EXP NO: 3

DATE:

DEPTH FIRST SEARCH - WATER JUG PROBLEM

AIM:

To implement depth first search for water jug problem using python.

ALGORITHM:

1) Initialize visited states as a 2D list to track jug1 and jug2 combinations

2) Start DFS from the state where both jugs are empty (0, 0).

3) In DFS:

-> If either jug1 or jug2 equals the target, return True

-> If the current state has already been visited, return False

-> Mark the current state as visited.

-> Explore possible operations:

* Fill jug1 or jug2.

* Empty jug1 or jug2

* Pour water from one jug to the other

4) Recursively explore all valid moves

5) If the target is reached, record the steps and return them

6) If no solution is found, return an empty list.

Program:

```
def water_jug_dfs(jug1, jug2, target):
    def dfs(jug1, jug2):
        if jug1 == target or jug2 == target:
            return True
        if visited[jug1][jug2]:
            return False
        visited[jug1][jug2] = True
        if jug1 < A:
            if dfs(A, jug2):
                steps.append((A, jug2))
                return True
        if jug2 < B:
            if dfs(jug1, B):
                steps.append((jug1, B))
                return True
```

```
        if jug1 > 0:
            if dfs(0, jug2):
                steps.append((0, jug2))
                return True
        if jug2 > 0:
            if dfs(jug1, 0):
                steps.append((jug1, 0))
                return True
        if jug1 > 0 and jug2 < B:
            to_pour = min(jug1, B - jug2)
            if dfs(jug1 - to_pour, jug2 + to_pour):
                steps.append((jug1 - to_pour, jug2 + to_pour))
                return True
        if jug2 > 0 and jug1 < A:
            to_pour = min(jug2, A - jug1)
            if dfs(jug1 + to_pour, jug2 - to_pour):
                steps.append((jug1 + to_pour, jug2 - to_pour))
                return True
        return False
    A, B = jug1, jug2
    visited = [[False] * (B + 1) for _ in range(A + 1)]
    steps = []
    if dfs(0, 0):
        steps.append((target, 0))
```

```
        return steps
    else:
        return []
jug1_capacity = 4
jug2_capacity = 3
target_capacity = 2
result = water_jug_dfs(jug1_capacity, jug2_capacity,
                       target_capacity)
if result:
    for step in result:
        print(f'Jug1: {step[0]}, Jug2: {step[1]}')
else:
    print("No solution found")
```

Output :

Jug1 : 4 , Jug2 : 2

Jug1 : 3 , Jug2 : 3

Jug1 : 3 , Jug2 : 0

Jug1 : 0 , Jug2 : 3

Jug1 : 4 , Jug2 : 3

Jug1 : 4 , Jug2 : 0

Jug1 : 2 , Jug2 : 0

RESULT :

Thus a python program has been implemented to solve the water jug problem.