**Experiment NO: 2**  **Date:**

**Title:** Water Jug Problem

**Aim:** To implement solution to the Water Jug problem using breadth-first search technique.

**Problem Statement:** There are two water jugs, a 4 gallon and another of 3 gallon. Neither of them has any measuring mark on it. There is a pump that can be used to fill the jugs with water. It is required to measure 2 gallon water in the 4 gallon jug. Find the solution to the Water Jug problem using depth-first search technique.

**Description:**The Water Jug Problem, also known as the Die Hard Problem, is a classic puzzle in computer science and recreational mathematics. It involves solving a problem related to filling and emptying water jugs to achieve a specific goal. The problem is often used to illustrate problem-solving techniques and algorithms.  
To formulate a solution to the problem, firstly we need to formulate the problem in a manner where understanding the solution steps become easier. The accepted convention is to use a 2-tuple to represents approachable states: (x, y) ,where x stands quantity of water in 4 gallon jug, stands quantity of water in 2 gallon jug. We assume the start state is (0,0), because initially both jugs are empty. We create a state space tree , by representing (0,0) as the root of the tree, and with children of each node representing the state that can be accessed via that state(filling or emptying or transferring of water from one jug to another), such that the constraints of the problem are not violated. To prevent infinite loops, we decide to keep track of which steps have been already obtained (maybe through some alternate path), and never append a child state that has already been visited/obtained.

The allowed operations are:

1. Fill jug A: (capacity(a), jug(b))
2. Fill jug B: (jug(a), capacity(b))
3. Empty jug A: (0, jug(b))
4. Empty jug B: (jug(a), 0)
5. Pour from A to B: (jug(a) - pour\_amount, jug(b)+ pour\_amount)
6. Pour from B to A: (jug(a)+ pour\_amount, jug(b)- pour\_amount)

Jug(a)=amount of water in jug A

Jug(a)=amount of water in jug A

We use Depth First Search technique to solve this problem. In this technique, we use LIFO(Last In First Out). Thus at every step, we make a decision, and the then explore all paths through this decision. And if this decision leads to win situation, we stop. For implementing the program for this problem, we use the data structure **implicit stack** ie by use of recursion which follows the concept of last in first out.

**Program:**

import sys

START = (0, 0, 'Start')

GOAL\_SET = [(2, 0), (2, 1), (2, 2), (2, 3)]

Visited = []

Visited.append(START[:2])

Parent = {}

Parent[START] = None

def DFSAdd(tuple, tup):

if (tup[:2] not in Visited):

Parent[tup] = tuple

Visited.append(tup[:2])

Rules(tup)

def Rules(tuple):

(jug1, jug2, type) = tuple

if (tuple[:2] in GOAL\_SET):

path = []

while tuple in Parent and Parent[tuple] is not None:

path.append(tuple)

tuple = Parent[tuple]

path.append(START)

path.reverse()

print(\*path, sep='\n')

sys.exit()

if (jug1 < 4):

tup = (4, jug2, 'Filled 4g')

DFSAdd(tuple, tup)

if (jug2 < 4):

tup = (jug1, 3, 'Filled 3g')

DFSAdd(tuple, tup)

if (jug1 > 0):

tup = (0, jug2, 'Emptied 4g')

DFSAdd(tuple, tup)

if (jug2 > 0):

tup = (jug1, 0, 'Emptied 3g')

DFSAdd(tuple, tup)

if (jug1+jug2 < 7 and 4-jug1 < jug2):

tup = (4, jug2-(4-jug1), 'Transferred 3g to 4g with some remaining')

DFSAdd(tuple, tup)

if (jug1+jug2 < 7 and 3-jug2 < jug1):

tup = (jug1-(3-jug2), 3, 'Transferred 4g to 3g with some remaining')

DFSAdd(tuple, tup)

if (jug1+jug2 < 4):

tup = (jug1+jug2, 0, 'Entirely transferred 3g to 4g')

DFSAdd(tuple, tup)

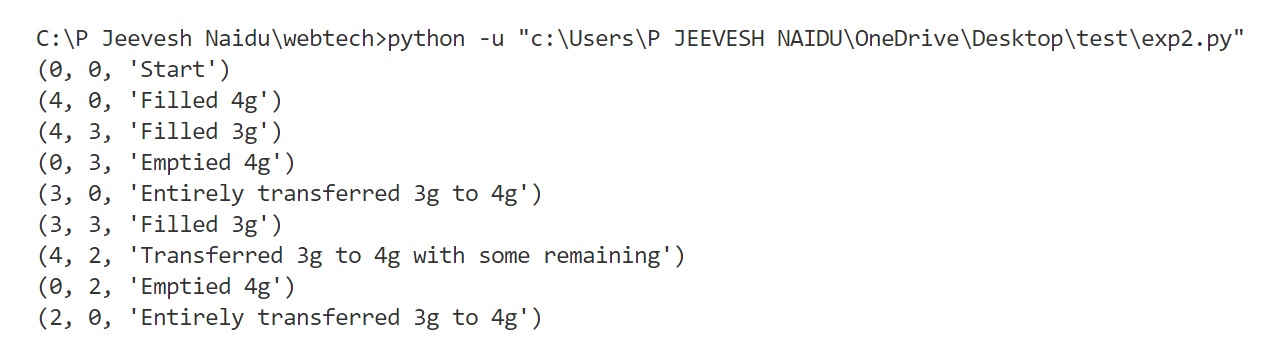
if (jug1+jug2 < 3):

tup = (0, jug1+jug2, 'Entirely transferred 4g to 3g')

DFSAdd(tuple, tup)

Rules(START)

**Output:**

****

**Conclusion:**

The water jug problem was studied and implemented its solution was implemented using depth-first search technique in Python.