import heapq

class Node:

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

self.position = position

self.parent = parent

self.g = 0 # Distance from start node

self.h = 0 # Heuristic (estimated cost to end)

self.f = 0 # Total cost

def \_\_lt\_\_(self, other):

return self.f < other.f

def heuristic(a, b):

# Manhattan distance

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

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)

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

# Goal check

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

# Explore neighbors

x, y = current\_node.position

neighbors = [(x + dx, y + dy) for dx, dy in

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

for nx, ny in neighbors:

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

neighbor\_pos = (nx, ny)

if neighbor\_pos in closed\_set:

continue

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

neighbor.g = current\_node.g + 1

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

neighbor.f = neighbor.g + neighbor.h

if any(open\_node.position == neighbor\_pos 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, 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]

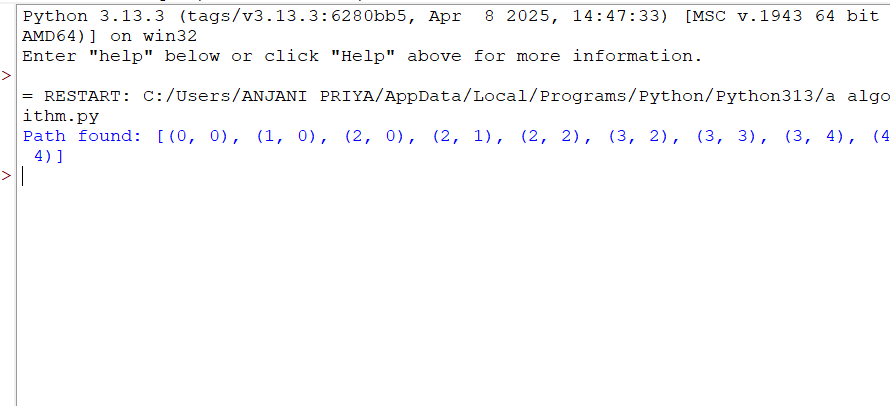
]

start = (0, 0)

end = (4, 4)

path = astar(grid, start, end)

print("Path found:", path)



8