

Ex. No : 5	Implementation of A*Algorithm
Date :	

Aim :-

To implement A*Algorithm using python.

Algorithm :-

Step 1: Start.

Step 2: Initialize the open list.

Step 3: Follow the steps until the open list is non-empty.

Step 4: Find the node with the least f in the open list and name it as q.

Step 5: Remove q from the open list.

Step 6: Produce q's successors (descendants) and set q as their parent.

Step 7: Calculate g and h for the successors.

Step 8: Skip a successor if a node in the open list with the same position has a better f value.

Step 9: Stop.

Program :-

```
import heapq

class Node:

    def __init__(self, position, g_cost=0, h_cost=0):

        self.position = position

        self.g_cost = g_cost

        self.h_cost = h_cost

        self.f_cost = g_cost + h_cost

        self.parent = None

    def __lt__(self, other):

        return self.f_cost < other.f_cost

class AStar:

    def __init__(self, start, goal, grid):

        self.start = start

        self.goal = goal

        self.grid = grid

        self.open_list = []

        self.closed_list = set()

    def get_neighbors(self, current):
```

```
neighbors = []
directions = [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1), (1, -1), (1, 1)]
for direction in directions:
    neighbor_position = (current[0] + direction[0], current[1] + direction[1])
    if self.is_valid(neighbor_position):
        neighbors.append(neighbor_position)
return neighbors

def is_valid(self, position):
    if 0 <= position[0] < len(self.grid) and 0 <= position[1] < len(self.grid[0]):
        return self.grid[position[0]][position[1]] == 0
    return False

def reconstruct_path(self, node):
    path = []
    while node:
        path.append(node.position)
        node = node.parent
    return path[::-1]

def heuristic_cost(self, current, goal):
    return abs(current[0] - goal[0]) + abs(current[1] - goal[1])

def search(self):
    start_node = Node(self.start, g_cost=0, h_cost=self.heuristic_cost(self.start, self.goal))
    heapq.heappush(self.open_list, start_node)
    while self.open_list:
        current_node = heapq.heappop(self.open_list)
        if current_node.position == self.goal:
            return self.reconstruct_path(current_node)
        self.closed_list.add(current_node.position)
        for neighbor_position in self.get_neighbors(current_node.position):
            if neighbor_position in self.closed_list:
                continue
```

```
tentative_g_cost = current_node.g_cost + 1

neighbor_node = Node(neighbor_position, g_cost=tentative_g_cost,
h_cost=self.heuristic_cost(neighbor_position, self.goal))

neighbor_node.parent = current_node

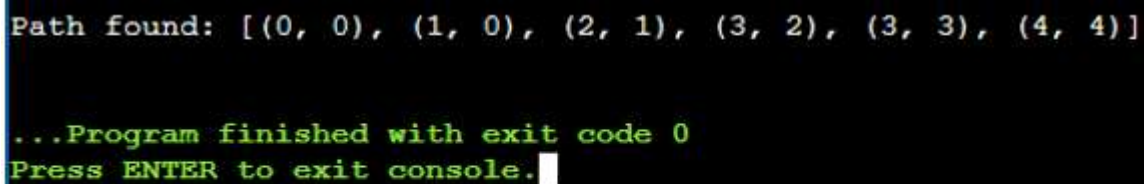
if not any(neighbor.position == neighbor_node.position and neighbor.f_cost <=
neighbor_node.f_cost for neighbor in self.open_list):

    heapq.heappush(self.open_list, neighbor_node)

return None

# Example Grid (0 = walkable, 1 = obstacle)
grid = [
    [0, 0, 0, 0, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 1, 0],
    [0, 1, 0, 0, 0],
    [0, 0, 0, 1, 0]
]

start = (0, 0)
goal = (4, 4)
print("717824P502")
a_star = AStar(start, goal, grid)
path = a_star.search()
if path:
    print("Path found:", path)
else:
    print("No path found.")
```

Output :-

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

...Program finished with exit code 0
Press ENTER to exit console.
```

Result :-

Thus, the program for A*Algorithm was implemented successfully.