Experiment-4

A* Informed Search

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 a goal node in a graph, using both the actual cost from the start node and a heuristic estimate to the goal.

ALGORITHM

1. 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. 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.
3. Initialize two sets: open_set and closed_set
4. 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 estimate from n to the goal.
5. Initialize the g values to infinity for all nodes except the start node, which is initialized to 0.
6. Initialize the f values using the start node.
7. Track the parent of each node to reconstruct the path after reaching the goal.

8. while open_set is not empty:
   (i) Select the node n in open_set with the lowest f(n).
   (ii) If n is the goal node, reconstruct the path and return it.
   (iii) Move n from open_set to closed_set.
   (iv) For each neighbor of n, calculate the tentative g value. If this is lower than the current g value for the neighbor, update the g value and calculate the new f value. If the neighbor is not in open_set, add it.

9. 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~~
```

```
        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', 1), ('E', 5)],
    'C': [('A', 3), ('F', 2)],
    'D': [('B', 1)],
    'E': [('B', 5), ('F', 2)],
    'F': [('C', 2), ('E', 2), ('G', 2)],
    'G': [('F', 1)]
}

heuristic = {
    'A': 7,
    'B': 6,
    'C': 2,
    'D': 5,
    'E': 3,
    'F': 1,
    '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']

Result

Thus, the A* Informed search algorithm was successfully executed and the output is verified.