Date :

Exp3:

Aim

To implement a DFS program to solve water jug problem

ALGORITHM:

Step1: Start

Step2: Create a queue for BFS. Create a set visited to keep track of visited states to avoid cycles Enqueue the initial state (0,0) where the both jugs are empty

Step3: While queue is empty. Dequeue the front state (x,y) where x is the amount of water in jug 1 and y is the amount of water in jug 2

Step4: If either x == target or y == target then solution is found. If the state x, y has been visited before skip to next iteration. mark the state (x,y) as visited

Step5: For the current state (x,y) generate all possible next states by applying: fill jug 1: (1, y) fill jug 2 (jug 2, x) empty jug 1: (0, y) empty jug 2 (x, 0)

Step6: Pour water from jug 1 to jug 2: with capacity of jug 2

Pour water from jug 2 to jug 1: with capacity of jug 1

Step7: Check for solution: If the Queue is exhausted and the target have been reached, print 'Solution is not possible

Step8: Stop

Date :

CODE:

```
from collection import deque
def solution (a, b, target):
        = { }
    is solvable = false
    Path = []
      q = deque ()
      a.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[i] < 0)
      Continue
    Path append (a[0] u[1]
    m[u[0], u[1]] = 1
   if ( u[0] == target or u[1] == target:
        issolvable = True
        if u[0] == target:
          if u[1] != 0:
            path.append ([u[0], 0])
        else:
        if u[0] != 0
            path. append ([0, u[1])
    s1 = len (path)
  for i in range (s1):
  print ("(", Path[i][0])
 break
  q.append ([u[0], b])
  q.append ([u[1], a])
for ap in range (max (a,b)+1):
        c = u[0] + ap
```

Date:

```
d = u[1] - ap
if ( c == a or (d == 0 and dr = 0)):
    q.append ([c, d])
c = u[0] - ap
d = u[1] + ap
if ( c == 0 and (r = 0) or d == b:
    q.append ([c, d])
    q.append ([a, 0])
    q.append ([0, b])
    if not i solvable:
        Print ("not possible")
if __name__ == '__main__':
    jug1 = int (input ("cap jug 1:"))
    jug2 = int (input ("cap jug 2:"))
    target = int (input ("Target amount:"))
    Print ("path of the solution")
    Solution (jug1, jug2, target)
```

Output

```
Cap of jug 1 = 4
Cap of jug 2 = 3
Target amount = 2
path from the inital to solution state
(0, 0)    (3, 3)
(0, 3)    (4, 2)
(4, 0)    (0, 2)
(4, 3)
(3, 0)
(1, 3)
```

RESULT:

Thus a DFS program to solve water jug problem is implemented and output verified