DATE:06.08.25

TASK 3
A * Algorithm

Implementation of A * Algorithm to find the optimal path using Python by following constraints.
•The goal of the A* algorithm is to find the shortest path from the starting point to the goal point as fast as possible.
•The full path cost (f) for each node is calculated as the distance to the starting node (g) plus the distance to the goal node (h).
•Distances is calculated as the manhattan distance (taxicab geometry) between nodes.
Tools- Python, Online Simulator - https://graphonline.ru/en/

PROBLEM STATEMENT: **CO2** **S3**

An AI navigation system for an autonomous drone flying over a network of waypoints (nodes). Each node represents a location, and edges represent possible paths between locations with different travel costs. The drone must find the shortest and most efficient route from a starting waypoint to a target waypoint, avoiding longer or costly paths. Each node has a heuristic value representing the estimated cost to reach the goal (e.g., based on straight-line distance).

DATE:06.08.25

TASK:3

Implementation of A * Algorithm to find the optimal path

AIM

To implement the A* (A-Star) Search Algorithm using Python to find the optimal shortest path between a start node and a goal node in a weighted graph, using both the actual path cost and a heuristic estimate (Manhattan or straight-line distance).

ALGORITHM

1. Start at the starting point on the map.
2. Write down the cost to reach this point from the start (g = 0), and estimate how far it is from the goal using straight lines or grid steps (h = heuristic).

Then calculate the total cost:

f = g + h.

3. Look at all the neighbouring points you can go to from your current position.
4. For each neighbour:
    - Add the cost to get there from where you are now (g).
    - Estimate how far it is from the goal (h).
    - Add them to get the total cost: f = g + h.
    - Write down this total cost for each possible path.
5. Pick the point with the lowest total cost (f) and go there next.
6. Repeat the process:
    - Check all neighbouring points from your current position.
    - Update their g, h, and f values.
    - Always move to the next point with the lowest f value.
7. Stop when you reach the goal.
8. Trace back the path you took to get the full route from start to goal.

PROGRAM
A* Algorithm on a Graph

```
import heapq

# Graph: adjacency list with edge costs
graph = {
    'A': [('B', 1), ('C', 4)],
    'B': [('D', 5), ('E', 12)],
    'C': [('F', 3)],
    'D': [('G', 2)],
    'E': [('G', 3)],
    'F': [('G', 5)],
    'G': []
}

# Heuristic values (estimated cost from node to goal)
heuristic = {
    'A': 7,
    'B': 6,
    'C': 5,
    'D': 3,
    'E': 2,
    'F': 4,
    'G': 0
}

# A* Algorithm
def a_star(graph, start, goal):
    open_list = []
    heapq.heappush(open_list, (0 + heuristic[start], 0, start, [start]))  # (f, g, current_node, path)
    visited = set()

    while open_list:
        f, g, current, path = heapq.heappop(open_list)

        if current in visited:
            continue
        visited.add(current)

        # Goal reached
        if current == goal:
            print("Optimal Path Found:", " → ".join(path))
            print("Total Cost:", g)
            return

        # Explore neighbors
        for neighbor, cost in graph[current]:
            if neighbor not in visited:
                new_g = g + cost
            new_f = new_g + heuristic[neighbor]
            heapq.heappush(open_list, (new_f, new_g, neighbor, path + [neighbor]))

    print("No path found.")

# Run the algorithm
start_node = 'A'
goal_node = 'G'
a_star(graph, start_node, goal_node)
```

OUTPUT

```
Optimal Path Found: A → B → D → G
Total Cost: 8
```

RESULT

Thus, the Implementation of A * Algorithm to find the optimal path using Python Was successfully executed and output was verified.