Aim: Write a program to implement Water Jug Problem using BFS.

A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in the 4-gallon jug?

Let X represents the content of the water in 4-gallon jug.

Let Y represent the content of the water in 3-gallon jug.

Write a program in python/Java to define a set of operators (Rules) that will take us from one state to another:

Start from initial state (X=0, Y=0) Reach

any of the Goal states

(X=2, Y=0)
(X=2, Y=1)
(X=2, Y=2)
(X=2, Y=3)

Find the minimum number of steps to reach any the above mentioned goal states.

WATER-JUG PROBLEM

- The production rules are formulated as follows:-
- Rule 1: (x, y) if x < 4 -> (4, y) (Fill the 4 liter jug)
- Rule 2: (x, y) if y < 3 -> (x, 3) (Fill the 3 liter jug)
- Rule 3: (x, y) if x > 0 -> (x-d, y) (Pour some water out from 4 liter jug)
- Rule 4: (x, y) if y > 0 -> (x, y-d) (Pour some water out from 3 liter jug)
- Rule 5: (x, y) if x > 0 -> (0, y) (Empty the 4 liter jug)
- Rule 6: (x, y) if y > 0 -> (x, 0) (Empty the 3 liter jug)
- Rule 7: (x, y) if x + y>= 4 and y > 0 -> (4, y-(4-x)) (Fill the 4 liter jug by pouring some water from 3 liter jug)
- Rule 8: (x, y) if x + y>= 3 and x > 0 -> (x-(3-y), 3) (Fill the 3 liter jug by pouring some water from 4 liter jug)
- Rule 9: (x, y) if x + y <= 4 and y > 0 -> (x + y, 0) (Empty 3 liter jug by pouring all its water in to 4 liter jug)
- Rule 10: (x, y) if x + y <= 3 and x > 0 -> (0, x +y) (Empty 4 liter jug by pouring all its water in to 3 liter jug)

Program:

```
import queue
import time

bfsq = 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=[]
    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 :
        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:
        bfsq.put(m)

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

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

def bfsMain(initialNode,GoalNode):
    bfsq.put(initialNode)
    while not bfsq.empty():
        visited_node = popnode()
        if isGoalNode(visited_node,GoalNode):
            return visited_node
        successor_nodes = generateAllSuccessors(visited_node)
        pushlist(successor_nodes)
    return None

def printpath(cnode):
    temp = cnode
    list1=[]
    while(temp != None):
        list1.append(temp)
        temp = temp.parent
    list1.reverse()
    for i in list1:
        i.printnode()
    print("Path Cost:",len(list1))

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 = bfsMain(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:

```
F:\Engineering\SEM #7\AI>python prac2.py
Enter value of maxjug1:8
Enter value of maxjug2:9
Enter value of Goal in jug1:2
Solution can Found:
( 0 , 0 )
( 0 , 9 )
( 8 , 1 )
( 0 , 1 )
( 1 , 0 )
( 1 , 9 )
( 8 , 2 )
( 0 , 2 )
( 2 , 0 )
Path Cost: 9
Execution Time: 31.948566436767578 ms
```