class Node:

def \_\_init\_\_(self, position, g\_cost=0, h\_cost=0, parent=None):

self.position = position # (x, y)

self.g\_cost = g\_cost # Cost from start node

self.h\_cost = h\_cost # Heuristic to goal

self.parent = parent # To reconstruct path

def f\_cost(self):

return self.g\_cost + self.h\_cost

def get\_neighbors(position, grid):

x, y = position

neighbors = []

directions = [(-1, 0), (1, 0), (0, -1), (0, 1)] # up, down, left, right

for dx, dy in directions:

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

def manhattan\_heuristic(position, goal):

x1, y1 = position

x2, y2 = goal

return abs(x1 - x2) + abs(y1 - y2)

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

open\_list = []

closed\_list = set()

start\_node = Node(start, g\_cost=0, h\_cost=heuristic(start, goal))

open\_list.append(start\_node)

while open\_list:

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

open\_list.remove(current\_node)

if current\_node.position == goal:

# Reconstruct path

path = []

while current\_node:

path.append(current\_node.position)

current\_node = current\_node.parent

return path[::-1]

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

for neighbor\_pos in get\_neighbors(current\_node.position, grid):

if neighbor\_pos in closed\_list:

continue

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

h\_cost = heuristic(neighbor\_pos, goal)

neighbor\_node = Node(neighbor\_pos, g\_cost, h\_cost, parent=current\_node)

skip = False

for node in open\_list:

if node.position == neighbor\_pos and node.f\_cost() <= neighbor\_node.f\_cost():

skip = True

break

if not skip:

open\_list.append(neighbor\_node)

return None

# === User Input Section ===

def main():

rows = int(input("Enter number of rows in the grid: "))

cols = int(input("Enter number of columns in the grid: "))

print("\nEnter the grid row by row (use 0 for free cell, 1 for obstacle):")

grid = []

for i in range(rows):

while True:

row = input(f"Row {i + 1}: ").strip().split()

if len(row) == cols and all(cell in ('0', '1') for cell in row):

grid.append([int(cell) for cell in row])

break

else:

print(f"Please enter {cols} values (0 or 1) separated by space.")

while True:

try:

start = tuple(map(int, input("\nEnter start position (row col): ").split()))

goal = tuple(map(int, input("Enter goal position (row col): ").split()))

if (

len(start) == 2 and len(goal) == 2 and

grid[start[0]][start[1]] == 0 and

grid[goal[0]][goal[1]] == 0

):

break

else:

print("Start and goal must be valid, unblocked positions.")

except:

print("Invalid input. Use row and column indexes like: 0 0")

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

if path:

print("\nPath found:")

print(" -> ".join(str(p) for p in path))

else:

print("\nNo path found.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Enter number of rows in the grid: 5

Enter number of columns in the grid: 5

Enter the grid row by row (use 0 for free cell, 1 for obstacle):

Row 1: 0 0 0 0 0

Row 2: 0 1 1 0 0

Row 3: 0 0 0 1 0

Row 4: 0 1 0 0 0

Row 5: 0 0 0 0 0

Enter start position (row col): 0 0

Enter goal position (row col): 4 4