Lab Task 3 (Report)

# Water Jug Problem using DFS

**1. Introduction**

* The Water Jug problem is a classic puzzle involving two jugs with specific capacities and a desired quantity of water.
* The objective is to find a sequence of steps (filling, emptying, pouring) to measure the target quantity using the given jugs.
* This report presents a solution using the Depth-First Search (DFS) algorithm.

**2. Depth-First Search Algorithm**

* DFS is a graph traversal algorithm that explores as far as possible along each branch before backtracking.
* It systematically explores states and transitions until a goal state is reached or all possibilities are complete.

**3. Code Implementation**

* The code tells a dfs\_waterjug function to perform the DFS search.
* It uses a visited set to track explored states and avoid cycles.
* A stack data structure stores states to be explored.
* Six rules (filling, emptying, pouring) are implemented to generate possible state transitions.

**4. Applying DFS to Water Jug Problem**

* The initial state is both jugs empty (0, 0).
* DFS explores states based on the six rules.
* If the target quantity is reached in either jug, the path is returned as the solution.
* If all reachable states are explored without finding the target, no solution exists.

**5. Testing**

* The code demonstrates an example with jug capacities of 4 and 3, and a target of 2.
* The DFS algorithm is applied, and the solution (sequence of steps) is printed.

The code implements a Depth-First Search (DFS) algorithm to solve the classic Water Jug problem.

### Code Explanation:

1. **Initialization:**
   * visited: A set to store visited states (water levels in the jugs) to avoid cycles.
   * stack: A stack to store states to be explored, initialized with the starting state (both jugs empty).
2. **DFS Iteration:**
   * While the stack is not empty, pop a state from the stack.
   * If the current state is a goal state (either jug contains the desired amount of water), return the path taken to reach this state.
   * If the current state has already been visited, skip it.
   * Otherwise, mark the current state as visited.
3. **Applying Rules:**
   * The code applies six rules representing possible actions:
     + Fill jug1
     + Fill jug2
     + Empty jug1
     + Empty jug2
     + Pour jug1 to jug2
     + Pour jug2 to jug1
   * For each applicable rule, generate the new state and add it to the stack if it hasn't been visited.
4. **Output:**
   * If a solution is found, print the steps taken to reach the goal state.
   * If no solution is found, print "no solution".

* 