**Name : Ali Azhar**

**Roll No : SU92-BSAIM-S24-026**

**Section : 3A**

**Subject : Artificial Intelligence**

**Documentation of Task 1**

**“A\* Algorithm”**

Here's the documentation for the code with each heading explained in 2 to 3 lines:

### 1. ****Graph Class Definition****

The Graph class represents a graph structure where nodes (vertices) are connected by edges. It is initialized with an adjacency list, which is a dictionary mapping nodes to their neighbors and edge weights.

### 2. \_\_init\_\_ ****Method :****

The \_\_init\_\_ method initializes the graph object with an adjacency list. This list holds the connections (neighbors) for each node and the corresponding edge weights.

### 3. Get\_neighbors ****Method :****

This method takes a node as input and returns its list of neighbors from the adjacency list. Each neighbor is represented by a tuple consisting of a neighboring node and the edge weight.

### 4. H ****Method :****

The h method is a heuristic function used in the A\* algorithm. It returns a constant heuristic value for each node, indicating how close a node is to the goal node. In this case, all nodes have the same heuristic value.

### 5. A\_star\_algorithm ****Method :****

The a\_star\_algorithm method implements the A\* search algorithm. It finds the shortest path between the start node and the destination node by exploring the graph using both cost and heuristic values to prioritize nodes.

### 6. ****Open and Closed Lists****

The open list contains nodes that need to be explored, and the closed list contains nodes that have already been fully explored. The algorithm picks the node with the lowest evaluation function (f = g + h) from the open list and proceeds to explore its neighbors.

### 7. ****Evaluation Function (f)****

The evaluation function combines the current path cost (g(n)) and the heuristic estimate (h(n)) to choose the most promising node to explore. It helps prioritize nodes with lower combined values of cost and heuristic.

### 8. ****Path Reconstruction****

Once the algorithm finds the destination node, the a\_star\_algorithm method reconstructs the path from the start node to the destination by tracing back through the parents dictionary, which maps each node to its predecessor.

### 9. ****Adjacency List Example****

The adjacency list defines the structure of the graph. For example, node 'A' is connected to 'B' (cost 1), 'C' (cost 3), and 'D' (cost 7). Other nodes like 'B', 'C', and 'D' have their respective connections and weights.

### 10. ****Creating and Using the Graph Object****

To use the graph, an instance is created by passing an adjacency list. After that, the A\_star\_algorithm method is called with the start and stop nodes (e.g., from 'A' to 'D'), and the shortest path is calculated.

### 11. ****Output the Result****

Once the algorithm computes the shortest path, the result is returned and printed. If no path is found, a message indicating that no path exists is displayed.

### Example Execution Output:

Given the adjacency list, the algorithm finds the shortest path from node 'A' to node 'D'. It might output something like:

Shortest path: ['A', 'B', 'D']

This output shows the sequence of nodes that lead from the start node to the destination.

This documentation provides a high-level overview of the main components of the Graph class and the A\* search algorithm implementation.

