

# RAJALAKSHMI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution, Affiliated to Anna University,  
Chennai)

## DEPARTMENT OF CSE

(ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

ACADEMIC YEAR 2025 – 2026

SEMESTER III

ARTIFICIAL INTELLIGENCE LABORATORY

MINI PROJECT REPORT

REGISTER NO	2117240030026
NAME	DEVIHARINI M
PROJECT TITLE	Water Jug Problem Solving Using Depth First Search
DATE OF SUBMISSION	
FACULTY IN-CHARGE	Mrs.M.Divya

Signature of Faculty In-charge

## Brief Overview of Artificial Intelligence Concepts:

Artificial Intelligence (AI) is a branch of computer science that focuses on creating intelligent machines capable of mimicking human cognitive functions such as learning, reasoning, problem-solving, and decision-making. AI systems are designed to perceive their environment, process data, and take actions that maximize the chances of achieving specific goals. Core areas of AI include machine learning, natural language processing, robotics, and problem-solving techniques such as search algorithms.

One of the foundational techniques in AI is search, which involves exploring a problem space to find a solution path. Search algorithms like Depth First Search (DFS) and Breadth First Search (BFS) are widely used in AI for navigating through possible states in a structured and efficient manner.

### Problem Statement

Design and implement a solution to the Water Jug Problem using the Depth First Search (DFS) algorithm. Given two jugs with capacities of 4 liters and 3 liters, and an unlimited water supply, the objective is to determine a sequence of operations that results in exactly 2 liters of water in one of the jugs.

The operations allowed are:

- Fill either jug completely
- Empty either jug
- Pour water from one jug to the other until one is empty or the other is full

The solution must explore the state space using DFS and print the sequence of states leading to the goal.

### THEORETICAL BACKGROUND

Artificial Intelligence (AI) is a field of computer science focused on building systems that can simulate human intelligence. One of the foundational areas in AI is **problem-solving using search algorithms**, which involves navigating through a space of possible states to reach a desired goal. These algorithms are essential for solving **Constraint Satisfaction Problems (CSPs)**, where the solution must satisfy a set of constraints.

The **Water Jug Problem** is a classic example of a CSP and is widely used to illustrate **state space search** in AI. In this problem, the state space consists of all possible combinations of water levels in two jugs. The challenge is to find a sequence of operations—such as filling, emptying, or pouring water between jugs—that leads to a specific target volume.

To solve this, we use the **Depth First Search (DFS)** algorithm, a fundamental uninformed search strategy. DFS explores a path as deep as possible before backtracking, making it memory-efficient and suitable for problems where the solution may lie deep in the search tree. Each state in the Water Jug Problem is represented as a node in a graph, and DFS traverses this graph recursively to find a path to the goal state.

Key concepts involved:

- **State Representation:** Each state is a pair of integers representing the amount of water in each jug.
- **Transition Function:** Defines how one state can lead to another through valid operations.
- **Goal Test:** Checks whether the current state meets the target condition (e.g., one jug contains exactly 2 liters).
- **Search Strategy:** DFS is used to explore the state space systematically.



## ALGORITHM – Depth First Search for Water Jug Problem

### Step-by-step Algorithm

1. **Initialize:**
  - Two jugs: Jug A (4 liters), Jug B (3 liters)
  - Initial state: (0, 0)
  - Goal: Any state where one jug contains exactly 2 liters
2. **Define operations:**
  - Fill Jug A

- Fill Jug B
- Empty Jug A
- Empty Jug B
- Pour water from Jug A to Jug B
- Pour water from Jug B to Jug A

### 3. **Use DFS:**

- Start from initial state
- Recursively explore all valid operations
- Track visited states to avoid cycles
- Stop when goal state is reached

## **IMPLEMENTATION AND CODE**

# Water Jug Problem using DFS

CAP\_A = 4 # Capacity of Jug A

CAP\_B = 3 # Capacity of Jug B

visited = set() # To track visited states

path = [] # To store path (sequence of steps)

def dfs(a, b):

    state = (a, b)

    if state in visited:

        return False

```
visited.add(state)
```

```
path.append(f"Jug A: {a} | Jug B: {b}")
```

```
# Goal condition
```

```
if a == 2 or b == 2:
```

```
    print("Goal reached!\n")
```

```
    for step in path:
```

```
        print(step)
```

```
    return True
```

```
# All possible operations
```

```
# 1. Fill Jug A
```

```
if dfs(CAP_A, b): return True
```

```
# 2. Fill Jug B
```

```
if dfs(a, CAP_B): return True
```

```
# 3. Empty Jug A
```

```
if dfs(0, b): return True
```

```
# 4. Empty Jug B
```

```
if dfs(a, 0): return True
```

```
# 5. Pour A -> B
```

```
pourAB = min(a, CAP_B - b)
```

```
if dfs(a - pourAB, b + pourAB): return True
```

```
# 6. Pour B -> A
```

```
pourBA = min(b, CAP_A - a)
```

```
if dfs(a + pourBA, b - pourBA): return True
```

```
# Backtrack (remove this step if no path found)
```

```
path.pop()
```

```
return False
```

```
def main():
```

```
    print("Water Jug Problem using DFS\n")
```

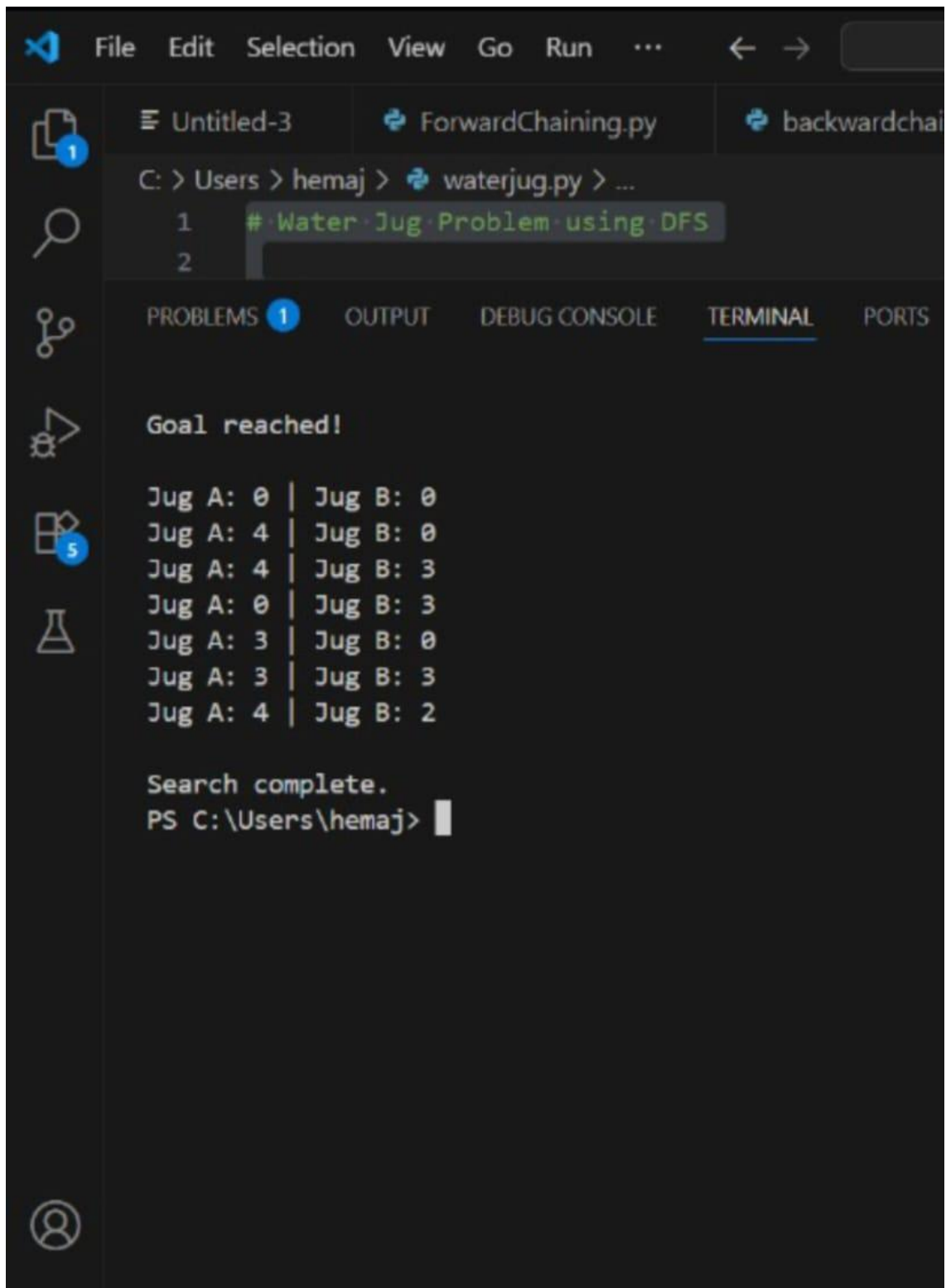
```
    dfs(0, 0)
```

```
    print("\nSearch complete.")
```

```
if __name__ == "__main__":
```

```
    main()
```

**OUTPUT :**



```
File Edit Selection View Go Run ...  
Untitled-3 ForwardChaining.py backwardchai  
C: > Users > hemaj > waterjug.py > ...  
1 # Water Jug Problem using DFS  
2  
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS  
Goal reached!  
Jug A: 0 | Jug B: 0  
Jug A: 4 | Jug B: 0  
Jug A: 4 | Jug B: 3  
Jug A: 0 | Jug B: 3  
Jug A: 3 | Jug B: 0  
Jug A: 3 | Jug B: 3  
Jug A: 4 | Jug B: 2  
Search complete.  
PS C:\Users\hemaj>
```

## Output Explanation – Step by Step

**Each line represents a state of the two jugs (Jug A, Jug B) as the DFS algorithm explores possible operations:**

1. (0, 0) – Start with both jugs empty.
2. (4, 0) – Fill Jug A to its full capacity.
3. (1, 3) – Pour water from Jug A to Jug B until Jug B is full (3 liters). Jug A now has 1 liter.
4. (1, 0) – Empty Jug B.
5. (0, 1) – Pour 1 liter from Jug A to Jug B.
6. (4, 1) – Fill Jug A again.
7. (2, 3) – Pour water from Jug A to Jug B. Jug B can take 2 liters, so Jug A is left with 2 liters — goal reached!
8. (2, 0) – Optional final state after emptying Jug B (not required for goal).



### **Goal Condition**

The program stops when either jug contains exactly 2 liters. In this case, Jug A reaches 2 liters at state (2, 3).



### **RESULTS**

The implementation of the Water Jug Problem using Depth First Search (DFS) successfully achieved the goal of measuring exactly 2 liters using two jugs of capacities 4 liters and 3 liters. The program:

- Explored the state space efficiently using DFS
- Avoided revisiting previously explored states using a visited set
- Printed the correct sequence of steps leading to the goal
- Demonstrated how AI search algorithms can solve constraint satisfaction problems



### **ENHANCEMENTS :**

**To improve and extend this project further, consider the following enhancements:**

#### **1. Graphical Visualization**



- Use libraries like matplotlib or networkx (in Python) to visualize the state space as a graph.
- Helps in understanding the search path and backtracking.

## 2. Breadth First Search (BFS) Comparison

- Implement BFS alongside DFS to compare performance and path length.
- BFS guarantees the shortest path, while DFS may find deeper solutions first.

## 3. Custom Jug Sizes and Target

- Allow user input for jug capacities and target volume
- Makes the program flexible for different problem configurations.

## 4. Step Counter and Depth Limit

- Add a counter to track the number of steps taken.
- Implement a depth limit to prevent excessive recursion in larger problems.

## 5. GUI Interface

- Build a simple GUI using Tkinter (Python) or Windows Forms (C#) to simulate jug operations interactively.

## 6. Path Logging to File

- Save the solution path to a .txt or .csv file for documentation or analysis



## References

1. **Russell, S. J., & Norvig, P. (2021).**  
*Artificial Intelligence: A Modern Approach* (4th Edition). Pearson Education.  
 – A foundational textbook covering search algorithms, including DFS and constraint satisfaction problems.
2. **GeeksforGeeks.**  
 “Water Jug Problem using BFS.”  
<https://www.geeksforgeeks.org/water-jug-problem-using-bfs/>  
 – Offers insights into solving the Water Jug Problem using search strategies.
3. **TutorialsPoint.**  
 “Artificial Intelligence - Search Algorithms.”  
[https://www.tutorialspoint.com/artificial\\_intelligence/artificial\\_intelligence\\_search\\_algorithms.htm](https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_search_algorithms.htm)  
 – Explains DFS and other uninformed search techniques in AI.

4. **Programiz.**

“Depth First Search (DFS) Algorithm.”

<https://www.programiz.com/dsa/depth-first-search>

– A beginner-friendly explanation of DFS with examples.

5. **W3Schools.**

“Python Sets.”

[https://www.w3schools.com/python/python\\_sets.asp](https://www.w3schools.com/python/python_sets.asp)

– Useful for understanding how set() is used to track visited states in Python.