EXPERIMENT 4

PROBLEM TITLE

Write a Program to Implement A* Search.

CODE

```
import heapq

def a_star_search(start, goal, grid):
    def heuristic(a, b):
        return abs(a[0] - b[0]) + abs(a[1] - b[1])  # Manhattan distance

    open_list = []
    heapq.heappush(open_list, (0 + heuristic(start, goal), 0, start))
    came_from = {}
    cost_so_far = {start: 0}

    while open_list:
        _, current_cost, current = heapq.heappop(open_list)
        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            return path[::-1]  # Return reversed path

        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            neighbor = (current[0] + dx, current[1] + dy)
            if 0 <= neighbor[0] < len(grid) and 0 <= neighbor[1] < len(grid[0])
and grid[neighbor[0]][neighbor[1]] == 0:
                new_cost = current_cost + 1
                if neighbor not in cost_so_far or new_cost < cost_so_far[neighbor]:
                    cost_so_far[neighbor] = new_cost
                    priority = new_cost + heuristic(neighbor, goal)
                    heapq.heappush(open_list, (priority, new_cost, neighbor))
                    came_from[neighbor] = current

    return None  # No path found
```

```
# Example grid: 0 = empty, 1 = obstacle
grid = [
    [0, 1, 0, 0, 0],
    [0, 1, 0, 1, 0],
    [0, 0, 0, 1, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 0, 0]
]

start = (0, 0)
goal = (4, 4)
path = a_star_search(start, goal, grid)
print("A* Path:", path)
```

EXPECTED OUTPUT

```
A* Path: [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (3, 2), (4, 2), (4, 3), (4, 4)]
```