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.