# WATER JUG PROBLEM

## Problem Statement

- The Water Jug Problem, also known as the **Die Hard** Problem, is a classic puzzle in mathematics and computer science.
- The problem involves two jugs of different capacities and the objective of measuring a specific quantity of water using only these jugs and an unlimited water supply
- Given two jugs with capacities m and n liters, where m and n are positive integers, and the task is to measure out a desired quantity of water, d liters, where d is a positive integer less than or equal to the capacity of the larger jug.
- The challenge is to determine if it's possible to measure d liters using the given jugs and find a sequence of actions that accomplishes this task.

# EXAMPLE

**You have a 2-liter jug and a 3-liter jug. Can you measure exactly 1 liters of water into 2-liter of jug**

**01**

Fill the 3-liter jug full

**02**

Pour the water from the 3-liter jug into the 2-liter jug

**03**

Empty the 2-liter jug

**04**

Pour the remaining 1 liter from the 3-liter jug into the 2-liter jug

# Approaches

01.

**Brute Force Method**

This approach involves systematically trying every possible combination of actions until a solution is found. It may involve creating a tree-like structure to represent all possible states of the jugs and then searching through this tree for a solution

02.

**Breadth-First Search**

BFS is a graph traversal algorithm that can be applied to solve the water jug problem. In this approach, you start with an initial state representing the empty jugs and explore all possible actions from that state. Then, you move to the next level of states and continue exploring until you find a solution.

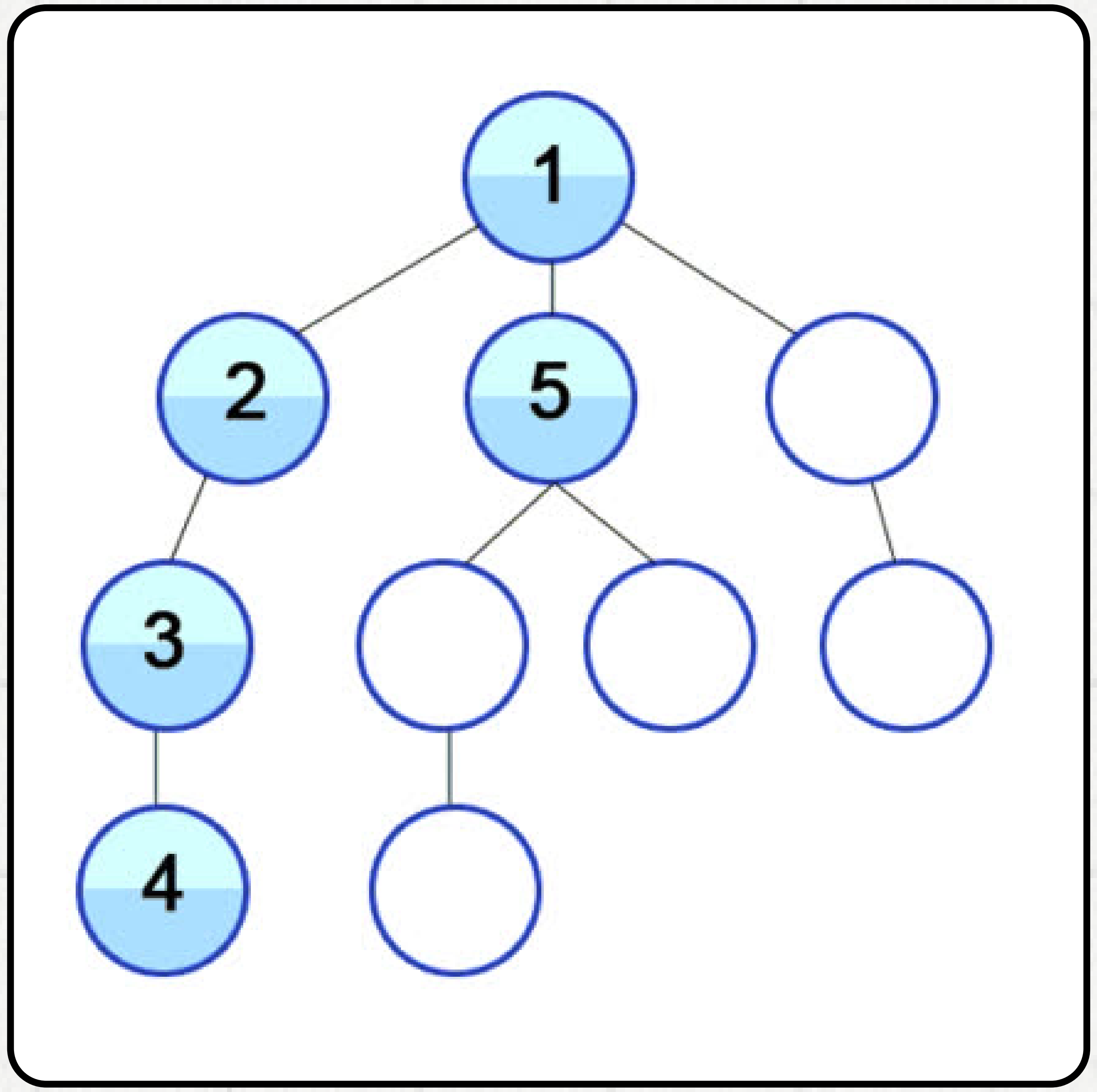
03.

**Depth-First Search**

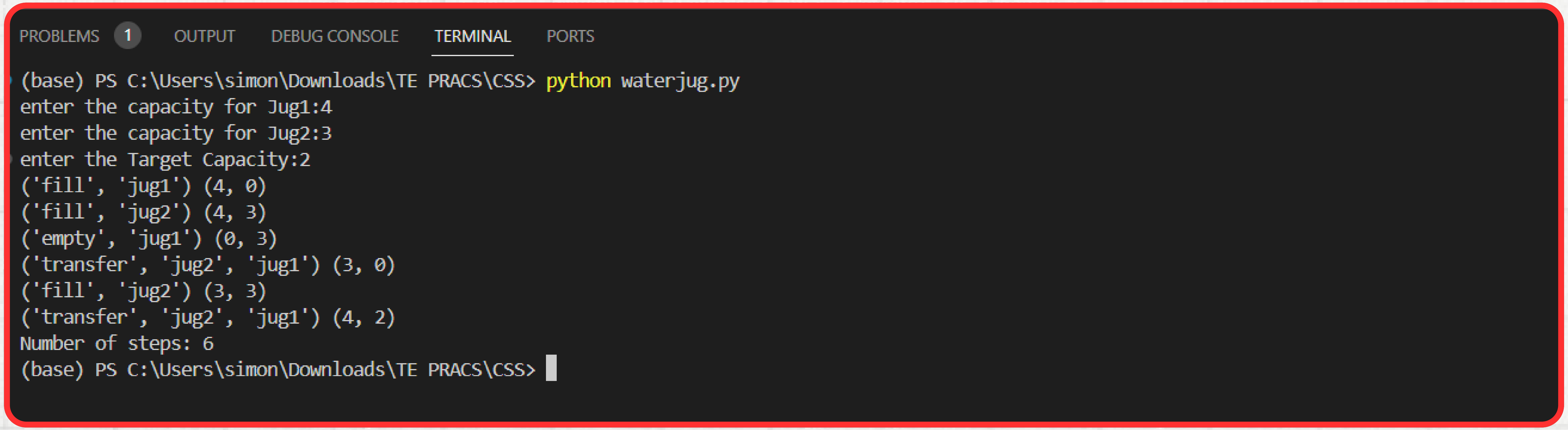
DFS is another graph traversal algorithm that can be used to solve the water jug problem. In this approach, you start with an initial state and explore as far as possible along each branch before backtracking.

# Depth-First Search

1. **Define Initial State**: Start with an initial state representing the current state as empty.
2. **Explore Possible Actions**:  In the water jug problem, these actions typically include filling a jug, emptying a jug, or transferring water from one jug to another.
3. **Recursively Explore States**: For each action, recursively explore the resulting state.
4. **Terminate or Backtrack**: If a goal state is reached, terminate the recursion and return the solution. If a dead-end is reached, backtrack to the previous state and explore a different action.
5. **Repeat**: Repeat steps 2-4 until a solution is found or until all possible states have been explored.
6. **Track Visited States**: To avoid revisiting the same state, keep track of visited states using a data structure like a set.
7. **Return Solution**: Once a solution is found, return the sequence of actions that lead to the solution, along with the final state of the jugs.



## Results



```
(base) PS C:\Users\simon\Downloads\TE PRACS\CSS> python waterjug.py
enter the capacity for Jug1:4
enter the capacity for Jug2:3
enter the Target Capacity:2
('fill', 'jug1') (4, 0)
('fill', 'jug2') (4, 3)
('empty', 'jug1') (0, 3)
('transfer', 'jug2', 'jug1') (3, 0)
('fill', 'jug2') (3, 3)
('transfer', 'jug2', 'jug1') (4, 2)
Number of steps: 6
(base) PS C:\Users\simon\Downloads\TE PRACS\CSS>
```

Implementation Link

# Advantages & Disadvantages

| | Advantages | Disadvantages |
|---|---|---|
| **DFS** | **Memory Efficiency**<br>• DFS is more memory-efficient compared to BFS in the water jug problem, exploring a single path at a time. | **Completeness and Optimality Not Guaranteed**<br>DFS does not ensure completeness or optimality in finding the shortest path to the goal state; it may encounter infinite loops or explore non-optimal solutions. |
| **BFS** | **Completeness and Optimality**<br>BFS guarantees finding the shortest path to the goal state in the water jug problem | **Memory Usage**<br>• BFS may consume a significant amount of memory, especially for problems with extensive search spaces. |

# THANK YOU