1. Implement and Demonstrate Depth First Search Algorithm on Water Jug Problem.

```
import queue
import time
import random

dfsq = queue.Queue()

class node:
    def __init__(self, data):
        self.x = 0
        self.y = 0
        self.parent = data

    def printnode(self):
        print("(", self.x, ",", self.y, ")")

def generateAllSuccessors(cnode):
    list1 = []
    list_rule = []
    while len(list_rule) < 8:
        rule_no = random.randint(1, 8)
        if (not rule_no in list_rule):
            list_rule.append(rule_no)
            nextnode = operation(cnode, rule_no)
            if nextnode != None and not IsNodeInlist(nextnode, visitednodelist):
                list1.append(nextnode)
    """for rule in range (1,9):
        nextnode = operation(cnode,rule) #current node
        if nextnode != None :
            list1.append(nextnode)"""
    return list1


def operation(cnode, rule):
    x = cnode.x
    y = cnode.y
    if rule == 1:c
        if x < maxjug1:
            x = maxjug1
        else:
            return None
    elif rule == 2:
        if y < maxjug2:
            y = maxjug2
        else:
            return None
    elif rule == 3:
        if x > 0:
            x = 0
        else:
```

```
            return None
        elif rule == 4:
            if y > 0:
                y = 0
            else:
                return None
        elif rule == 5:
            if x + y >= maxjug1:
                y = y - (maxjug1 - x)
                x = maxjug1
            else:
                return None
        elif rule == 6:
            if x + y >= maxjug2:
                x = x - (maxjug2 - y)
                y = maxjug2
            else:
                return None
        elif rule == 7:
            if x + y < maxjug1:
                x = x + y
                y = 0
            else:
                return None
        elif rule == 8:
            if x + y < maxjug2:
                x = 0
                y = x + y
            else:
                return None
        if (x == cnode.x and y == cnode.y):
            return None
        nextnode = node(cnode)
        nextnode.x = x
        nextnode.y = y
        nextnode.parent = cnode
        return nextnode


def pushlist(list1):
    for m in list1:
        dfsq.put(m)


def popnode():
    if (dfsq.empty()):
        return None
    else:
        return dfsq.get()
```

```
def isGoalNode(cnode, gnode):
    if (cnode.x == gnode.x and cnode.y == gnode.y):
        return True
    return False


visitednodelist = []


def dfsMain(initialNode, GoalNode):
    dfsq.put(initialNode)
    while not dfsq.empty():
        visited_node = popnode()
        print("Pop node:")
        visited_node.printnode()
        if isGoalNode(visited_node, GoalNode):
            return visited_node
        successor_nodes = generateAllSuccessors(visited_node)
        pushlist(successor_nodes)
    return None


def IsNodeInlist(node, list1):
    for m in list1:
        if (node.x == m.x and node.y == m.y):
            return True
        return False


def printpath(cnode):
    temp = cnode
    list2 = []
    while (temp != None):
        list2.append(temp)
        temp = temp.parent
    list2.reverse()
    for i in list2:
        i.printnode()
    print("Path Cost:", len(list2))


if __name__ == '__main__':

    list2 = []
    maxjug1 = int(input("Enter value of maxjug1:"))
    maxjug2 = int(input("Enter value of maxjug2:"))
    initialNode = node(None)
    initialNode.x = 0
    initialNode.y = 0
```

```
    initialNode.parent = None
    GoalNode = node(None)
    GoalNode.x = int(input("Enter value of Goal in jug1:"))
    GoalNode.y = 0
    GoalNode.parent = None
    start_time = time.time()
    solutionNode = dfsMain(initialNode, GoalNode)
    end_time = time.time()
    if (solutionNode != None):
        print("Solution can Found:")
    else:
        print("Solution can't be found.")
    printpath(solutionNode)
    diff = end_time - start_time
    print("Execution Time:", diff * 1000, "ms")
```

Output:

```
Enter value of maxjug1:5
Enter value of maxjug2:3
Enter value of Goal in jug1:4
Solution can Found:
( 0 , 0 )
( 0 , 3 )
( 3 , 0 )
( 3 , 3 )
( 5 , 1 )
( 0 , 1 )
( 1 , 0 )
( 1 , 3 )
( 4 , 0 )
Path Cost: 9
```