DATE:

EX: 3

DFS (WATER JUG PROBLEM)

Aim:

To implement water jug problem using the depth-first search (DFS) algorithm.

Algorithm:

1) State representation: to represent the state of jugs as a tuple (x, y) where x is the amount of water in jug.

2) Start with both jugs empty.

3) Define the possible operation.

→ If either jug1 or jug2 equals the target, return True

→ If the current state as visited.

→ Explore possible operation:

* 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

def

JUG1 PROBLEM)

ug problem
DFS)

to represent
,y) where
Jug.
empty.
peration
jug2 equals

e as

peration:
g2
or jug2
from one jug
er
valid moves.
, record

d, return

Program:

```
def water_jug_dfs(jug1, jug2, target):
    def dfs(jug1, jug2):
        if jug1 == jug2 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, jug
                -to-pour):

                step.append(
                jug_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.
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

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 : 1 0 , jug2 : 3
jug1 : 4 , jug2 : 3
jug1 : 4 , jug2 : 0
jug1 : 2 , jug2 : 0

Result:

Thus, python program for water jug problem is successfully implemented.