from collections import deque

# Define the starting and goal states

start\_state = (3, 3, 1) # (Missionaries, Cannibals, Boat on left side)

goal\_state = (0, 0, 0)

# Valid combinations for boat moves

moves = [

(1, 0), # 1 missionary

(2, 0), # 2 missionaries

(0, 1), # 1 cannibal

(0, 2), # 2 cannibals

(1, 1), # 1 missionary and 1 cannibal

]

def is\_valid(state):

m, c, \_ = state

m\_right = 3 - m

c\_right = 3 - c

# Check if within valid bounds

if m < 0 or c < 0 or m > 3 or c > 3:

return False

if m\_right < 0 or c\_right < 0 or m\_right > 3 or c\_right > 3:

return False

# Check if missionaries are outnumbered on either side

if (m > 0 and m < c) or (m\_right > 0 and m\_right < c\_right):

return False

return True

def bfs():

queue = deque()

queue.append((start\_state, []))

visited = set()

visited.add(start\_state)

while queue:

current\_state, path = queue.popleft()

if current\_state == goal\_state:

return path + [current\_state]

m, c, b = current\_state

for move\_m, move\_c in moves:

if b == 1:

# Move from left to right

new\_state = (m - move\_m, c - move\_c, 0)

else:

# Move from right to left

new\_state = (m + move\_m, c + move\_c, 1)

if is\_valid(new\_state) and new\_state not in visited:

visited.add(new\_state)

queue.append((new\_state, path + [current\_state]))

return None

# Run the solver

solution = bfs()

# Display the result

if solution:

print("Solution path (Missionaries, Cannibals, Boat side):")

for step in solution:

print(step)

else:

print("No solution found.")

OUTPUT:

