PAI LAB
BS in Artificial Intelligence

Department of Software Engineering

Faculty of Computer Science & Information Technology

The Superior University, Lahore

Sr.#	Reg.#	Student Name
1.	SU92-BSAIM-F23-107	Ayesha Faisal

Table of Contents

Table of Figures

Water Jug Problem:

1 INTRODUCTION:

Solves the classic Water Jug Problem using Depth-First Search (DFS).

Takes as input:

- Two jug capacities (capacity1 and capacity2).
- A target volume of water (target) to measure.
- Explores all possible states (amounts of water in each jug) by applying a set of rules (fill, empty, pour).
- Keeps track of visited states to avoid repeating and infinite loops.
- Records the sequence of states and the actions/rules applied to reach the target.

2 CODE IMPLEMENTATION:

```python
def water_jug_dfs(capacity1, capacity2, target):
    visited = set()
    path = []
    actions_taken = []

    rule_descriptions = {
        1: f"Fill {capacity1}-liter jug",
        2: f"Fill {capacity2}-liter jug",
        3: f"Empty {capacity1}-liter jug",
        4: f"Empty {capacity2}-liter jug",
        5: f"Pour {capacity1}-liter jug into {capacity2}-liter jug until {capacity2}-liter jug is full",
        6: f"Pour {capacity2}-liter jug into {capacity1}-liter jug until {capacity1}-liter jug is full",
        7: f"Pour {capacity1}-liter jug into {capacity2}-liter jug until {capacity1}-liter jug is empty",
        8: f"Pour {capacity2}-liter jug into {capacity1}-liter jug until {capacity2}-liter jug is empty",
    }

    def dfs(jug1, jug2):
        if (jug1, jug2) in visited:
            return False

        visited.add((jug1, jug2))
        path.append((jug1, jug2))
        if jug1 == target or jug2 == target:
            return True
```

Figure a code-1

```
        rules = [
            (capacity1, jug2, 1),
            (jug1, capacity2, 2),
            (0, jug2, 3),
            (jug1, 0, 4),
            (capacity1, abs(jug2 - (capacity1 - jug1)), 5),
            (abs(jug1 - (capacity2 - jug2)), capacity2, 6),
            (jug1 + jug2, 0, 7),
            (0, jug1 + jug2, 8),
        ]

        for new_jug1, new_jug2, rule_num in rules:
            if new_jug1 > capacity1 or new_jug2 > capacity2:
                continue
            if (new_jug1, new_jug2) not in visited:
                actions_taken.append(rule_descriptions.get(rule_num, "Unknown Action"))
                if dfs(new_jug1, new_jug2):
                    return True
                actions_taken.pop()

        path.pop()
        return False

    dfs(0, 0)

    if path and (path[-1][0] == target or path[-1][1] == target):
        return path,actions_taken
    else:
        return None, None
```

Figure b code-2

```
capacity1 = 6
capacity2 = 5
target = 4
solution, actions = water_jug_dfs(capacity1, capacity2, target)

if solution:
    print("Solution steps:")
    for i, step in enumerate(solution):
        action = actions[i - 1] if i > 0 else "Start"
        print(f"Step {i}: {step} - Action: {action}")
else:
    print("No solution found.")
```

✓ 0.0s

Figure c code-3

3 OUTPUT:

```
Solution steps:
Step 0: (0, 0) - Action: Start
Step 1: (6, 0) - Action: Fill 6-liter jug
Step 2: (6, 5) - Action: Fill 5-liter jug
Step 3: (0, 5) - Action: Empty 6-liter jug
Step 4: (6, 1) - Action: Pour 6-liter jug into 5-liter jug until 5-liter jug is full
Step 5: (0, 1) - Action: Empty 6-liter jug
Step 6: (4, 5) - Action: Pour 5-liter jug into 6-liter jug until 6-liter jug is full
```

Figure d OUTPUT