class Node:

    def \_\_init\_\_(self, position, parent=None):

        self.position = position

        self.parent = parent

        self.g = 0

        self.h = 0

        self.f = 0

    def \_\_eq\_\_(self, other):

        return self.position == other.position

def heuristic(node, goal):

    return abs(node.position[0] - goal.position[0]) + abs(node.position[1] - goal.position[1])

def a\_star\_algorithm(start\_pos, end\_pos, grid):

    start\_node = Node(start\_pos)

    end\_node = Node(end\_pos)

    open\_list = []

    closed\_list = []

    open\_list.append(start\_node)

    while open\_list:

        current\_node = min(open\_list, key=lambda node: node.f)

        if current\_node == end\_node:

            path = []

            while current\_node:

                path.append(current\_node.position)

                current\_node = current\_node.parent

            return path[::-1]

        open\_list.remove(current\_node)

        closed\_list.append(current\_node)

        neighbors = [

            (0, -1),

            (0, 1),

            (-1, 0),

            (1, 0)

        ]

        for neighbor\_offset in neighbors:

            neighbor\_pos = (current\_node.position[0] + neighbor\_offset[0], current\_node.position[1] + neighbor\_offset[1])

            if neighbor\_pos[0] < 0 or neighbor\_pos[0] >= len(grid) or neighbor\_pos[1] < 0 or neighbor\_pos[1] >= len(grid[0]):

                continue

            if grid[neighbor\_pos[0]][neighbor\_pos[1]] != 0:

                continue

            neighbor\_node = Node(neighbor\_pos, current\_node)

            if neighbor\_node in closed\_list:

                continue

            neighbor\_node.g = current\_node.g + 1

            neighbor\_node.h = heuristic(neighbor\_node, end\_node)

            neighbor\_node.f = neighbor\_node.g + neighbor\_node.h

            if neighbor\_node in open\_list:

                open\_node = next(node for node in open\_list if node == neighbor\_node)

                if neighbor\_node.g >= open\_node.g:

                    continue

            open\_list.append(neighbor\_node)

    return None

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)

end = (4, 4)

path = a\_star\_algorithm(start, end, grid)

if path:

    print("Path found:", path)

else:

    print("No path found")