**Name: Muhammad Zeeshan**

**Roll no: 064**

**Section: BSAI-4B**

**Code:**

**from collections import deque**

**def waterJugProblem(capacity1, capacity2, goal):**

**queue = deque()**

**visited = set()**

**queue.append((0, 0, []))**

**visited.add((0, 0))**

**while queue:**

**jug1, jug2, path = queue.popleft()**

**path.append((jug1, jug2))**

**if jug1 == goal or jug2 == goal:**

**print("Solution Found!")**

**for action in path:**

**print(action)**

**return True**

**possible\_states = [**

**(capacity1, jug2),**

**(jug1, capacity2),**

**(0, jug2),**

**(jug1, 0),**

**(jug1 - min(jug1, capacity2 - jug2), jug2 + min(jug1, capacity2 - jug2)),**

**(jug1 + min(jug2, capacity1 - jug1), jug2 - min(jug2, capacity1 - jug1))**

**]**

**for state in possible\_states:**

**if state[:2] not in visited:**

**visited.add(state[:2])**

**queue.append((state[0], state[1], path.copy()))**

**print("No Solution Found")**

**return False**

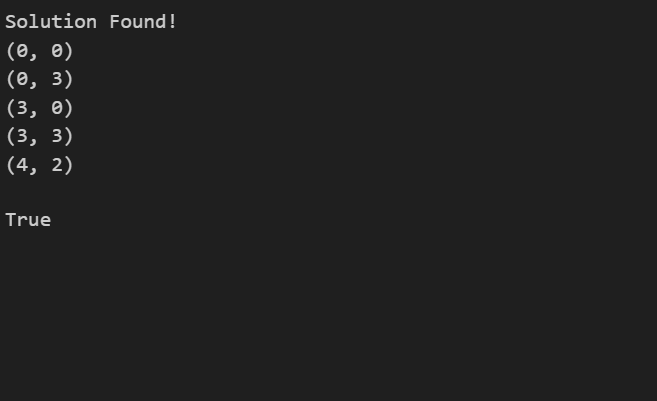
**jug1Capacity = 4**

**jug2Capacity = 3**

**target = 2**

**waterJugProblem(jug1Capacity, jug2Capacity, target)**

**Output:**

****

**Explanation**

The code solves the Water Jug Problem using a step-by-step approach. It starts by considering two empty jugs and explores all possible ways to fill, empty, or transfer water between them to reach the target amount.

First, it sets up a queue to keep track of jug states and a visited set to avoid repeating the same steps. It begins with both jugs empty (0,0) and uses Breadth-First Search (BFS) to explore possible actions.

At each step, it checks if either jug contains the desired amount of water. If so, it prints the solution path showing how water was moved to reach that state.

The code considers six possible actions:

1. Fill Jug 1 to its full capacity.
2. Fill Jug 2 to its full capacity.
3. Empty Jug 1 completely.
4. Empty Jug 2 completely.
5. Pour water from Jug 1 to Jug 2 without overflowing.
6. Pour water from Jug 2 to Jug 1 without overflowing.