**Exp 10: Write the python program to implement A\* algorithm.**

**Input:**

import heapq

def heuristic(a, b):

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

def a\_star(grid, start, goal):

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

    open\_set = []

    heapq.heappush(open\_set, (0 + heuristic(start, goal), 0, start))  # (f, g, position)

    came\_from = {}

    g\_score = {start: 0}

    while open\_set:

        f, current\_g, current = heapq.heappop(open\_set)

        if current == goal:

            path = []

            while current in came\_from:

                path.append(current)

                current = came\_from[current]

            path.append(start)

            path.reverse()

            return path

        neighbors = [

            (current[0] + 1, current[1]),

            (current[0] - 1, current[1]),

            (current[0], current[1] + 1),

            (current[0], current[1] - 1),

        ]

        for neighbor in neighbors:

            r, c = neighbor

            if 0 <= r < rows and 0 <= c < cols and grid[r][c] == 0:

                tentative\_g = current\_g + 1

                if neighbor not in g\_score or tentative\_g < g\_score[neighbor]:

                    g\_score[neighbor] = tentative\_g

                    f\_score = tentative\_g + heuristic(neighbor, goal)

                    heapq.heappush(open\_set, (f\_score, tentative\_g, neighbor))

                    came\_from[neighbor] = current

    return None

# 0 = free cell, 1 = obstacle

grid = [

    [0, 1, 0, 0, 0],

    [0, 1, 0, 1, 0],

    [0, 0, 0, 1, 0],

    [1, 1, 0, 0, 0],

    [0, 0, 0, 1, 0]

]

start = (0, 0)

goal = (4, 4)

path = a\_star(grid, start, goal)

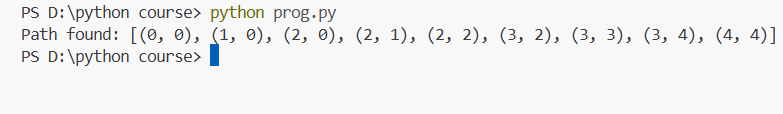
if path:

    print("Path found:", path)

else:

    print("No path found")

**output:**

****