****

**PROGRAMMING FOR AI (LAB)**

**Lab Task No 03**

**Submitted To :- Sir Rasikh Ali**

**Submitted By :- Aiman Ijaz**

**Roll No :- SU92-BSAIM-F23-010**

**Department :- Software Engineering**

**Section :- BSAI-4A**

**Water Jug Problem - Depth-First Search (DFS) Approach**

**Introduction**

The **Water Jug Problem** is a classic problem in artificial intelligence and algorithm design. It involves two jugs with specific capacities, and the goal is to measure an exact amount of water using a set of allowed operations.

This implementation uses **Depth-First Search (DFS)** with a **stack-based approach** to explore possible states and find a solution.

**Problem Statement**

Given two jugs with capacities capacity1 and capacity2, the objective is to determine whether it is possible to measure exactly goal liters using the allowed operations:

1. **Fill Jug 1 completely**
2. **Fill Jug 2 completely**
3. **Empty Jug 1 completely**
4. **Empty Jug 2 completely**
5. **Pour water from Jug 1 to Jug 2** until Jug 2 is full or Jug 1 is empty
6. **Pour water from Jug 2 to Jug 1** until Jug 1 is full or Jug 2 is empty

**How the Code Works**

**1. Initializing Data Structures**

* A **stack** is used to implement DFS, storing the current state of both jugs.
* A **visited set** keeps track of explored states to avoid infinite loops.
* An **actions list** records the steps taken to reach the goal.

**2. Algorithm Execution**

The program starts with **both jugs empty (0,0)** and follows these steps:

1. **Push (0,0) to the stack** and mark it as visited.
2. **Iterate while the stack is not empty:**
   * Pop the top state (jug1, jug2) from the stack.
   * Check if either jug1 or jug2 matches the goal.
   * If yes, print the solution and exit.
3. **Generate possible moves:**
   * Apply all six operations to transition to new states.
   * If a new state has not been visited, mark it visited and push it to the stack.
   * Print the performed action and the resulting state.
4. **If the stack is exhausted without finding the target, print "No Solution Found."**

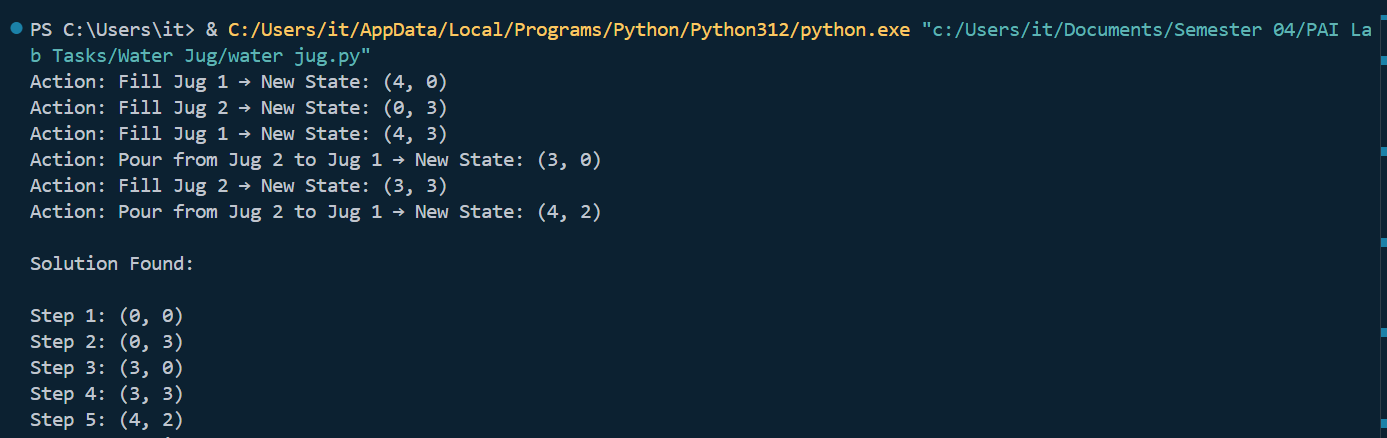
**Limitations and Improvements**

**DFS may not always find the shortest path** to the goal. A **BFS-based approach** would be better for optimal solutions.

**Conclusion**

The **Water Jug Problem** is effectively solved using DFS in this implementation. The algorithm explores all possible moves, avoids revisiting states, and terminates when a solution is found. While DFS ensures a solution if one exists, using **BFS or mathematical methods** could improve efficiency.

**Output:**

****