from collections import deque

def is\_goal(state, goal):

return goal in state

def get\_successors(state, a\_cap, b\_cap):

a, b = state

successors = set()

# Fill Jug A

successors.add((a\_cap, b))

# Fill Jug B

successors.add((a, b\_cap))

# Empty Jug A

successors.add((0, b))

# Empty Jug B

successors.add((a, 0))

# Pour A → B

transfer = min(a, b\_cap - b)

successors.add((a - transfer, b + transfer))

# Pour B → A

transfer = min(b, a\_cap - a)

successors.add((a + transfer, b - transfer))

return successors

def bfs(a\_cap, b\_cap, goal):

start = (0, 0)

visited = set()

queue = deque([(start, [])]) # (state, path)

while queue:

current, path = queue.popleft()

if current in visited:

continue

visited.add(current)

path = path + [current]

if is\_goal(current, goal):

return path

for next\_state in get\_successors(current, a\_cap, b\_cap):

if next\_state not in visited:

queue.append((next\_state, path))

return None

# Example usage

a\_capacity = 4

b\_capacity = 3

goal\_amount = 2

solution = bfs(a\_capacity, b\_capacity, goal\_amount)

if solution:

print("Steps to reach the goal:")

for step in solution:

print(f"Jug A: {step[0]}L, Jug B: {step[1]}L")

else:

print("No solution found.")

