Experiment No.: 3 DFS (Water Jug Problem)

Aim:

To solve the Water Jug Problem using the Depth-First Search (DFS) algorithm. The goal is to measure exactly 2 litres of water in the 4-liter jug while ensuring that the 3-liter jug is empty, using a series of operations like filling, emptying and pouring between the two jugs.

Algorithm:

1) State Representation: Represent the state of the jugs as a tuple (x, y) where x is the amount of water in the 4 liter jug and y is the amount of water in the 3-liter jug.
2) Initial State: Start with both jugs empty: $(0, 0)$.
3) Define Possible Operations:
   - Fill the 4-liter jug: $(4, y)$
   - Fill the 3-liter jug: $(x, 3)$
   - Empty the 4-liter jug: $(0, y)$
   - Empty the 3-liter jug: $(x, 0)$
   - Pour water from the 4-liter jug to the 3-liter jug until one is empty or the other is full: $(x - \min(y, 4-x), y - \min(y, 4-x))$.
   - Pour water from the 3-liter jug to the 4-liter jug until one is empty or the other is full: $(x + \min(y, 4-x), y - \min(y, 4-x))$.
4) Use DFS to Explore States:
   - Start from the initial state $(0,0)$ and use DFS to explore all possible states by applying the operations.
   - Mark each visited state to avoid revisiting.
   - If the state $(2, 0)$ is reached, where the 4-liter jug has 2 litres and the 3-liter jug is empty, the solution is found.

5) Backtracking: If a state leads to no further valid states, backtrack and try a different operation.

Program:

```
def is_valid_state(x,y):
    # Ensure that the state is within the capacity of the jugs
    return 0 <= x <= 4 and 0 <= y <= 3

def dfs (x,y, visited, path):
    if (x,y) in visited:
        return False

    # Mark the current state as visited
    visited.add((x,y))
    # store the current state in the path
    path.append ((x,y))

    # If we have 2 liters of water in 4 liter jug and 0 liters in 3 liter jug, the problem is solved.
    if x==2 and y==0:
        return True
    # possible moves
    possible_moves = [
      (4,y) , # Fill the 4-liter jug
      (x,3) , # Fill the 3-liter jug
      (0,y) , # Empty the 4-liter jug
      (x,0) , # Empty the 3-liter jug
      (x-min(x,3-y), y+min(x,3-y)), # pour from 4L to 3L
      (x+min(y,4-x), y-min(y,4-x)) # pour from 3L to 4L
    ]
```

```
    # Explore all possible moves using DFS
    for (next_x, next_y) in possible_moves:
        if is_valid_state(next_x, next_y) and
        dfs(next_x, next_y, visited, path):
            return True
    # Backtrack: remove the last state if it leads
                 to no solution
    path.pop()
    return False

def solve_water_jug_problem():
    initial_state = (0,0)
    visited = set()
    path = []
    if dfs(initial_state[0], initial_state[1], visited, path):
        print("Solution found!")
        for step in path:
            print(step)
    else:
        print("No solution exists")

# Main code
if __name__ == "__main__":
    solve_water_jug_problem()
```

Output:

solution found!

(0,0)

(0,3)

(3,0)

(3,3)

(4,2)

(0,2)

(2,0)

Result:

Thus, the program for Water Jug Problem is successfully executed and the output is verified