Lab Assignment: Solving a Maze with the A* Algorithm (Python)

Objective

In this lab, students will:

- Represent a maze using a 2D grid
- Define walls, a start point, and a goal point
- Implement the A* (A-star) pathfinding algorithm
- Use an AI search strategy to find the shortest path from start to goal

By the end of this lab, the program should automatically find and display a valid path through the maze.

Background

Pathfinding is a classic problem in Artificial Intelligence and robotics.
The A* algorithm is an informed search algorithm that uses:

- The cost so far (g)
- A heuristic estimate to the goal (h)

The total cost function is:

```
f(n) = g(n) + h(n)
```

A* is widely used in navigation systems, robotics, and video games.

Maze Representation

The maze is represented as a 2D grid:

- `0` → free cell
- `1` → wall
- `S` → start position
- `G` → goal position

Example:

```
S 0 1 0 0
1 0 1 0 1
0 0 0 0 0
0 1 1 1 0
0 0 0 1 G
```

Allowed movements: up, down, left, right (no diagonal moves).

Heuristic Function

Use the Manhattan distance as the heuristic function:

```
h(n) = |x_goal - x_n| + |y_goal - y_n|
```

This heuristic is admissible and suitable for grid-based environments.

Tasks

1. Create a maze as a 2D list
2. Define the start and goal positions

3. Implement the A* algorithm
4. Compute the shortest path
5. Display the resulting path

A* Algorithm – Pseudocode

```
function A_STAR(start, goal, maze):

    open_set ← priority queue
    open_set.push(start, priority = 0)

    came_from ← empty map

    g_score ← map with default value infinity
    g_score[start] ← 0

    f_score ← map with default value infinity
    f_score[start] ← heuristic(start, goal)

    while open_set is not empty:

        current ← node in open_set with the lowest f_score

        if current == goal:
            return reconstruct_path(came_from, current)

        remove current from open_set

        for each neighbor of current:
            if neighbor is a wall:
                continue

            tentative_g ← g_score[current] + 1

            if tentative_g < g_score[neighbor]:
```

```
                came_from[neighbor] ← current
                g_score[neighbor] ← tentative_g
                f_score[neighbor] ← tentative_g + heuristic(neighbor,
goal)

                if neighbor not in open_set:
                    open_set.push(neighbor, priority =
f_score[neighbor])

    return "No path found"
```

Path Reconstruction – Pseudocode

```
function reconstruct_path(came_from, current):

    path ← [current]

    while current in came_from:
        current ← came_from[current]
        add current to path

    reverse path
    return path
```

Optional Extensions

- Display the maze and the path using text or simple graphics
- Animate the search process
- Compare A* with BFS or DFS
- Allow diagonal movements

Expected Output

- The shortest path from `S` to `G`
- Or the message: "No path found"