Experiment: 1.1

Student Name: SANJIV GUPTA
UID: 21BCS-*3478*
Branch: CSE
Section/Group: 21BCS-IOT-602B
Semester: 5th
Date: 17/08/23
Subject Name: AIML Lab
Subject Code: 21CSH-316

1. **AIM:** *Evaluate the performance and effectiveness of the A* algorithm implementation in Python*

2. **Objective:** *The objective is to assess how well the A* algorithm performs in solving a specific problem or scenario, and to analyze its effectiveness in comparison to other algorithms or approaches.*

3. **Tools/Resource Used:**

1. *Python programming language.*
2. *A* algorithm implementation in Python.*
3. *Relevant data or problem scenario for testing the algorithm.*

4. **Algorithm:**

1. *Define the problem scenario or task for which the A* algorithm will be used.*
2. *Implement the A* algorithm in Python, taking into accounts the specific problem requirements and constraints.*
3. *Provide necessary data structures, such as graphs or grids, to represent the problem space.*
4. *Write code to initialize the start and goal states or nodes.*
5. *Implement the A* algorithm, including the heuristic function and the necessary data structures, such as priority queues or heaps.*
6. *Run the algorithm on the given problem scenario and record the execution time.*
7. *Monitor and log the nodes expanded, the path generated, and any other relevant information during the algorithm's execution.*
8. *Repeat steps 4-7 for multiple problem scenarios or test cases, if applicable.dq*

5. Program 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):
    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)]:
```

```
            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)
```

6. Output/Result:

```
[(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (2, 3), (3, 3)
]
PS C:\Users\SANJIV\Downloads\CSE-5TH-SEM-WORKSHEETS-DAA
-AIML-IOT-AP>
```

7. Learning Outcomes:

1. *Record the execution time of the A* algorithm for each problem scenario.*
2. *Note the number of nodes expanded during the algorithm's execution.*
3. *Record the optimal path generated by the A* algorithm.*
4. *Evaluate the correctness of the generated path by comparing it with known optimal solutions, if available.*
5. *Analyze the efficiency and effectiveness of the A* algorithm based on the execution time, number of nodes expanded, and the quality of the generated paths.*
6. *Compare the performance of the A* algorithm with other algorithms or approaches, if applicable.*