BFS Code for Water Jug Problem

```
from collections import deque

def bfs_water_jug(jug1_capacity, jug2_capacity, target_amount):
    queue = deque()
    visited = set()
    parent = {}  # To store the parent state for each state

    # Start with an empty jug state (0, 0)
    initial_state = (0, 0)
    queue.append(initial_state)
    visited.add(initial_state)

    while queue:
        current_state = queue.popleft()

        # Check if we found the target amount in either jug
        if target_amount in current_state:
            # Reconstruct the solution path
            solution_path = [current_state]
            while current_state != initial_state:
                current_state = parent[current_state]
                solution_path.append(current_state)
            solution_path.reverse()
            return solution_path

        # Fill jug1 to its maximum capacity
        new_state = (jug1_capacity, current_state[1])
        if new_state not in visited:
            queue.append(new_state)
            visited.add(new_state)
            parent[new_state] = current_state

        # Fill jug2 to its maximum capacity
        new_state = (current_state[0], jug2_capacity)
        if new_state not in visited:
            queue.append(new_state)
            visited.add(new_state)
            parent[new_state] = current_state

        # Pour water from jug1 to jug2
        pour_amount = min(current_state[0], jug2_capacity - current_state[1])
        new_state = (current_state[0] - pour_amount, current_state[1] + pour_amount)
        if new_state not in visited:
            queue.append(new_state)
            visited.add(new_state)
            parent[new_state] = current_state

        # Pour water from jug2 to jug1
        pour_amount = min(current_state[1], jug1_capacity - current_state[0])
        new_state = (current_state[0] + pour_amount, current_state[1] - pour_amount)
        if new_state not in visited:
            queue.append(new_state)
            visited.add(new_state)
            parent[new_state] = current_state

    # If the target amount cannot be achieved, return None
    return None

# Function to print the solution path
def print_solution_path(solution_path):
    for idx, state in enumerate(solution_path):
        print(f"Step {idx}: Jug 1: {state[0]} liters, Jug 2: {state[1]} liters")

# Example usage:
jug1_capacity = 4
jug2_capacity = 3
target_amount = 2

solution_path = bfs_water_jug(jug1_capacity, jug2_capacity, target_amount)
if solution_path:
    print("Solution Path:")
    print_solution_path(solution_path)
else:
    print(f"Target amount {target_amount} cannot be achieved with the given jug capacities.")
```

```
Solution Path:
Step 0: Jug 1: 0 liters, Jug 2: 0 liters
Step 1: Jug 1: 0 liters, Jug 2: 3 liters
Step 2: Jug 1: 3 liters, Jug 2: 0 liters
Step 3: Jug 1: 3 liters, Jug 2: 3 liters
Step 4: Jug 1: 4 liters, Jug 2: 2 liters
```

DFS Code for Water Jug Problem

```
def dfs_water_jug(jug1_capacity, jug2_capacity, target_amount):
    stack = []
    visited = set()
    parent = {}  # To store the parent state for each state

    # Start with an empty jug state (0, 0)
    initial_state = (0, 0)
    stack.append(initial_state)

    while stack:
        current_state = stack.pop()

        # Check if we found the target amount in either jug
        if target_amount in current_state:
            # Reconstruct the solution path
            solution_path = [current_state]
            while current_state != initial_state:
                current_state = parent[current_state]
                solution_path.append(current_state)
            solution_path.reverse()
            return solution_path

        # Fill jug1 to its maximum capacity
        new_state = (jug1_capacity, current_state[1])
        if new_state not in visited:
            stack.append(new_state)
            visited.add(new_state)
            parent[new_state] = current_state

        # Fill jug2 to its maximum capacity
        new_state = (current_state[0], jug2_capacity)
        if new_state not in visited:
            stack.append(new_state)
            visited.add(new_state)
            parent[new_state] = current_state

        # Pour water from jug1 to jug2
        pour_amount = min(current_state[0], jug2_capacity - current_state[1])
        new_state = (current_state[0] - pour_amount, current_state[1] + pour_amount)
        if new_state not in visited:
            stack.append(new_state)
            visited.add(new_state)
            parent[new_state] = current_state

        # Pour water from jug2 to jug1
        pour_amount = min(current_state[1], jug1_capacity - current_state[0])
        new_state = (current_state[0] + pour_amount, current_state[1] - pour_amount)
        if new_state not in visited:
            stack.append(new_state)
            visited.add(new_state)
            parent[new_state] = current_state

    # If the target amount cannot be achieved, return None
    return None

# Function to print the solution path
def print_solution_path(solution_path):
    for idx, state in enumerate(solution_path):
        print(f"Step {idx}: Jug 1: {state[0]} liters, Jug 2: {state[1]} liters")

# Example usage:
jug1_capacity = 4
jug2_capacity = 3
target_amount = 2

solution_path = dfs_water_jug(jug1_capacity, jug2_capacity, target_amount)
if solution_path:
    print("Solution Path:")
    print_solution_path(solution_path)
else:
    print(f"Target amount {target_amount} cannot be achieved with the given jug capacities.")
```

```
Solution Path:
Step 0: Jug 1: 0 liters, Jug 2: 0 liters
Step 1: Jug 1: 0 liters, Jug 2: 3 liters
Step 2: Jug 1: 3 liters, Jug 2: 0 liters
Step 3: Jug 1: 3 liters, Jug 2: 3 liters
Step 4: Jug 1: 4 liters, Jug 2: 2 liters
```

✓ 0s completed at 10:30 AM