EX.NO: 03
DATE:

DFS-Depth First Search [Water Jug].

AIM:-

create a DFS program to solve the water jug problem using python code.

ALGORITHM:-

1. Initialize the Queue:-
   → create a queue 'q' for BFS
   → create a set visited to keep track of visited states to avoid cycles.
   → Enqueue the intial state (0,0) where the both Jugs are empty.

2. BFS loop:-
   → while queue is not empty.
   * Dequeue the front state (x,y) where x is the amount of water in Jug1 and y is the amount of water in Jug2.
   * If either x == target or y == target then solution is found
   * If the state x,y has been visited before skip to the next iteration.
   * Mark the state (x,y) as visited.
   * For the current state (x,y) generate all possible next states by applying.
     - fill Jug1: (Jug1, y)
     - fill Jug2: (Jug2, x)
     - Empty Jug1: (0,y)
     - Empty Jug2: (x,0)
     - Pour water from Jug1 to Jug2:
       → with capacity of Jug2.
     - Pour water from Jug2 to Jug1:
       → with capacity of Jug1.

3: Check for solution:

→ If the queue is exhausted and the target has been reached, print "Solution is not possible".

→ Otherwise, print the sequence of operations leading to the solution.

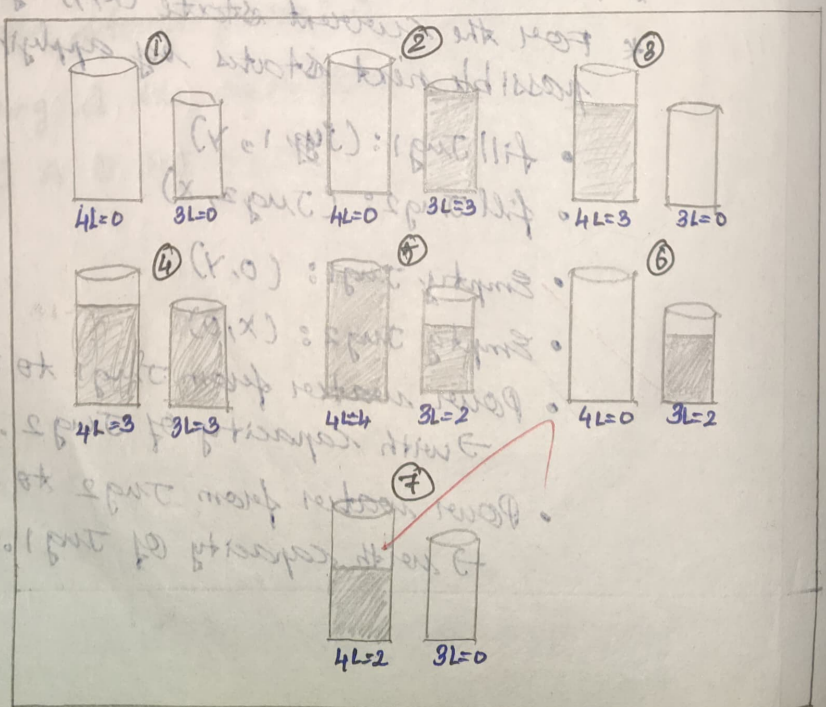
Example:-

1) • Jug1 has 4 liters of water
• Jug2 has 3 liters of water.
• To fill the 2 liters of water.

Soln:

Steps	Gallons of water in 4 gallon Jug	Gallons of water in 3 gallon Jug	Rule Applied
1	0	0	R1: Intial state
2	0	3	R2: Fill 3-gallon Jug
3	3	0	R5: Pour all water from 3 to 4 gallon
4	3	3	R2: Fill 3 gallon Jug
5	4	2	R5: Pour from 3 to 4 until fill
6	0	2	R3: Empty 4 gallon Jug
7	2	0	R7: Pour all water from 3 to 4 Jug

Demonstration:-



Code:-

```
from collection import deque
def solution(a,b,target):
    m={}
    is solvable =False
    Path=[]
    q=deque()
    q.append((0,0))
    while len(q)>0:
    u=q.popleft()
     if (u[0], u[1]) in m:
        continue
    if u[0]>a or u[1]>b or u[0]<0 or u[1]<0
        continue
    path.append([u[0], u[1]])
    m[(u[0], u[1])]=1
    if u[0]==target or u[1]==target:
        is solvable= true
        if u[0]==target:
            if u[1]!=0:
                path.append([u[0], 0])
        else:
            if u[0]!=0:
                path.append([0, u[1]])
        SI=len(path)
        for i in range(SI):
            Print("(", path[i][0], ",", Path[i][1]")")
        break.
    q.append([u[0], b])
    q.append([u[1], a])
    for ap in range(max(a,b)+1):
        c=u[0]+ap
        d=u[1]-ap
        if c==a or (d==0 and d>=0):
            q.append([c,d])
```

```
            c = u[0] - ap
            c = u[1] + ap
            if (c == 0 and c >= 0) and d == b:
                q.append([c, d])
        q.append([a, 0])
        q.append([0, b])
    if not is_solvable:
        Print("Solution not possible")
if __name__ == '__main__':
    Jug1 = int(input("Enter the capacity of Jug1: "))
    Jug2 = int(input("Enter the capacity of Jug2: "))
    Target = int(input("Enter the target amount: "))
    Print("Path from initial state to solution state")
    Solution(Jug1, Jug2, Target).
```

Output:-

```
Enter the capacity of Jug1: 4
Enter the capacity of Jug2: 3
Enter the target amount: 2
Path from initial state to solution state
        (0, 0)
        (0, 3)
        (4, 0)
        (4, 3)
        (3, 0)
        (1, 3)
        (3, 3)
        (4, 2)
        (0, 2)
```

Result:-

Thus the water Jug program is executed and output is verified successfully.