BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India

DEPARTMENT OF COMPUTER ENGINEERING
SUBJECT: Artificial Intelligence and Machine Learning

Name	Manish Shashikant Jadhav
UID no.	2023301005

Experiment 2

AIM :

Implement a given problem using the Uninformed (DFS) searching technique. Analyze the algorithms with respect to Completeness, Optimality, time and space Complexity.

- A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in a 4-gallon jug?

Production Rules for Water Jug Problem in Artificial Intelligence:

1	(x, y) is X<4 ->(4, Y)	Fill the 4-liter jug
2	(x, y) if Y<3 -> (x, 3)	Fill the 3-liter jug
3	(x, y) if x>0 -> (x-d, d)	Pour some water out of the 4-liter jug.
4	(x, y) if Y>0 -> (d, y-d)	Pour some water out of the 3-liter jug.
5	(x, y) if x>0 -> (0, y)	Empty the 4-liter jug on the ground
6	(x, y) if y>0 -> (x,0)	Empty the 3-liter jug on the ground
7	(x, y) if X+Y >= 4 and y>0 -> (4, y-(4-x))	Pour water from the 3-liter jug into the 4-liter jug until the 4-liter jug is full
8	(x, y) if X+Y>=3 and x>0 -> (x-(3-y), 3))	Pour water from the 4-liter jug into the 3-liter jug until the 3-liter jug is full.
9	(x, y) if X+Y <=4 and y>0 -> (x+y, 0)	Pour all the water from the 3-liter jug into the 4-liter jug.
10	(x, y) if X+Y<=3 and x>0 -> (0, x+y)	Pour all the water from the 4-liter jug into the 3-liter jug.
11	(0, 2) -> (2, 0)	Pour the 2-liter water from the 3-liter jug into the 4-liter jug.
12	(2, Y) -> (0, y)	Empty the 2-liter in the 4-liter jug on the ground.

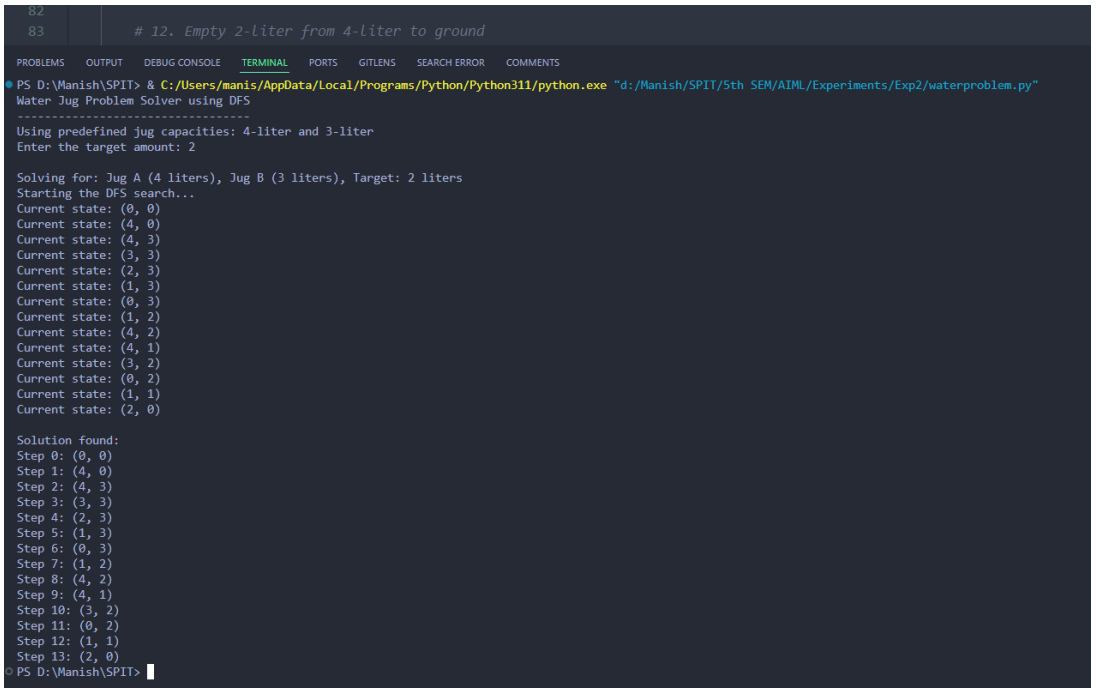
CODE:

```
def water_jug_dfs(capacity_a, capacity_b, target):
    def dfs(x, y, path):
        if (x, y) in visited:
            return None
```

```
    visited.add((x, y))
    path = path + [(x, y)]

    print(f"Current state: ({x}, {y})")

    if x == target and y == 0:
        return path

    # Apply production rules
    # 1. Fill the 4-liter jug
    if x < 4:
        result = dfs(4, y, path)
        if result:
            return result

    # 2. Fill the 3-liter jug
    if y < 3:
        result = dfs(x, 3, path)
        if result:
            return result

    # 3. Pour some water out of the 4-liter jug
    if x > 0:
        for d in range(1, x + 1):
            result = dfs(x - d, y + d if y + d <= 3 else 3, path)
            if result:
                return result

    # 4. Pour some water out of the 3-liter jug
    if y > 0:
        for d in range(1, y + 1):
            result = dfs(x + d if x + d <= 4 else 4, y - d, path)
            if result:
                return result

    # 5. Empty the 4-liter jug on the ground
    if x > 0:
        result = dfs(0, y, path)
        if result:
```

```
        return result

    # 6. Empty the 3-liter jug on the ground
    if y > 0:
        result = dfs(x, 0, path)
        if result:
            return result

    # 7. Pour from 3-liter to 4-liter until 4-liter is full
    if x + y >= 4 and y > 0:
        result = dfs(4, y - (4 - x), path)
        if result:
            return result

    # 8. Pour from 4-liter to 3-liter until 3-liter is full
    if x + y >= 3 and x > 0:
        result = dfs(x - (3 - y), 3, path)
        if result:
            return result

    # 9. Pour all from 3-liter to 4-liter
    if x + y <= 4 and y > 0:
        result = dfs(x + y, 0, path)
        if result:
            return result

    # 10. Pour all from 4-liter to 3-liter
    if x + y <= 3 and x > 0:
        result = dfs(0, x + y, path)
        if result:
            return result

    # 11. Pour 2-liter from 3-liter to 4-liter
    if (x, y) == (0, 2):
        result = dfs(2, 0, path)
        if result:
            return result

    # 12. Empty 2-liter from 4-liter to ground
```

```
        if x == 2:
            result = dfs(0, y, path)
            if result:
                return result

        return None

    visited = set()
    return dfs(0, 0, [])

def get_positive_int_input(prompt):
    while True:
        try:
            value = int(input(prompt))
            if value > 0:
                return value
            else:
                print("Please enter a positive integer.")
        except ValueError:
            print("Invalid input. Please enter a positive integer.")

def main():
    print("Water Jug Problem Solver using DFS")
    print("----------------------------------")

    print("Using predefined jug capacities: 4-liter and 3-liter")
    target = get_positive_int_input("Enter the target amount: ")

    print(f"\nSolving for: Jug A (4 liters), Jug B (3 liters), Target: {target} liters")
    print("Starting the DFS search...")

    solution = water_jug_dfs(4, 3, target)

    if solution:
        print("\nSolution found:")
        for step, (a, b) in enumerate(solution):
            print(f"Step {step}: ({a}, {b})")
    else:
        print("No solution found.")
```

	if __name__ == "__main__": main()
OUTPUT:	

Analysis of Algorithm	**1. Completeness:**
	• Complete: Will find a solution if one exists.
	• Reason: Explores all states in the finite state space (20 possible states).
	2. Optimality:
	• Not optimal: Doesn't guarantee the shortest solution.
	• Reason: DFS explores deep paths first, potentially finding longer solutions before shorter ones.
	3. Time Complexity:
	• Worst-case: O(b^d)
	  ○ b: branching factor (max 12 production rules)
	  ○ d: maximum depth (can exceed 20 due to potential state revisits)
	• Practically better due to visited set and early termination.
	4. Space Complexity:
	• O(d)
	  ○ d: maximum depth of the search
	• Includes:
	  ○ Recursive call stack
	  ○ Visited set (max 20 states).

CONCLUSION:	Hence by completing this experiment I came to know about Implementation of a problem using the Uninformed (DFS) searching technique.