import numpy as np

# Manhattan distance heuristic

def manhattan\_distance(state, goal\_state):

total\_distance = 0

for i in range(3):

for j in range(3):

value = state[i, j]

if value != 0:

goal\_pos = np.argwhere(goal\_state == value)[0]

distance = abs(i - goal\_pos[0]) + abs(j - goal\_pos[1])

total\_distance += distance

return total\_distance

# Generate all possible moves for a given state

def generate\_moves(state):

moves = []

empty\_pos = np.argwhere(state == 0)[0]

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

new\_pos = empty\_pos + move

if 0 <= new\_pos[0] < 3 and 0 <= new\_pos[1] < 3:

new\_state = state.copy()

new\_state[empty\_pos[0], empty\_pos[1]] = state[new\_pos[0], new\_pos[1]]

new\_state[new\_pos[0], new\_pos[1]] = 0

moves.append(new\_state)

return moves

# Steepest ascent hill-climbing algorithm

def steepest\_ascent(initial\_state, goal\_state):

current\_state = initial\_state.copy()

level = 1

while True:

print("\nLevel:", level)

print("Current State:\n", current\_state)

current\_distance = manhattan\_distance(current\_state, goal\_state)

print("Heuristic Value:", current\_distance)

best\_next\_state = None

best\_next\_distance = float("inf")

for move in generate\_moves(current\_state):

move\_distance = manhattan\_distance(move, goal\_state)

if move\_distance < best\_next\_distance:

best\_next\_state = move

best\_next\_distance = move\_distance

if best\_next\_distance >= current\_distance:

print("\nReached local maximum. Final state:\n", current\_state)

break

current\_state = best\_next\_state

level += 1

initial\_state = np.array([[5,0,8],

[4,2,1],

[7,3,6]])

goal\_state = np.array([[1,2,3],

[4,5,6],

[7, 8,0]])

# Solve the puzzle using steepest ascent

steepest\_ascent(initial\_state, goal\_state)