Exp. No:3   DFS : Depth first Sea

[water Jug]

Date :

AIM :- create a DFS program to solve the water Jug problem usin python code.

Algorithm :-

1) Initialize the queue :

Step 1 :- create a queue 'q' for BFS

Step 2 :- create a set visited to keep track of visited states to avoid cycles.

Step 3 :- Enqueue the initial state (0,0) where the both jugs are empty.

2) BFS loop

Step 4 :- while queue is not empty

- Dequeue the front state (x, y) where x is the amount of water in jug (and y is the amount of water in Jug 2.

- If either x == target or y == target then solution is found.

- If the state x, y has been visited before skip to the next iteratio

Mark the state x,y has visited
for the current state (x,y)
Generate all possible next
states by applying.

→ fill jug1 (jug1,y)
→ fill jug 2 (jug 2, x)
→ empty jug 1 ! (0,y)
→ empty jug 2 ! (x,0)
→ pour water from jug1 tojug2
with capacity of jug2
→ pour water from jug2 tojug1
→ with capacity of jug1.

3) Check for solution:

Step 5 :- If the queue is enhausted and the target been reached, print "solution is not possible".

Step 6 :- Otherwise, print the sequence of operation leading to the solution.

```
from collection import dequeue

def solution (a, b, target):
        m= {}
        is solvable = False
        path= []
        q = dequeue ()
        q.append ((0,0))
      while len (q)>0:
         u=q.popleft()
         if (u[0], u[1]) in m;
             contioue
         if is [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])
    SI =len (path)
    for i in range (SI):
         print ("(", path[i][0], ",", path[i]
             break                        [1]")")
         q. append ([u[0], b])
         q. append ([u[1], a])
         for ap in range (max (a,b)+1):
```

```
u[0]+ap
d= u[1]-ap
if c==a or (d==0 and d>=0):
    q.append ([c,d])
 c =2[= u[0]-ap
    c= u[i]+ap
  if (c==0 and c>=0) or 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__":
    jug 1= unit (input ("enter the
            capacity of jug)"))
    jug d= int [input["Enter the
            target amount"))
    print ("path from initial
   state to solution state")
    solution (jug 1, jug2, target)
```

Output :

```
Enter the capacity of Jug 1 : 4
Enter the capacity of Jug 2 : 3
Enter the target amount : 2
Path from initial state to solution
                                -state

    (0,0)        (1,3)
    (0,3)        (3,3)
    (4,0)        (4,2)
    (4,3)        (0,2)
    (3,0)
```

Result :- Thus the water jug program is executed. and output is verified successfully.