Experiment:- 4

Objective:- IMPLEMENT A* ALGORITHM

Theory:- The A* algorithm is a highly effective and well-known search technique utilized for finding the most efficient path between two points in a graph. It is applied in scenarios such as pathfinding in video games, network routing and various artificial intelligence (AI) applications. It was developed in 1968 by Peter Hart, Nils Nilsson and Bertram Raphael as an improvement on Dijkstra's algorithm.

Key Components of A* Algorithm

A* uses two important parameters to find the cost of a path:

1.g(n): Actual cost of reaching node n from the start node. This is the accumulated cost of the path from the start node to node n.

2.h(n): The heuristic finds the cost to reach the goal from node n . This is a weighted guess about how much further it will take to reach the goal.

The function, f(n)=g(n)+h(n) is the total estimated cost of the cheapest solution through node n. This function combines the path cost so far and the heuristic cost to estimate the total cost guiding the search more efficiently.

To understand A* algorithm, you need to be familiar with these fundamental concepts:

- **Nodes**: Points in your graph (like intersections on a map)
- **Edges**: Connections between nodes (like roads connecting intersections)
- **Path Cost**: The actual cost of moving from one node to another
- **Heuristic**: An estimated cost from any node to the goal
- **Search Space**: The collection of all possible paths to explore

Program:-

```
import heapq
def heuristic(a, b):

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

def a_star_search(grid, start, goal):

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

```
open_set = []
heapq.heappush(open_set, (0, start))
came_from = {}

g_score = {node: float('inf') for row in range(rows) for node in [(row, c) for c in range(cols)]}
g_score[start] = 0

f_score = {node: float('inf') for row in range(rows) for node in [(row, c) for c in range(cols)]}
f_score[start] = heuristic(start, goal)

while open_set:
current_f_cost, current_node = heapq.heappop(open_set)

if current_node == goal:

path = []
while current_node in came_from:
path.append(current_node)
current_node = came_from[current_node]
path.append(start)
return path[::-1]

for dr, dc in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
neighbor = (current_node[0] + dr, current_node[1] + dc)

if 0 <= neighbor[0] < rows and \
0 <= neighbor[1] < cols and \
grid[neighbor[0]][neighbor[1]] == 0:
tentative_g_score = g_score[current_node] + 1

if tentative_g_score < g_score[neighbor]:
came_from[neighbor] = current_node
g_score[neighbor] = tentative_g_score
f_score[neighbor] = g_score[neighbor] + heuristic(neighbor, goal)
heapq.heappush(open_set, (f_score[neighbor], neighbor))

return None
```

```
if __name__ == "__main__":
 grid = [

 [0, 0, 0, 0, 0],
 [0, 1, 0, 1, 0],
 [0, 1, 0, 0, 0],
 [0, 0, 0, 1, 0],
 [0, 0, 0, 0, 0]
 ]
 start_node = (0, 0)
 goal_node = (4, 4)
 path = a_star_search(grid, start_node, goal_node)
 if path:
 print("Path found:")
 for r, c in path:
 print(f"({r}, {c})")
 else:
 print("No path found.")
```

Output:-

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