import heapq

GOAL\_STATE = ((1, 2, 3),

(4, 5, 6),

(7, 8, 0)) # 0 is the empty tile

DIRECTIONS = [(-1, 0), (1, 0), (0, -1), (0, 1)] # Up, Down, Left, Right

def manhattan\_distance(state):

distance = 0

for i in range(3):

for j in range(3):

value = state[i][j]

if value != 0:

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

goal\_y = (value - 1) % 3

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

return distance

def get\_neighbors(state):

neighbors = []

# Find empty tile (0)

for i in range(3):

for j in range(3):

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

x, y = i, j

break

for dx, dy in DIRECTIONS:

nx, ny = x + dx, y + dy

if 0 <= nx < 3 and 0 <= ny < 3:

new\_state = [list(row) for row in state]

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

neighbors.append(tuple(tuple(row) for row in new\_state))

return neighbors

def reconstruct\_path(came\_from, g\_score, current):

path = []

while current in came\_from:

path.append((current, g\_score[current], manhattan\_distance(current)))

current = came\_from[current]

path.append((current, g\_score[current], manhattan\_distance(current))) # add start state

path.reverse()

return path

def a\_star(start\_state):

open\_set = []

heapq.heappush(open\_set, (manhattan\_distance(start\_state), 0, start\_state))

came\_from = {}

g\_score = {start\_state: 0}

while open\_set:

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

if current == GOAL\_STATE:

return reconstruct\_path(came\_from, g\_score, current)

for neighbor in get\_neighbors(current):

tentative\_g = g + 1

if neighbor not in g\_score or tentative\_g < g\_score[neighbor]:

came\_from[neighbor] = current

g\_score[neighbor] = tentative\_g

f\_score = tentative\_g + manhattan\_distance(neighbor)

heapq.heappush(open\_set, (f\_score, tentative\_g, neighbor))

return None

def print\_state(state, g, h):

print(f"g(n) = {g}, h(n) = {h}, f(n) = {g + h}")

for row in state:

print(' '.join(str(x) if x != 0 else ' ' for x in row))

print()

# Example solvable start state:

start\_state = ((1, 2, 3),

(4, 0, 6),

(7, 5, 8))

solution = a\_star(start\_state)

if solution:

print(f"Solution found in {len(solution) - 1} moves:\n")

for state, g, h in solution:

print\_state(state, g, h)

else:

print("No solution found.")