from collections import deque

# State: (missionaries\_left, cannibals\_left, boat\_side)

# boat\_side: 0 = left, 1 = right

def is\_valid(m\_left, c\_left, m\_right, c\_right):

if m\_left < 0 or c\_left < 0 or m\_right < 0 or c\_right < 0:

return False

if m\_left > 0 and m\_left < c\_left:

return False

if m\_right > 0 and m\_right < c\_right:

return False

return True

def get\_successors(state):

successors = []

m\_left, c\_left, boat = state

m\_right = 3 - m\_left

c\_right = 3 - c\_left

moves = [

(1, 0), (2, 0), # One or two missionaries

(0, 1), (0, 2), # One or two cannibals

(1, 1) # One of each

]

for m, c in moves:

if boat == 0: # Boat on left side

new\_state = (m\_left - m, c\_left - c, 1)

else: # Boat on right side

new\_state = (m\_left + m, c\_left + c, 0)

new\_m\_left, new\_c\_left, new\_boat = new\_state

new\_m\_right = 3 - new\_m\_left

new\_c\_right = 3 - new\_c\_left

if is\_valid(new\_m\_left, new\_c\_left, new\_m\_right, new\_c\_right):

successors.append(new\_state)

return successors

def bfs():

start = (3, 3, 0)

goal = (0, 0, 1)

queue = deque([(start, [start])])

visited = set()

while queue:

current, path = queue.popleft()

if current in visited:

continue

visited.add(current)

if current == goal:

return path

for successor in get\_successors(current):

queue.append((successor, path + [successor]))

return None

def print\_solution(path):

print("Solution Steps:")

for step in path:

m\_left, c\_left, boat = step

side = "left" if boat == 0 else "right"

print(f"Left: M={m\_left}, C={c\_left} | Boat: {side}")

# Run the solver

solution = bfs()

if solution:

print\_solution(solution)

else:

print("No solution found.")

