SUPERIOR UNIVERSITY

Name : **MUHAMMAD AHMAD**

Roll No : **SU92-BSAIM-F23-135**

Section : **4-C**

Task : **03**

Date : **05th March, 2025**

Subject : **Programming AI Lab**

Submitted To : **Prof Rasikh Ali**

TASK : 03

Water Jug Problem

1. Introduction The Water Jug Problem is a classic puzzle that involves two jugs of different capacities and a goal to measure a specific amount of water using them. This implementation solves the problem using a depth-first search (DFS) approach, ensuring that all possible states are explored efficiently.

2. Approach The algorithm follows a stack-based DFS method to explore different water states of two jugs. It maintains a set of visited states to prevent cycles and unnecessary computations. The solution is found when either of the jugs contains the desired amount of water.

3. Code Explanation

- **State Representation:** Each state is represented as a tuple (jug1, jug2) where jug1 and jug2 denote the current amounts of water in each jug.
- **State Transition Rules:**
    1. Fill Jug 1 completely.
    2. Fill Jug 2 completely.
    3. Empty Jug 1.
    4. Empty Jug 2.
    5. Pour water from Jug 1 to Jug 2 without overflowing.
    6. Pour water from Jug 2 to Jug 1 without overflowing.
- **Algorithm Flow:**
    1. Initialize a stack with the starting state (0,0) and a set for visited states.
    2. Explore possible moves by applying transition rules.
    3. If a new state is reached, store the move and continue searching.
    4. If the goal amount is found, print the sequence of actions and exit.

4. Implementation Issues and Fixes

- The original implementation contained errors in rule application, leading to incorrect transitions.
- The rule conditions were refined to ensure no invalid or redundant moves.

- The tracking of rule numbers was fixed to correctly associate each step with its corresponding transition.

5. Conclusion This implementation successfully finds a solution to the Water Jug Problem using an optimized DFS approach. The use of visited states prevents unnecessary recomputations, making the algorithm efficient and ensuring it terminates when a solution is found.