the nearest neighbor algorithm's main goal is to find the next closest node from the current position without considering the overall shortest route to traverse all nodes, Dijkstra's objective is to determine the shortest path from a given start node to all other nodes in the graph.

*Scope:* Dijkstra's algorithm provides a global view by accounting for the overall layout of the graph, whereas the nearest neighbor algorithm takes a localized view by considering only the current node and its immediate neighbors.

*Order of Visiting Nodes:* The nearest neighbor algorithm continually selects the closest unvisited node to the current node until all nodes have been visited. In contrast, Dijkstra's algorithm selects nodes based on global shortest distances from the start node.

**2. Greedy Algorithm:**

*a. Description:* A greedy algorithm makes the locally optimal choice at each step, hoping to find the global optimum. It doesn't evaluate the overall consequences of a choice but simply picks the best option available at the current step. For the package delivery problem, a greedy algorithm might work by selecting the nearest unvisited location from the current location at every step until all locations are visited.

*b. Difference from Nearest Neighbor Algorithm:*

*Local vs. Global Optimality:* Both the nearest neighbor and the greedy algorithm aim for local optimality. However, the difference is in the nature of the problems they typically address. The nearest neighbor is a type of greedy algorithm, specifically tailored for problems like the Delivering Packages Problem.

*Generalizability:* The greedy algorithm is a more general strategy that can be applied to a wide range of problems beyond just the package delivery scenario, such as coin change problems, fractional knapsack problems, and more. The nearest neighbor algorithm, as implemented, is specific to problems where one must visit a set of nodes and return to the start.

*Solution Quality:* Depending on the problem and the specific greedy criteria chosen, the solution obtained by a generic greedy algorithm might differ in quality from that obtained by the nearest neighbor approach.

G. 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.

🡪 **Incorporate Real-world Mapping APIs:**

Instead of relying on predefined distances, integrating real-world mapping APIs like Google Maps or OpenStreetMap would provide more accurate and updated distance and route information. This would also help account for real-world issues like traffic conditions, road closures, or construction.

**Advanced Package Prioritization:**

Packages could be prioritized not only based on delivery deadlines but also on other factors like package size, weight, or special handling requirements. This could lead to a more efficient packing and delivery strategy.

**Dynamic Truck Allocation:**

Instead of statically allocating packages to trucks, the system could be designed to dynamically allocate packages based on current truck location, available space, and pending deliveries. This would lead to a more flexible and adaptive system, especially useful in unforeseen circumstances.

**Expandability:**

Designing the system with scalability in mind would be crucial. As the service grows, more hubs, trucks, and delivery areas might be added. The system should be robust enough to handle such growth without major overhauls.

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

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

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

🡪 The data structure primarily used in the provided solution is a hash table (***HashMap***).

Hash Table (as used in the solution):

**Purpose:** This is used for efficient O(1) average-time complexity lookups and insertions of packages based on their package ID.

**Strengths:** Quick lookup times, ability to directly access objects based on a unique key.

**Requirements Met:**

Efficient retrieval of package data using package ID.

Dynamically storing package data as they are read from the CSV.

Adjusting or updating package data if needed (like delivery status, location, etc.).

**Two other data structures that could meet the same requirements in the scenario:**

1. **Balanced Binary Search Tree (BST) (e.g., AVL Tree or Red-Black Tree):**

**a. Differences from Hash Table:**

**- Nature of Storage:** In BST, data is stored in a tree structure, where each node has a key, and data associated with that key, and up to two children. The left child always has a key less than its parent, and the right child always has a key greater.

**- Lookup Time:** O(log n) for balanced BSTs compared to average O(1) for hash tables.

**- Order Preservation:** BSTs inherently maintain the order of elements, allowing for operations like "find the next package" or "find the previous package" which is not straightforward in hash tables.

**- Memory:** Might use slightly more memory than a hash table due to storage of tree node pointers.

**- Balancing:** Balanced BSTs like AVL Trees and Red-Black Trees require mechanisms to ensure the tree remains balanced after insertions and deletions.

1. **Array/List:**

**a. Differences from Hash Table:**

**- Nature of Storage:** A contiguous block of memory storing package objects sequentially.

**- Lookup Time:** O(n) for general lookups. If sorted by package ID, binary search could reduce this to O(log n).

**- Insertion Time:** O(1) if added to the end but O(n) in the worst case if maintaining a specific order.

**- Dynamic Resizing:** If using a dynamic array (like Python's list), it may occasionally need to be resized, which could be a time-consuming operation.

**- Order Maintenance:** A list would maintain the order of insertion, and if required, can be sorted based on any attribute of the packages.

**- Memory:** Might be more memory-efficient for storing the package data than BST but would need resizing mechanisms if it's dynamic.