Title: Implement the Water Jug problem using BFS and DFS

Ex. No.:01 **Reg.No.:RA2311003010210**

Date: 16/02/25 **Name:Jake Roberts**

Aim:

The goal is to use BFS and DFS to solve the Water Jug Problem in Python. Given two jugs of different capacities, we need to find a way to measure exactly Z liters by filling, emptying, or pouring between them. BFS helps find the shortest solution, while DFS explores different possibilities.

Procedure:

1. Define the problem by taking two jugs with capacities X and Y and determining whether it is possible to measure exactly Z liters using them.
2. Implement BFS (Breadth-First Search) by using a queue to explore all possible states, tracking visited states to avoid repetition, and stopping when a jug contains exactly Z liters.
3. Implement DFS (Depth-First Search) using recursion to explore different paths, marking visited states to prevent cycles, and returning any valid solution.
4. Take inputs for X, Y, and Z, then run both BFS and DFS to find possible solutions.
5. Compare the results of BFS and DFS to analyze their efficiency in solving the problem.
6. Display the sequence of steps for each algorithm, or indicate if no solution exists.

Program:

ex1.py ex2.py

ex2.py > dfs

```python
def is_valid(state, visited):
    return state not in visited
def get_next_states(state, X, Y):
    a, b = state
    return [
        (X, b), (a, Y), (0, b), (a, 0),  # Fill and empty operations
        (a - min(a, Y - b), b + min(a, Y - b)),  # Pour A → B
        (a + min(b, X - a), b - min(b, X - a))   # Pour B → A
    ]
def bfs(X, Y, Z):
    queue = [((0, 0), [])]  # (current state, path)
    visited = set()
    while queue:
        state, path = queue.pop(0)
        if state in visited:
            continue
        visited.add(state)

        if state[0] == Z or state[1] == Z:
            return path + [state]

        for next_state in get_next_states(state, X, Y):
            if is_valid(next_state, visited):
                queue.append((next_state, path + [state]))
    return None
def dfs(X, Y, Z, state=(0, 0), path=[], visited=set()):
    if state in visited:
        return None
    visited.add(state)
    if state[0] == Z or state[1] == Z:
        return path + [state]
    for next_state in get_next_states(state, X, Y):
        result = dfs(X, Y, Z, next_state, path + [state], visited)
        if result:
            return result
    return None
X, Y, Z = 4, 3, 2
print("BFS Solution:", bfs(X, Y, Z))
print("DFS Solution:", dfs(X, Y, Z))
```

Manual Output: Manual Calculation of BFS for water jug problem

(0, 0) → Fill Jug 1

(4, 0) → Pour from Jug 1 to Jug 2

(1, 3) → Empty Jug 2

(1, 0) → Pour from Jug 1 to Jug 2

(0, 1) → Fill Jug 1

(4, 1) → Pour from Jug 1 to Jug 2

(2, 3) → Goal reached (Jug 1 has 2 liters)

Manual Output: Manual Calculation of DFS for water jug problem

(0, 0) → Fill Jug 1

(4, 0) → Fill Jug 2

(4, 3) → Empty Jug 1

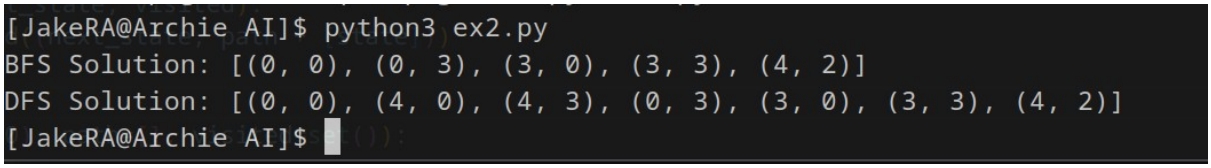
(0, 3) → Pour from Jug 2 to Jug 1

3, 0) → Fill Jug 2

(3, 3) → Pour from Jug 2 to Jug 1

(4, 2) → Goal reached (Jug 1 has 2 liters)

Screenshot of Output:



Result:
The Water Jug problem using BFS and DFS was successfully implemented.