**Data Structures Used**

1. **Graph Representation**:
   * **Adjacency List**: The graph is represented using a HashMap<Node, List<Edge>>, where each key is a node, and the value is a list of edges (representing neighbors and weights). This is space-efficient for sparse graphs.
2. **Node and Edge**:
   * **Node Class**: Represents vertices with a name attribute.
   * **Edge Class**: Represents an edge with a destination node and weight.
3. **Priority Queue**:
   * Used in Dijkstra's algorithm to retrieve the next node with the smallest distance efficiently.
4. **Path Storage**:
   * **Lists**: Paths and their associated costs are stored in List<List<String>>.
5. **HashMaps**:
   * Used for storing distances and previous nodes during Dijkstra's algorithm.

**Algorithms Used**

1. **Depth-First Search (DFS)**:
   * Used to explore all possible paths from a start node to an end node recursively (Task 4).
   * Backtracking is applied to explore multiple paths.
2. **Dijkstra's Algorithm**:
   * Used to find the shortest path from a source to a destination (Task 5).
   * Implements a greedy strategy using a priority queue to select the smallest distance node.
3. **Path Cost Calculation**:
   * Iterates over the edges of a path to sum up weights.

**Time Complexity Analysis**

**1. Graph Construction**

* **Nodes**: Adding n nodes has a time complexity of O(n)O(n).
* **Edges**: Adding m edges has a time complexity of O(m)O(m).

**2. DFS for Pathfinding (Task 4)**

* **Time Complexity**:
  + DFS visits every node and edge in the graph.
  + In the worst case (exploring all paths), it generates all possible paths between start and end.
  + **Worst-case time complexity**: O(V+E+P⋅L)O(V + E + P \cdot L):
    - VV: Number of vertices.
    - EE: Number of edges.
    - PP: Total number of paths found.
    - LL: Average length of paths.

**3. Dijkstra's Algorithm (Task 5)**

* **Time Complexity**:
  + Initializing distances and priority queue: O(Vlog⁡V)O(V \log V).
  + Processing edges using the priority queue: O((V+E)log⁡V)O((V + E) \log V), where each edge relaxation is O(log⁡V)O(\log V).
  + **Total time complexity**: O((V+E)log⁡V)O((V + E) \log V).

**4. Path Cost Calculation**

* Iterates over each edge in the path:
  + For PP paths with an average length of LL, the complexity is O(P⋅L)O(P \cdot L).

**Overall Complexity**

* **Task 4 (DFS)**: O(V+E+P⋅L)O(V + E + P \cdot L).
* **Task 5 (Dijkstra)**: O((V+E)log⁡V)O((V + E) \log V).

In most real-world scenarios, Dijkstra's algorithm will dominate in runtime because of its reliance on the priority queue. DFS can be costly if the graph has many long or overlapping paths.