from queue import PriorityQueue

# Define goal state

goal\_state = [[1, 2, 3],

[4, 5, 6],

[7, 8, 0]]

# Define heuristic function (misplaced tiles)

def heuristic\_place(state):

misplaced\_tiles = 0

for i in range(3):

for j in range(3):

if state[i][j] != 0 and state[i][j] != goal\_state[i][j]:

misplaced\_tiles += 1

return misplaced\_tiles

def best\_first\_search(initial\_state, heuristic\_func):

priority\_queue = PriorityQueue()

priority\_queue.put((heuristic\_func(initial\_state), initial\_state))

visited = set()

parent = {} # To store parent-child relationships

while not priority\_queue.empty():

heuristic\_value, current\_state = priority\_queue.get()

visited.add(tuple(map(tuple, current\_state))) # Convert to tuple for hashing

if current\_state == goal\_state:

# Reconstruct path and count moves

moves = 0

path = [current\_state]

while current\_state != initial\_state:

current\_state = parent[tuple(map(tuple, current\_state))]

path.insert(0, current\_state)

moves += 1

return path, moves # Solution found

# Generate successor states

for next\_state in generate\_successors(current\_state):

if tuple(map(tuple, next\_state)) not in visited:

next\_heuristic = heuristic\_func(next\_state)

priority\_queue.put((next\_heuristic, next\_state))

parent[tuple(map(tuple, next\_state))] = current\_state

return None, -1 # No solution found

def generate\_successors(state):

successors = []

for i in range(3):

for j in range(3):

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

for dx, dy in [(1, 0), (-1, 0), (0, 1), (0, -1)]:

if 0 <= i + dx < 3 and 0 <= j + dy < 3:

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

new\_state[i][j], new\_state[i + dx][j + dy] = new\_state[i + dx][j + dy], new\_state[i][j]

successors.append(new\_state)

return successors

# Example usage

initial\_state = [[1, 2, 3],

[4, 0, 5],

[7, 8, 6]]

path, moves = best\_first\_search(initial\_state, heuristic\_place)

if path:

print("Solution found in", moves, "moves:")

for idx, (state, heuristic) in enumerate(zip(path, [heuristic\_place(state) for state in path])):

print("Move", idx)

for row in state:

print(row)

print("Heuristic value:", heuristic)

else:

print("No solution found.")