from collections import deque

# Function to check if the state is already visited

def is\_visited(state, visited):

return state in visited

# Function to print the path from start to goal

def print\_path(parent, goal):

path = []

while goal is not None:

path.append(goal)

goal = parent.get(goal)

path.reverse()

print("Steps to reach the goal:")

for step in path:

print(f"Jug1: {step[0]}L, Jug2: {step[1]}L")

# BFS to solve water jug problem

def water\_jug\_problem(jug1\_capacity, jug2\_capacity, target):

visited = set()

parent = {}

# Initial state is both jugs empty

start = (0, 0)

queue = deque([start])

visited.add(start)

parent[start] = None

while queue:

jug1, jug2 = queue.popleft()

# Check if the goal is reached

if jug1 == target or jug2 == target:

print\_path(parent, (jug1, jug2))

return

# List of all possible next states

possible\_states = [

(jug1\_capacity, jug2), # Fill jug1

(jug1, jug2\_capacity), # Fill jug2

(0, jug2), # Empty jug1

(jug1, 0), # Empty jug2

# Pour jug1 -> jug2

(jug1 - min(jug1, jug2\_capacity - jug2), jug2 + min(jug1, jug2\_capacity - jug2)),

# Pour jug2 -> jug1

(jug1 + min(jug2, jug1\_capacity - jug1), jug2 - min(jug2, jug1\_capacity - jug1))

]

for state in possible\_states:

if state not in visited:

visited.add(state)

parent[state] = (jug1, jug2)

queue.append(state)

print("No solution found.")

# Example usage

jug1\_capacity = 4

jug2\_capacity = 3

target = 2

water\_jug\_problem(jug1\_capacity, jug2\_capacity, target)

OUTPUT:

