Exp No: 3

# Depth First Search [water Jug]

## Aim:

Create a DFS program to solve the water Jug problem using python code.

## Algorithm:

i) Start

ii) Create a queue 'q' for BFS.

iii) Create a Set visited to keep track of visited states to avoid cycles.

iv) Enqueue the initial State (0,0) where the both Jugs are empty.

BFS Loop:

v) 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: (jug2, x)

→ empty Jug 1: (0, y)

→ empty Jug2: (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.

Check for solution:

vi) If the queue is exhausted and the target been reached. print "Solution is not possible".

vii) otherwise, print 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.
        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][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 >=
                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 capaci
                of Jug1"))
    Jug2 = int(input("Enter the capacit
                of Jug2"))
    target = int(input("Enter the target
                amount"))
```

print ("Path from initial State to Solution State")

Solution (Jug1, Jug2, target)

Output:

Input 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.