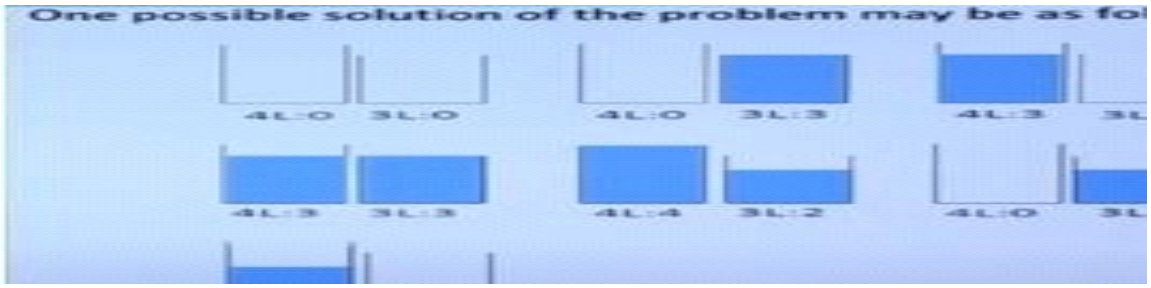
EX.NO: 3

DATE:

DEPTH FIRST SEARCH – WATER JUG PROBLEM

In the **water jug problem in Artificial Intelligence**, we are provided with two jugs: one having the capacity to hold 3 gallons of water and the other has the capacity to hold 4 gallons of water. There is no other measuring equipment available and the jugs also do not have any kind of marking on them. So, the agent's task here is to fill the 4-gallon jug with 2 gallons of water by using only these two jugs and no other material. Initially, both our jugs are empty.



AIM :

To implement a python program for Water Jug problem using depth first search problem

CODE :

+ Code + Text

```
# Experiment 3 - water jug
def is_valid_state(x, y):
    # Ensure that the state is within the capacity of the jugs
    return 0 <= x <= 4 and 0 <= y <= 3

def dfs(x, y, visited, path):
    if (x, y) in visited:
        return False

    # Mark the current state as visited
    visited.add((x, y))

    # Store the current state in the path
    path.append((x, y))

    # If we have 2 liters in the 4-liter jug and 0 liters in the 3-liter jug, the problem is solved
    if x == 2 and y == 0:
        return True

    # Possible moves
    possible_moves = [
        (4, y),  # Fill the 4-liter jug
        (x, 3),  # Fill the 3-liter jug
        (0, y),  # Empty the 4-liter jug
        (x, 0),  # Empty the 3-liter jug
        (x - min(x, 3 - y), y + min(x, 3 - y)),  # Pour from 4L to 3L
        (x + min(y, 4 - x), y - min(y, 4 - x))   # Pour from 3L to 4L
    ]

    # Explore all possible moves using DFS
    for (next_x, next_y) in possible_moves:
        if is_valid_state(next_x, next_y) and dfs(next_x, next_y, visited, path):
            return True
```

```
+ Code + Text
        for (next_x, next_y) in possible_moves:
            if is_valid_state(next_x, next_y) and dfs(next_x, next_y, visited, path):
                return True

        # Backtrack: remove the last state if it leads to no solution
        path.pop()
        return False

    def solve_water_jug_problem():
        # Initialize the starting state
        initial_state = (0, 0)

        # To keep track of visited states
        visited = set()

        # To store the path to the solution
        path = []

        if dfs(initial_state[0], initial_state[1], visited, path):
            print("Solution found!")
            for step in path:
                print(step)
        else:
            print("No solution exists.")

    # Main code
    if __name__ == "__main__":
        solve_water_jug_problem()
```

OUTPUT :

```
Solution found!
(0, 0)
(4, 0)
(4, 3)
(0, 3)
(3, 0)
(3, 3)
(4, 2)
(0, 2)
(2, 0)
```

RESULT :

Thus the output is succesfully executed and output is verified .