Exp.NO :

Date :

Depth First search [Water Jug]

Aim

To write a python program to solve a water jug problem

Algorithm

Step 1 : Initialize the queue

i) Create a queue q for BFS

ii) Create a set visited to keep track of visited state to avoid cycles

iii) Enqueue the initial state (0,0) where the both jugs are empty

Step 2 : BFS loop

i) While queue is not empty

ii) 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

iii) If either x == target or y == target then solution is found

iv) If the state x,y has been visited before skip to next iteration

v) Mark the state (x,y) as visited

vi) For the current state (x,y) generate all possible next states by applying

* fill Jug 1 (Jug 1, 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

Step 3: Check for solution

i) If the queue is exchanged and the target have been reached, print "solution is not possible"

ii) Otherwise, print the sequence of operation leading to the solution

Program

```
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.top left()
      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]])
    SZ = len(path)
    for i in range(SI):
      Print("(", path[i][0], ",", path[i][i]")
    break
    q.append([u[0], b])
```

Output:-

Enter the capacity of Jug 1 : 4

Enter the capacity of Jug 2 : 3

~~Enter the capacity of Jug 2 : 2~~

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 problem is executed and output is verified.