Exp: No : 3

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 :

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 1 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 iteration.

- Mark the state (x, y) as visited

- for the current state (x, y) generate all possible next states by applying.

→ fill Jug 1 (jug1, y)

→ fill Jug 2 : (jug 2, x)

→ empty Jug 1 : (0, y)

→ empty Jug 2 : (x, 0)

→ 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

3) check for solution:

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

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

Program :

```
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 :
            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])
```

```
SI = len(path)
for i in range(SI):
    print("(". path[i][0], ",", path[i][i]")")
    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])
    E = AC = 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_':
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 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.