A* SEARCH

NAME:Sai rajaram.J
ROLL NO: 241801238
EX.NO:4

EX Name: A* SEARCH
DATE: 03/05/2025

```
import heapq

# Define the grid and movements

class Node:

    def __init__(self, position, parent=None, g=0, h=0):

        self.position = position  # (row, col)

        self.parent = parent  # Parent node

        self.g = g  # Cost from start node

        self.h = h  # Heuristic cost to goal

        self.f = g + h  # Total cost (f = g + h)

    def __lt__(self, other):

        return self.f < other.f  # Priority queue comparison (min-heap)


# Heuristic function (Manhattan Distance)

def heuristic(a, b):

    return abs(a[0] - b[0]) + abs(a[1] - b[1])

# A* algorithm

def a_star(grid, start, goal):

    rows, cols = len(grid), len(grid[0])

    open_list = []

    heapq.heappush(open_list, Node(start, None, 0, heuristic(start, goal)))  # Push start node

    closed_set = set()  # To keep track of visited nodes

    while open_list:

        current_node = heapq.heappop(open_list)  # Get node with lowest f-value

        # If goal is reached, reconstruct the path

        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 (start to goal)
```

A* SEARCH

```
        closed_set.add(current_node.position)  # Mark the current node as visited
        # Explore neighbors (up, down, left, right)
        for dr, dc in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            new_pos = (current_node.position[0] + dr, current_node.position[1] + dc)
            # Check if the new position is valid
            if (0 <= new_pos[0] < rows and 0 <= new_pos[1] < cols and
                grid[new_pos[0]][new_pos[1]] == 0 and new_pos not in closed_set):
                # Calculate g and h for the new node
                new_node = Node(new_pos, current_node, current_node.g + 1, heuristic(new_pos, goal))
                heapq.heappush(open_list, new_node)  # Add to open list

    return None  # Return None if no path is found
# Example grid: 0 = free space, 1 = obstacle
warehouse_grid = [
    [0, 0, 0, 0, 1],
    [1, 1, 0, 1, 0],
    [0, 0, 0, 0, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 0, 0]
]
# Start and goal positions
start_position = (0, 0)
goal_position = (4, 4)]
# Find the optimal path using A*
path = a_star(warehouse_grid, start_position, goal_position)
print("Optimal Path:", path)
```

```
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/HDC0719089/AppData/Local/Programs/Python/Python311/jytdfuj.py
Optimal Path: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (2, 3), (2, 4), (3, 4), (4, 4)]
Sai rajaram.J 241801238
>>>
```