**Name: Muhammad Huzaifa**

**Roll No: SU92-BSAIM-S24-010**

**Section: 3A**

**Subject: AI LAB**

**DOCUMENTATIONS:**

**Code Explanation:**

**1. Graph Class:**

The Graph class is used to represent a **graph with weighted edges**. It provides methods for defining **adjacency lists**, getting neighbors, and implementing the A\* search algorithm.

**Attributes:**

* **adjacency\_list**: A dictionary representing the graph where **keys are nodes** and **values are lists of tuples** (neighbor, weight).

**Methods:**

**2. get\_neighbors(self, v):**

* **Purpose:** Returns a list of neighboring nodes and their corresponding edge weights.
* **Input:** v → A node in the graph.
* **Output:** A list of (neighbor, weight) tuples.

**3. h(self, n):**

* **Purpose:** Heuristic function that provides an estimated cost from node n to the goal.
* **Input:** n → A node in the graph.
* **Output:** The heuristic value (integer).

**4. a\_star\_algorithm(self, start\_node, stop\_node):**

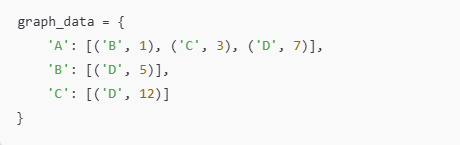
* **Purpose:** Implements the *A search algorithm*\* to find the shortest path from start\_node to stop\_node.
* **Inputs:**
  + start\_node: The starting node.
  + stop\_node: The destination node.
* **Outputs:**
  + If a path exists, prints and returns the shortest path as a **list of nodes**.
  + If no path exists, prints "Path does not exist!" and returns None.

**Algorithm Steps:**

1. **Initialize Data Structures:**
   * open\_list: Stores nodes that need to be evaluated.
   * closed\_list: Stores nodes that have already been evaluated.
   * g: Dictionary storing the cost from start\_node to each node.
   * parents: Dictionary storing the parent of each node for path reconstruction.
2. **Find the Best Node:**
   * The node with the **lowest f(n) = g(n) + h(n)** is selected, where:
     + g(n): Cost from start\_node to n.
     + h(n): Estimated heuristic cost from n to stop\_node.
3. **Path Reconstruction:**
   * If n == stop\_node, reconstruct the path by tracing back from the goal to the start node.
4. **Expand Neighbors:**
   * Check neighboring nodes:
     + If the new path to a neighbor is shorter, update g and parents.
     + Move the processed node from open\_list to closed\_list.
5. **Repeat Until Path is Found or No More Nodes to Explore.**

**5. Graph Data Representation:**

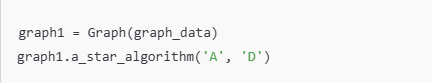
A dictionary named graph\_data defines the graph as:



This represents:

* **A** connects to **B** (weight = 1), **C** (weight = 3), and **D** (weight = 7).
* **B** connects to **D** (weight = 5).
* **C** connects to **D** (weight = 12).

1. **Running the Algorithm:**

****

 **Starts from** node '**A**'

 **Finds the shortest path** to node **'D'.**

**OUTPUT:**

****

**------------------------------------------------------------------------------------**