VISHALINI.R
231701062
CSD

A* SEARCH ALGORITHM

AIM

To implement the **A* (A-star) search algorithm** to find the shortest path between a **start node** and a **goal node** in a given grid. The A* algorithm uses a combination of **cost to reach a node (g-cost)** and an **estimated cost to the goal (h-cost, heuristic)** to efficiently find the shortest path.

PROGRAM

```
from queue import PriorityQueue



class Node:
    def __init__(self, position, parent=None):
        self.position = position  # (x, y) coordinates
        self.parent = parent
        self.g = 0  # Cost from start node
        self.h = 0  # Heuristic cost to goal
        self.f = 0  # Total cost
    def __lt__(self, other):
        return self.f < other.fdef heuristic(a, b):
    """Calculate Manhattan distance heuristic."""
    return abs(a[0] - b[0]) + abs(a[1] - b[1])
 def a_star_search(grid, start, goal):
    """A* search algorithm implementation."""
    open_list = PriorityQueue()
    start_node = Node(start)

    goal_node = Node(goal)

    open_list.put((0, start_node))

    closed_set = set()

    while not open_list.empty():

        _, current_node = open_list.get()

        if current_node.position in closed_set:

            continue

        closed_set.add(current_node.position)

        if current_node.position == goal:

            path = []

            while current_node:

                path.append(current_node.position)

                current_node = current_node.parent

            return path[::-1]  # Return reversed path

        neighbors = [(0, -1), (0, 1), (-1, 0), (1, 0)]  # Up, Down, Left, Right

        for dx, dy in neighbors:

            neighbor_pos = (current_node.position[0] + dx, current_node.position[1] + dy)

           if (0 <= neighbor_pos[0] < len(grid) and 0 <= neighbor_pos[1] < len(grid[0]) and
grid[neighbor_pos[0]][neighbor_pos[1]] == 0):

                neighbor_node = Node(neighbor_pos, current_node)

                neighbor_node.g = current_node.g + 1

                neighbor_node.h = heuristic(neighbor_pos, goal)

                neighbor_node.f = neighbor_node.g + neighbor_node.h
                open_list.put((neighbor_node.f, neighbor_node)

    return None  # No path found

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(grid, start, goal)

print("Shortest Path:", path)
```

OUTPUT

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

RESULT

The program successfully finds the shortest path in a grid while avoiding obstacles.The output includes the **optimal path from the start node to the goal node** if a path exists.