import heapq

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

def heuristic(a, b):

# Manhattan distance

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

def get\_neighbors(node):

neighbors = []

x, y = node

for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:

nx, ny = x + dx, y + dy

if 0 <= nx < len(grid) and 0 <= ny < len(grid[0]) and grid[nx][ny] == 0:

neighbors.append((nx, ny))

return neighbors

open\_list = []

heapq.heappush(open\_list, (0 + heuristic(start, goal), 0, start))

came\_from = {}

cost\_so\_far = {start: 0}

while open\_list:

\_, current\_cost, current = heapq.heappop(open\_list)

if current == goal:

path = []

while current in came\_from:

path.append(current)

current = came\_from[current]

path.append(start)

return path[::-1]

for neighbor in get\_neighbors(current):

new\_cost = current\_cost + 1

if neighbor not in cost\_so\_far or new\_cost < cost\_so\_far[neighbor]:

cost\_so\_far[neighbor] = new\_cost

priority = new\_cost + heuristic(neighbor, goal)

heapq.heappush(open\_list, (priority, new\_cost, neighbor))

came\_from[neighbor] = current

return None

# Example grid (0 = open, 1 = blocked)

grid = [

[0, 0, 0, 0, 0],

[0, 1, 1, 1, 0],

[0, 1, 0, 0, 0],

[0, 1, 0, 1, 0],

[0, 0, 0, 0, 0]

]

# Call A\* algorithm

start = (0, 0)

goal = (4, 4)

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

if path:

print("Path found:", path)

else:

print("No path found")

Output:

