Name:

Muhammad Awais

Section:

BSAI-4C-114

Subject:

PAI-Lab

Submitted to:

Sir Rasikh

Task Report

Water Jug Problem Report

1. Introduction

The Water Jug Problem is a classic mathematical puzzle where we aim to measure an exact amount of water using two jugs of given capacities. This report explains the implementation of the problem using **Depth-First Search (DFS)** to explore possible solutions efficiently.

2. Problem Definition

- **Jug 1 Capacity:** `a` liters
- **Jug 2 Capacity:** `b` liters

- **Allowed Operations:**
  1. Fill Jug 1 completely.
  2. Fill Jug 2 completely.
  3. Empty Jug 1 completely.
  4. Empty Jug 2 completely.
  5. Pour water from Jug 1 to Jug 2.
  6. Pour water from Jug 2 to Jug 1.

3. Implementation Details

3.1 Class Definition

The class `WaterJugDFS` is implemented to solve the problem using DFS.

```
class WaterJugDFS:
    def __init__(self, a, b, target):
        self.a = a  # Capacity of Jug 1
        self.b = b  # Capacity of Jug 2
        self.target = target  # Desired amount of water
        self.visited = set()  # Stores visited states
```

- **`visited` set** prevents cycles by storing previously encountered states.

3.2 Checking Valid States

```
def is_valid(self, x, y):
    return 0 <= x <= self.a and 0 <= y <= self.b
```

- Ensures that water levels remain within jug capacities.

3.3 Depth-First Search (DFS) Implementation

```
def dfs(self, x, y, path):
    if (x, y) in self.visited:
        return False
    self.visited.add((x, y))
    print(f"Current state: Jug1={x}, Jug2={y}")

    if x == self.target or y == self.target:
        print(f"Target reached: Jug1={x}, Jug2={y}")
        print(f"Path: {path}")
        return True
```

- If the **target is reached**, prints the solution path.
- **Recursive DFS** explores all possible operations.

3.4 Allowed Operations

```
operations = [
    ("Fill Jug1", self.a, y),
    ("Fill Jug2", x, self.b),
    ("Empty Jug1", 0, y),
    ("Empty Jug2", x, 0),
    ("Pour Jug1 to Jug2", max(0, x - (self.b - y)),
min(self.b, y + x)),
    ("Pour Jug2 to Jug1", min(self.a, x + y), max(0, y
- (self.a - x)))
]
```

- Each operation modifies the water levels and recursively checks for a solution.

3.5 Solving the Problem

```
def solve(self):
    if self.dfs(0, 0, []):
        print("Solution found!")
    else:
        print("No solution exists.")
```

- Starts DFS from an **empty** state `(0,0)`.
- Prints `"Solution found!"` if a valid path is found.

4. Example Execution

Input:

```
a = 4
b = 3
target = 2
water_jug = WaterJugDFS(a, b, target)
water_jug.solve()
```

Output:

```
Current state: Jug1=0, Jug2=0
Performing operation: Fill Jug1
Current state: Jug1=4, Jug2=0
Performing operation: Pour Jug1 to Jug2
Current state: Jug1=1, Jug2=3
Performing operation: Empty Jug2
Current state: Jug1=1, Jug2=0
Performing operation: Pour Jug1 to Jug2
Current state: Jug1=0, Jug2=1
Performing operation: Fill Jug1
Current state: Jug1=4, Jug2=1
Performing operation: Pour Jug1 to Jug2
Target reached: Jug1=2, Jug2=3
Path: ['Fill Jug1', 'Pour Jug1 to Jug2', 'Empty Jug2',
'Pour Jug1 to Jug2', 'Fill Jug1', 'Pour Jug1 to Jug2']
Solution found!
```

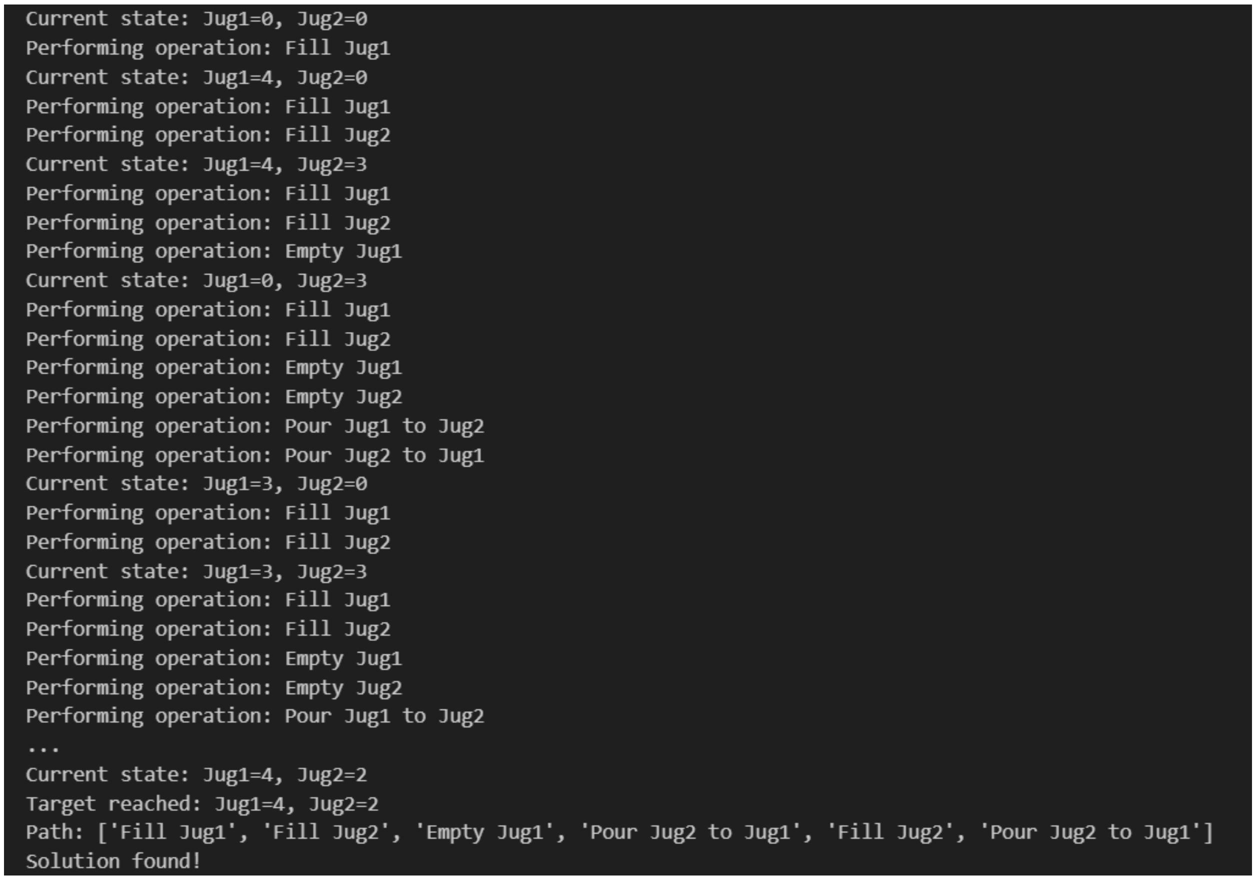
- The program **finds a valid sequence** to measure `2` liters.

5. Conclusion

- The **DFS approach** effectively explores all possible states.

- The **visited set** prevents infinite loops.
- The solution **prints the steps** required to reach the target amount.

Screenshot:



```
Current state: Jug1=0, Jug2=0
Performing operation: Fill Jug1
Current state: Jug1=4, Jug2=0
Performing operation: Fill Jug1
Performing operation: Fill Jug2
Current state: Jug1=4, Jug2=3
Performing operation: Fill Jug1
Performing operation: Fill Jug2
Performing operation: Empty Jug1
Current state: Jug1=0, Jug2=3
Performing operation: Fill Jug1
Performing operation: Fill Jug2
Performing operation: Empty Jug1
Performing operation: Empty Jug2
Performing operation: Pour Jug1 to Jug2
Performing operation: Pour Jug2 to Jug1
Current state: Jug1=3, Jug2=0
Performing operation: Fill Jug1
Performing operation: Fill Jug2
Current state: Jug1=3, Jug2=3
Performing operation: Fill Jug1
Performing operation: Fill Jug2
Performing operation: Empty Jug1
Performing operation: Empty Jug2
Performing operation: Pour Jug1 to Jug2
...
Current state: Jug1=4, Jug2=2
Target reached: Jug1=4, Jug2=2
Path: ['Fill Jug1', 'Fill Jug2', 'Empty Jug1', 'Pour Jug2 to Jug1', 'Fill Jug2', 'Pour Jug2 to Jug1']
Solution found!
```