****

**Superior University Lahore**

**Instructor**: **Sir Rasikh Ali**

**Subject: Programming for AI**

**Task 03**

**Water jug Problem**  
**BS in Artificial Intelligence**

Table of Contents

[**Introduction** 1](#_Toc192418843)

[**Problem Statement** 1](#_Toc192418844)

[**Implementation Details** 2](#_Toc192418845)

[**Algorithm Steps** 2](#_Toc192418846)

[**Python Implementation** 2](#_Toc192418847)

[**Execution & Output** 3](#_Toc192418848)

[**Output Image** 3](#_Toc192418849)

[**Conclusion** 3](#_Toc192418850)

**Introduction**

The **Water Jug Problem** is a classic problem in artificial intelligence and graph theory. It involves two jugs with fixed capacities and the goal of measuring a specific amount of water using the least number of steps. In this report, we solve this problem using the **Depth First Search (DFS)** approach.

**Problem Statement**

Given two jugs with capacities **jug1Capacity** and **jug2Capacity**, we need to measure exactly **target** liters using the following operations:

1. Fill jug1 to full capacity.
2. Fill jug2 to full capacity.
3. Empty jug1 completely.
4. Empty jug2 completely.
5. Pour water from jug1 into jug2 until jug2 is full or jug1 is empty.
6. Pour water from jug2 into jug1 until jug1 is full or jug2 is empty.
7. Pour all water from jug1 into jug2.
8. Pour all water from jug2 into jug1.

We use **DFS** to explore all possible states and find a solution.

**Implementation Details**

**Algorithm Steps**

1. **Initialize a Stack**: Start with an empty stack and push the initial state (0,0), meaning both jugs are empty.
2. **Track Visited States**: Maintain a set to avoid revisiting already explored states.
3. **DFS Exploration**:
   * Pop the top state from the stack (LIFO order ensures deep exploration).
   * Apply all possible operations (rules) and generate new states.
   * If a state is the goal (goal, \_) or (\_, goal), print the solution path.
   * If not visited, push the new state into the stack.
4. **Continue until stack is empty**.
5. **If no solution is found, return failure.**

**Python Implementation**

# Water Jug Problem using DFS

def waterJugDFS(capacity1, capacity2, goal):

stack = []

visited = set()

stack.append((0, 0))

visited.add((0, 0))

actions = []

while stack:

jug1, jug2 = stack.pop()

actions.append((jug1, jug2))

if jug1 == goal or jug2 == goal:

print("Solution Found")

for action in actions:

print(action)

return True

rules = [

(capacity1, jug2), # Fill jug1

(jug1, capacity2), # Fill jug2

(0, jug2), # Empty jug1

(jug1, 0), # Empty jug2

(jug1 - min(jug1, capacity2 - jug2), jug2 + min(jug1, capacity2 - jug2)), # Pour jug1 → jug2

(jug1 + min(jug2, capacity1 - jug1), jug2 - min(jug2, capacity1 - jug1)), # Pour jug2 → jug1

(jug1 + jug2, 0), # Pour all jug1 → jug2

(0, jug1 + jug2), # Pour all jug2 → jug1

]

for state in rules:

if state not in visited:

visited.add(state)

stack.append(state)

print("No Solution Found")

return False

# Test Case

jug1Capacity = 4

jug2Capacity = 3

target = 2

waterJugDFS(jug1Capacity, jug2Capacity, target)

**Execution & Output**

For **Jug 1 Capacity = 4**, **Jug 2 Capacity = 3**, and **Target = 2**, the solution is:

Solution Found

(0, 0)

(4, 0)

(1, 3)

(1, 0)

(0, 1)

(4, 1)

(2, 3)

**Output Image**

**Conclusion**

* This implementation successfully solves the Water Jug Problem using **Depth First Search (DFS)**.
* The algorithm explores states deeply before backtracking, making it efficient for this type of problem.
* The solution sequence is recorded, ensuring we can trace back the steps taken.
* The **DFS approach is useful in scenarios where depth exploration is preferred over breadth exploration**.

This implementation can be modified further to optimize memory usage or improve efficiency by **limiting unnecessary state expansions**.