# 1. Implement and Demonstrate Depth First Search Algorithm on Water Jug Problem

```
from collections import defaultdict

jug1, jug2, aim = 4, 3, 2

visited = defaultdict(lambda: False)

def waterJugSolverDFS(amt1, amt2):
    if (amt1 == aim and amt2 == 0) or (amt2 == aim and amt1 == 0):
        print(amt1, amt2)
        return True

    visited[(amt1, amt2)] = True
    print(amt1, amt2)

    # Try all possible moves
    if not visited[(0, amt2)] and waterJugSolverDFS(0, amt2):
        return True
    if not visited[(amt1, 0)] and waterJugSolverDFS(amt1, 0):
        return True
    if not visited[(jug1, amt2)] and waterJugSolverDFS(jug1, amt2):
        return True
    if not visited[(amt1, jug2)] and waterJugSolverDFS(amt1, jug2):
        return True

    # Pour from jug1 to jug2
    pour_amt = min(amt1, jug2 - amt2)
    if not visited[(amt1 - pour_amt, amt2 + pour_amt)] and waterJugSolverDFS(amt1 - pour_amt,
amt2 + pour_amt):
        return True
```

```
    # Pour from jug2 to jug1
    pour_amt = min(amt2, jug1 - amt1)
    if not visited[(amt1 + pour_amt, amt2 - pour_amt)] and waterJugSolverDFS(amt1 + pour_amt, amt2 - pour_amt):
        return True

    return False

print("Steps:")
waterJugSolverDFS(0, 0)
```

## OUTPUT

```
Steps:
0 0
4 0
4 3
0 3
3 0
3 3
4 2
0 2
```

## Explanation

1. `from collections import defaultdict`: This line imports the `defaultdict` class from the `collections` module. `defaultdict` is a subclass of the built-in `dict` class, which provides a default value for keys that haven't been explicitly set.

2. `jug1, jug2, aim = 4, 3, 2`: This line initializes three variables `jug1`, `jug2`, and `aim` with values 4, 3, and 2 respectively. These variables represent the capacities of the two jugs and the desired amount of water to be measured.

3. `visited = defaultdict(lambda: False)`: This line creates a defaultdict named `visited` with a default value of False. This dictionary will be used to keep track of visited states during the DFS traversal.

4. `def waterJugSolverDFS(amt1, amt2):`: This line defines the main function `waterJugSolverDFS` that implements the DFS algorithm to solve the water jug problem. It takes two parameters `amt1` and `amt2`, representing the current amounts of water in jug1 and jug2 respectively.

5. `if (amt1 == aim and amt2 == 0) or (amt2 == aim and amt1 == 0):`: This line checks if the current state is the goal state, where either jug1 or jug2 contains `aim` liters of water. If so, it prints the current state and returns True to indicate that the goal is reached.

6. `visited[(amt1, amt2)] = True`: This line marks the current state as visited by setting the corresponding key in the `visited` dictionary to True.

7. `print(amt1, amt2)`: This line prints the current amounts of water in both jugs.

8. The subsequent lines explore all possible moves from the current state:

- Emptying jug1 and jug2 (`if not visited[(0, amt2)] and waterJugSolverDFS(0, amt2)`)

- Filling jug1 and jug2 to their capacities (`if not visited[(jug1, amt2)] and waterJugSolverDFS(jug1, amt2)`)

- Pouring water from jug1 to jug2 and vice versa (`if not visited[(amt1 - pour_amt, amt2 + pour_amt)] and waterJugSolverDFS(amt1 - pour_amt, amt2 + pour_amt)` and `if not visited[(amt1 + pour_amt, amt2 - pour_amt)] and waterJugSolverDFS(amt1 + pour_amt, amt2 - pour_amt)`)

9. `return False`: If none of the above moves lead to the goal state, the function returns False, indicating that the goal is not reachable from the current state.

10. `print("Steps:")`: This line prints the header for the steps that will be printed during the DFS traversal.

11. `waterJugSolverDFS(0, 0)`: This line calls the `waterJugSolverDFS` function with initial amounts of 0 in both jugs to start the DFS traversal from the initial state.

This code essentially implements a Depth-First Search (DFS) algorithm to explore all possible states of the water jug problem until the goal state is reached. During the traversal, it prints the steps taken to reach the goal state.