Experiment-1

Student Name: Alasso **UID:**
Branch: CSE **Section/Group:**
Semester: 5th **Date of Performance: 10/08/23**
Subject Name: AIML **Subject Code: 21CSH-316**

1. **Aim:** Implement A* algorithm in python.

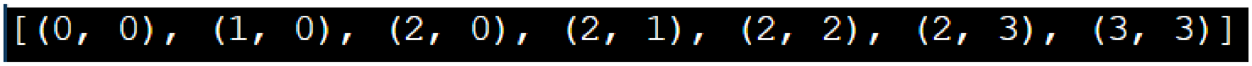
2. **Objective:** Implement the A* algorithm to efficiently find the shortest path in a graph while considering both the cost to reach a node and a heuristic estimate of its potential to reach the goal.

3. **Source Code**:

```
import heapq

class Node:
    def___init_(self, position, parent=None):
        self.position = position
        self.parent = parent
        self.g = 0  # Cost from start node to current node
        self.h = 0  # Heuristic (estimated cost) from current node to goal node
        self.f = 0  # Total cost (g + h)

    def___lt_(self, other):
        return self.f < other.f

def heuristic(node, goal):
    # Manhattan distance heuristic (can be changed to Euclidean distance or others)
    return abs(node.position[0] - goal[0]) + abs(node.position[1] - goal[1])
```

```
def astar(grid, start, goal):
    open_list = []
    closed_set = set()

    start_node = Node(start)
    goal_node = Node(goal)

    heapq.heappush(open_list, start_node)

    while open_list:
        current_node = heapq.heappop(open_list)

        if current_node.position == goal_node.position:
            path = []
            while current_node is not None:
                path.append(current_node.position)
                current_node = current_node.parent
            return path[::-1]

        closed_set.add(current_node.position)

        for next_position in [(0, -1), (0, 1), (-1, 0), (1, 0)]:  # Possible adjacent
positions
            node_position = (current_node.position[0] + next_position[0],
current_node.position[1] + next_position[1])

            if node_position[0] < 0 or node_position[0] >= len(grid) or
node_position[1] < 0 or node_position[1] >= len(grid[0]):
                continue

            if grid[node_position[0]][node_position[1]] == 1:
                continue

            if node_position in closed_set:
                continue

            new_node = Node(node_position, current_node)
            new_node.g = current_node.g + 1
            new_node.h = heuristic(new_node, goal_node.position)
            new_node.f = new_node.g + new_node.h

            for node in open_list:
                if new_node.position == node.position and new_node.f >= node.f:
                    break
            else:
                heapq.heappush(open_list, new_node)
    return None  # No path found

# Example usage:
grid = [
    [0, 0, 0, 0],
    [0, 1, 1, 0],
    [0, 0, 0, 0],
    [0, 0, 1, 0]
]

start_point = (0, 0)
goal_point = (3, 3)

path = astar(grid, start_point, goal_point)
print(path)
```

4. Output:



```
[(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (2, 3), (3, 3)]
```