Ex No: 1d

IMPLEMENTATION OF A* SEARCH ALGORITHM

Aim:

To implement A* Search Algorithm.

Case Scenario:

A delivery robot in a warehouse needs to find the shortest path from the entrance to a specific package location. The warehouse is represented as a grid with some obstacles. Implement the A* search algorithm to help the robot navigate efficiently.

Procedure:

1. **Define the Node class** with attributes: position, parent, cost (g), heuristic (h), and total cost (f = g + h).
2. **Implement the heuristic function** using the Manhattan distance formula.
3. **Initialize A Search* with:**

- `open_list` (priority queue) containing the start node.
- `closed_set` to store visited nodes.

4. **While open_list is not empty:**

- Extract the node with the lowest `f-value`.
- If the goal is reached, trace back the path and return it.
- Add the current node to `closed_set`.
- Generate new valid moves (up, down, left, right) ensuring they are within bounds and not obstacles.
- Calculate new cost `g`, heuristic `h`, and total `f`.
- Add new nodes to `open_list` for further exploration.

5. **Return the optimal path** if found, else return `None` if no path exists.

Program:

```
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
    def __lt__(self, other):
        return self.f < other.f  # Priority queue comparison
def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])  # Manhattan Distance
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)))
    closed_set = set()
    while open_list:
        current_node = heapq.heappop(open_list)  # Get node with lowest f-value
        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

        closed_set.add(current_node.position)
        for dr, dc in [(-1, 0), (1, 0), (0, -1), (0, 1)]:  # Possible moves
            new_pos = (current_node.position[0] + dr, current_node.position[1] + dc)
            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):
                new_node = Node(new_pos, current_node, current_node.g + 1, heuristic(new_pos, 
goal))
                heapq.heappush(open_list, new_node)
    return None  # No path 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_position = (0, 0)
goal_position = (4, 4)
path = a_star(warehouse_grid, start_position, goal_position)
print("Optimal Path:", path)
```

Output:

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