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 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 : (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

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.

```
from collection import deque
def solution (a, b, target) :
    m = {}
```

```
if not solvable = false
  Path = []
  q = deque()
  q.append([0,0])
  while len(q)>0:
    u = q.popleft()
  if [u[0], u[1]] in m:
    continue
  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:
      if 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])
        c = u[0] = u[0] - ap
        c = u[1] + ap
    if (i == 0 and c >= 0) or d == b:
        q.append([c,d])
    q.append([a,0])
    q.append([0,b])
    if not isSolvable:
        print("Solution not possible")
if __name__ == '__main__':
    jug1 = int(input("Enter the capacity of jug1"))
    jug2 = int(input("Enter the target amount"))
print("Path from initial state to solution state")
solution(jug1, 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.