Write the python program to implement A* algorithm

AIM

To implement the **A* (A-Star) Search Algorithm** in Python to find the shortest path from a start node to a goal node using a heuristic function.

ALGORITHM

1. Represent the graph as an adjacency list with edge costs.
2. Define a **heuristic function h(n)** estimating cost from each node to the goal.
3. Initialize an **open set (priority queue)** with the start node, storing (f, g, node, path) where:
    a. g = cost from start to current node
    b. f = g + h[node] = estimated total cost
4. Initialize a **visited set** to track explored nodes.
5. While the open set is not empty:
    a. Pop the node with the smallest f value.
    b. If it is the goal, return the path and cost.
    c. Otherwise, for each neighbor not visited, calculate new g and f and push to the open set.
6. Repeat until the goal node is reached.

8 PUZZLE AI.py - C:/Users/gayathri/Downloads/8 PUZZLE AI.py (3.8.2)

File Edit Format Run Options Window Help

```
import heapq
def a_star(graph, start, goal, h):
    open_set = []
    heapq.heappush(open_set, (h[start], 0, start, [start]))
    visited = set()
    while open_set:
        f, g, node, path = heapq.heappop(open_set)
        if node == goal:
            return path, g
        if node in visited:
            continue
        visited.add(node)
        for neighbor, cost in graph[node].items():
            if neighbor not in visited:
                heapq.heappush(open_set, (g+cost+h[neighbor], g+cost, neighbor, path+[neighbor]))
graph = {
    'A': {'B': 1, 'C': 4},
    'B': {'C': 2, 'D': 5},
    'C': {'D': 1},
    'D': {}
}
h = {'A': 7, 'B': 6, 'C': 2, 'D': 0}
path, cost = a_star(graph, 'A', 'D', h)
print("Path:", path)
print("Cost:", cost)
```

```
==========
Path: ['A', 'C', 'D']
Cost: 5
>>>
```

RESULT

The program successfully found the shortest path from node 'A' to 'D' using A* search.