### **F1. Two or More Strengths of the Algorithm**

The Greedy Nearest Neighbor algorithm provides notable strengths for solving the WGUPS routing problem. First, it is highly efficient in short-distance optimization. By always selecting the next closest location to visit, it significantly reduces travel time and mileage, making it ideal for local delivery scenarios where time is critical. Second, it has low computational overhead. Its simple logic enables fast execution, which is especially beneficial for small datasets like the 40-package scenario, which allows the program to respond quickly while still accounting for constraints such as deadlines and grouped deliveries.

### **F2. Verification the Algorithm Meets All Requirements**

The implemented algorithm satisfies all of the scenario’s delivery constraints. It ensures all 40 packages are delivered, and their status is updated accurately with timestamps indicating whether a package is at the hub, en route, or delivered. Delivery deadlines are honored by prioritizing packages with earlier deadlines and grouping them accordingly. Truck capacity is never exceeded, maintaining the 16-package limit per truck. Total mileage remains below 140 miles by leveraging efficient routing and making strategic use of the two drivers and three trucks. The special condition for Package 9, which requires address correction at 10:20 a.m., is handled properly within the program’s logic.

### **F3. Two Other Named Algorithms That Could Meet All Requirements**

Two alternative algorithms that could satisfy the delivery constraints of the WGUPS routing program are Dijkstra’s Algorithm and the *A (A-Star) Search Algorithm*\*. Both are commonly used for route optimization and can be adapted to solve the package delivery problem while adhering to the 140-mile total travel limit and time-sensitive constraints.

### **F3a. Differences from the Implemented Algorithm**

Dijkstra’s Algorithm differs from the Greedy Nearest Neighbor in that it calculates the shortest paths from a single source to all other nodes by considering the cumulative distance, not just the nearest next node. This makes it more suitable for globally optimal routing where returning to the hub or managing multiple delivery constraints is required. It’s more precise but has higher computational overhead. This approach is especially valuable in cases with a larger number of nodes or complex routes (GeeksforGeeks).

*A Search Algorithm*\* builds on Dijkstra by incorporating heuristics, such as the straight-line distance to the goal, making it more efficient in large search spaces. A\* can prioritize certain routes intelligently based on predicted future costs, offering better performance on complex maps. While A\* offers more optimal pathfinding, its complexity and need for accurate heuristics can make it overkill for simpler, local delivery scenarios like WGUPS.

### **G. What Would Be Done Differently in a Future Version**

In a future version of this project, one significant enhancement would be the integration of real-time traffic data to support dynamic routing. This would allow the algorithm to adjust delivery sequences in response to real-world events like congestion or road closures. Such a feature would require integration with a traffic API and the use of more adaptable algorithms like A\* or Dynamic Programming. Another improvement would be implementing package prioritization using a scoring system that emphasizes earlier deadlines, even if it slightly increases travel distance. Additionally, introducing automated unit testing and a graphical user interface (GUI) would enhance the program’s maintainability and user experience.

### **H. Data Structure Verification**

The core data structure used in the solution is a custom-built hash table. This structure supports constant-time (O(1)) insertion and retrieval of package data using package ID, which is ideal for quickly updating and accessing delivery information during route execution. It is specifically tailored for the WGUPS system, storing detailed package attributes such as address, delivery status, deadline, and delivery time which are all crucial for monitoring and verifying delivery progress throughout the day.

### **H1. Two Other Data Structures That Could Meet the Same Requirements**

Two other data structures that could potentially serve the same purpose are a Binary Search Tree (BST) and an adjacency list (commonly used in graph representations). A BST would allow ordered traversal of package data based on attributes like delivery deadline or zip code, offering efficient ways to sort and prioritize deliveries. An adjacency list, on the other hand, is ideal for representing the distance matrix between delivery points, which is critical for routing logic.

### **H1a. Differences Compared to Hash Table**

A BST differs from a hash table in that it provides sorted access, typically in O(log n) time for lookups, compared to the hash table’s constant-time access. This makes BSTs useful when sorting or range queries are needed, though they are slower for direct lookups by package ID. The adjacency list differs more significantly; while it excels at representing graphs for routing calculations (e.g., for use with Dijkstra or A\*), it is not optimized for storing individual package data. Unlike the hash table, it does not offer fast direct access to individual packages, making it more suitable for route mapping than for package management.

### **I. Sources**

**1.** GeeksforGeeks. *Dijkstra’s Shortest Path Algorithm | Greedy Algo-7*. 28 Apr. 2024, <https://www.geeksforgeeks.org/dsa/dijkstras-shortest-path-algorithm-greedy-algo-7/>