import heapq

# Define goal state

goal\_state = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 0]

]

# Directions: (dx, dy) for up, down, left, right

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

def manhattan\_distance(state):

distance = 0

for i in range(3):

for j in range(3):

value = state[i][j]

if value == 0:

continue

goal\_x = (value - 1) // 3

goal\_y = (value - 1) % 3

distance += abs(i - goal\_x) + abs(j - goal\_y)

return distance

def find\_zero(state):

for i in range(3):

for j in range(3):

if state[i][j] == 0:

return i, j

def valid(x, y):

return 0 <= x < 3 and 0 <= y < 3

def generate\_successors(state):

x, y = find\_zero(state)

successors = []

for dx, dy in moves:

nx, ny = x + dx, y + dy

if valid(nx, ny):

new\_state = [row[:] for row in state]

new\_state[x][y], new\_state[nx][ny] = new\_state[nx][ny], new\_state[x][y]

successors.append(new\_state)

return successors

def is\_goal(state):

return state == goal\_state

def state\_to\_tuple(state):

return tuple(tuple(row) for row in state)

def a\_star(start\_state):

open\_list = []

closed\_set = set()

heapq.heappush(open\_list, (manhattan\_distance(start\_state), 0, start\_state, []))

while open\_list:

f, g, current, path = heapq.heappop(open\_list)

if is\_goal(current):

return path + [current]

current\_tuple = state\_to\_tuple(current)

if current\_tuple in closed\_set:

continue

closed\_set.add(current\_tuple)

for neighbor in generate\_successors(current):

if state\_to\_tuple(neighbor) in closed\_set:

continue

heapq.heappush(open\_list, (

g + 1 + manhattan\_distance(neighbor),

g + 1,

neighbor,

path + [current]

))

return None

def print\_state(state):

for row in state:

print(" ".join(str(n) if n != 0 else " " for n in row))

print()

# Example initial state

initial\_state = [

[1, 2, 3],

[4, 0, 6],

[7, 5, 8]

]

solution = a\_star(initial\_state)

if solution:

print("Steps to solve the 8-puzzle:")

for step in solution:

print\_state(step)

else:

print("No solution found.")

