EX NO: 4 A* informed search

Date:

Aim:

To implement the A* informed search algorithm in Python. the A* algorithm is used to find the shortest Path from a start node to goal node in a graph, using both the actual cost from the start node and a heuristic estimate to the goal.

Algorithm:

1. Initialize the Graph:
   - Represent the graph as a dictionary where each node has a list of tuples representing its neighboring nodes and the cost to reach them
2. Heuristic function:
   - Define a heuristic function h(n) that estimates the cost from node n to the goal node. the choice of heuristic depends on the problem domain. for simplicity assume we are given this heuristic function.
3. Define the A search function*:

Initialize two sets: open_set (nodes to evaluate) and closed_set (nodes already evaluated).

Use a Priority queue (min-heap) to pick the node with the lowest f(n) = g(n) + h(n) where g(n) is the cost from the start node to n and h(n) is the heuristic from n to the goal

- Initialize the g values (costs from the start) to infinity for all nodes except the start node.
- Track the parent of each node to reconstruct the path after reaching the goal.

4. A search process*:

- while open_set is not empty:

1. Select the nodes n in open_set with the lowest f(n)
2. If n is the goal node, reconstruct the path and return it.
3. Move n from open_set to closed_set

- If the goal node is not reached and open_set is empty, return that there is no solution.

Program:

```
import heapq

def a_star(graph, start, goal, h):
    open_set = []
    heapq.heappush(open_set, (h[start], start))
    g = {node: float('inf') for node in graph}
    g[start] = 0
    f = {node: float('inf') for node in graph}
    f[start] = h[start]
    came_from = {}
    closed_set = set()
```

```
while open_set:
    _, current = heapq.heappop(open_set)
    if current == goal:
        path = []
        while current in came_from:
            path.append(current)
            current = came_from[current]
        path.append(start)
        return path[::-1]
    closed_set.add(current)
    for neighbor, cost in graph[current]:
        if neighbor in closed_set:
            continue
        tentative_g = g[current] + cost
        if tentative_g < g[neighbor]:
            came_from[neighbor] = current
            g[neighbor] = tentative_g
            f[neighbor] = g[neighbor] + h(neighbor)
        if neighbor not in open_set:
            heapq.heappush(open_set, (f[neighbor], neighbor))
    return None
graph = {
    'A': [('B', 1), ('C', 3)],
    'B': [('A', 1), ('D', 2), ('E', 5)],
    'C': [('A', 3), ('F', 2)],
    'D': [('B', 2)],
```

```
'E': [('B', 5), ('F', 2)],
'F': [('C', 2), ('E', 2), ('G', 2)],
'G': [('F', 2)]
}
heuristic = {
    'A' = 7,
    'B' = 6,
    'C' = 2,
    'D' = 5,
    'E' : 3,
    'F' : 2,
    'G' : 0
}
start_node = 'A'
goal_node = 'G'
Path = a_star (graph, start_node, goal_node,
                heuristic)
if Path:
    Print ("Path Found:", Path)
else:
    Print ("No Path Found")
```

OutPut:

Path Found: ['A', 'C', 'F', 'G']