from collections import deque

def is\_valid(state):

m1, c1, m2, c2, boat = state

if m1 < 0 or c1 < 0 or m2 < 0 or c2 < 0:

return False

if (m1 > 0 and m1 < c1) or (m2 > 0 and m2 < c2):

return False

return True

def get\_successors(state):

m1, c1, m2, c2, boat = state

moves = [(1,0), (2,0), (0,1), (0,2), (1,1)]

successors = []

for m, c in moves:

if boat == 1: # Boat on left

new\_state = (m1 - m, c1 - c, m2 + m, c2 + c, 0)

else: # Boat on right

new\_state = (m1 + m, c1 + c, m2 - m, c2 - c, 1)

if is\_valid(new\_state):

successors.append(new\_state)

return successors

def bfs():

start = (3, 3, 0, 0, 1)

goal = (0, 0, 3, 3, 0)

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

visited = set()

while queue:

state, path = queue.popleft()

if state in visited:

continue

visited.add(state)

if state == goal:

return path

for succ in get\_successors(state):

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

return None

# Run the solver

solution = bfs()

if solution:

for step in solution:

print(f"Left: {step[0]}M {step[1]}C | Right: {step[2]}M {step[3]}C | Boat on {'Left' if step[4] == 1 else 'Right'}")

else:

print("No solution found.")

A screenshot of a computer

AI-generated content may be incorrect.