**THE SUPERIOR UNIVERSITY LAHORE** 

**LAB#3**

**Semester: 4th Se~~ctio~~n: AI (B)**

**Faculty of Computer Science and Information Technology Deadline:**

**Subject: PAI LAB Total Marks:**

**Name: M.Hassan Shahid**

**Roll No: 059**

***Instructions:***

* Copying of the assignment willresult in failure.
* Assignment should be submitted in word or pdf.

# ****Report on Water Jug Problem using Stack (DFS)****

# ****Introduction****

The **Water Jug Problem** is a classic **state-space search problem** where we need to measure a specific amount of water using two jugs with predefined capacities. This report details the implementation of the **(DFS) algorithm using a stack** to solve the problem.

The objective is to determine if it's possible to measure the desired amount of water using the given jugs and operations:

Fill a jug completely.

Empty a jug completely.

Transfer water from one jug to another.

## ****1. Importing Required Libraries****

from collections import deque

* The **deque** module from collections is imported, but it's not actually used in the stack-based implementation.

## ****2. Implementing the Water Jug Problem using Stack (DFS Approach)****

### 

### ****Key Points:****

1. **Stack-based DFS approach** is used to explore all possible states.
2. **Set visited ensures no duplicate states** are revisited, preventing infinite loops.
3. **The possible moves list contains**:
   * Filling either jug completely.
   * Emptying either jug.
   * Pouring from one jug to another while respecting capacities.
4. If a move leads to an **unvisited state**, it is added to the **stack** and processed further.
5. If the **goal amount** is found in either jug, the function prints "Solution found" and returns True.

## ****3. Running the Algorithm****

jug1\_capacity = 4

jug2\_capacity = 3

goal = 2

waterjugproblem(jug1\_capacity, jug2\_capacity, goal)

### ****Explanation of Execution:****

* The function starts with **both jugs empty (0,0)**.
* Using DFS, it explores different ways of **filling, emptying, and transferring water** between jugs.
* The process continues **until the goal amount is found or all possibilities are exhausted**.
* If a solution exists, it is displayed. Otherwise, "Nothing found" is printed.

**4. Example Execution Trace**

### ****Step-by-step Execution:****

### 

* **Start with (0,0)** → Fill Jug1 (4,0)
* **Fill Jug2 (4,3)**
* **Pour water from Jug1 to Jug2 until it’s full → (1,3)**
* **Empty Jug2 → (1,0)**
* **Pour from Jug1 to Jug2 → (0,1)**
* **Fill Jug1 again → (4,1)**
* **Pour from Jug1 to Jug2 → (2,3) (Solution found!)**

## ****5. Complexity Analysis****

### ****Time Complexity****

* **Worst case: O(N!)**, where N is the number of possible states.
* Since the search is **depth-first**, it explores a path fully before backtracking.
* If the goal is found early, execution stops.

### ****Space Complexity****

* **O(N)** where N is the number of visited states.
* The stack can **grow linearly** in worst-case scenarios.

## ****7. Conclusion****

* **This implementation successfully solves the Water Jug Problem using a stack-based DFS approach.**
* The algorithm efficiently explores possible states and avoids infinite loops using a visited set.
* The method **does not always guarantee the shortest path** but finds a solution if one exists.
* The approach is **computationally efficient** and works well for small-scale problems.