import heapq

class Node:

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

self.position = position

self.parent = parent

self.g = 0 # Cost from start node

self.h = 0 # Heuristic cost

self.f = 0 # Total cost

def \_lt\_(self, other):

return self.f < other.f

def heuristic(a, b):

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

def astar(grid, start, end):

open\_list = []

closed\_set = set()

start\_node = Node(start)

end\_node = Node(end)

heapq.heappush(open\_list, start\_node)

while open\_list:

current\_node = heapq.heappop(open\_list)

if current\_node.position == end\_node.position:

path = []

while current\_node:

path.append(current\_node.position)

current\_node = current\_node.parent

return path[::-1] # Return reversed path

closed\_set.add(current\_node.position)

for move in [(0, -1), (0, 1), (-1, 0), (1, 0)]:

node\_position = (current\_node.position[0] + move[0], current\_node.position[1] + move[1])

if node\_position[0] < 0 or node\_position[0] >= len(grid) or node\_position[1] < 0 or node\_position[1] >= len(grid[0]):

continue

if grid[node\_position[0]][node\_position[1]] == 1:

continue

if node\_position in closed\_set:

continue

neighbor = Node(node\_position, current\_node)

neighbor.g = current\_node.g + 1

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

neighbor.f = neighbor.g + neighbor.h

if any(open\_node.position == neighbor.position and open\_node.f <= neighbor.f for open\_node in open\_list):

continue

heapq.heappush(open\_list, neighbor)

return None # No path found

# Example Grid (0 = free space, 1 = obstacle)

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)

end = (4, 4)

path = astar(grid, start, end)

print("Path:", path)