**AI EXPERIMENT 06**

***Program: [2 JUG PROBLEM]***  
#Capacity Of j1 and j2  
j1cap=int(input("Enter capacity of j1: "))  
j2cap=int(input("Enter capacity of j2: "))  
  
#What fnal state you want  
final\_j1=int(input("Enter final capacity of j1: "))  
final\_j2=int(input("Enter final capacity of j2: "))  
  
#Initialise the jugs  
jug1=0  
jug2=0  
print("Initial: [",jug1," , ",jug2," ]")  
print("Final: [",final\_j1," , ",final\_j2," ]")  
print("1 => Fill jug A completely")  
print("2 => Fill jug B completely")  
print("3 => Pour from jug A completely onto the ground")  
print("4 => Pour from jug B completely onto the ground")  
print("5 => Pour from Jug A to Jug B until it s full or A becomes empty")  
print("6 => Pour from Jug B to Jug A until it s full or B becomes empty")  
print("7 => Pour all water from Jug B to Jug A")  
print("8 => Pour all water from Jug A to Jug B")  
  
while jug1!=final\_j1 or jug2!=final\_j2:  
    op=int(input("Enter the rule number: "))  
    if op==1:  
        jug1=j1cap  
    elif op==2:  
        jug2=j2cap  
    elif op==3:  
        jug1=0  
    elif op==4:  
        jug2=0  
    elif op==5:  
        if jug1+jug2<=j2cap:  
            jug2=jug1+jug2  
            jug1=0  
        else:  
            jug1=(jug1+jug2)-j2cap  
            jug2=j2cap  
    elif op==6:  
        if jug1+jug2<=j1cap:  
            jug1=jug1+jug2  
            jug2=0  
        else:  
            jug2=(jug1+jug2)-j1cap  
            jug1=j1cap  
    elif op==7:  
        jug1=jug1+jug2  
        if jug1>j1cap:  
            jug1=j1cap  
        jug2=0  
    elif op==8:  
        jug2=jug1+jug2  
        if jug2>j2cap:  
            jug2=j2cap  
        jug1=0      
    print("j1: ",jug1," j2: ",jug2)      
print("Goal Achieved")    

***Output:***

Enter capacity of j1: 4

Enter capacity of j2: 3

Enter final capacity of j1: 2

Enter final capacity of j2: 0

Initial: [ 0 , 0 ]

Final: [ 2 , 0 ]

1 => Fill jug A completely

2 => Fill jug B completely

3 => Pour from jug A completely onto the ground

4 => Pour from jug B completely onto the ground

5 => Pour from Jug A to Jug B until it s full or A becomes empty

6 => Pour from Jug B to Jug A until it s full or B becomes empty

7 => Pour all water from Jug B to Jug A

8 => Pour all water from Jug A to Jug B

Enter the rule number: 2

j1: 0 j2: 3

Enter the rule number: 6

j1: 3 j2: 0

Enter the rule number: 2

j1: 3 j2: 3

Enter the rule number: 6

j1: 4 j2: 2

Enter the rule number: 3

j1: 0 j2: 2

Enter the rule number: 6

j1: 2 j2: 0

Goal Achieved

***Program: [3 JUG PROBLEM]***  
#Capacity Of j1 and j2  
x=int(input("Enter capacity of j1: "))  
y=int(input("Enter capacity of j2: "))  
z=int(input("Enter capacity of j3: "))  
  
#What fnal state you want  
goala=int(input("Enter final state of j1: "))  
goalb=int(input("Enter final state of j2: "))  
goalc=int(input("Enter final state of j3: "))  
  
initial\_state=(8,0,0) #Init\_state  
visited={}            #keep track of visisted states  
path=[]               #path that leadf to soln  
  
def wjp(state):  
    #get curent states  
    #print("Current state: ",state)  
    a=state[0]  
    b=state[1]  
    c=state[2]  
    #check if the goal is reached  
    if a==goala and b==goalb:  
        path.append(state);  
        return True;  
    #Check if this state was generated  
    if state in visited:  
        #print("Repeated state")  
        return False  
    #Else mark it visited  
    visited[state]=1  
    #TRy 4 cases wrt to each jug  
   #Try to pour from A to B,C  
    if a>0:  
        #Pour a to b case  
        if a+b<=y:  
            if wjp((0,a+b,c)):  
                path.append(state)  
                return True  
        else:  
            if wjp((a-(y-b),y,c)):  
                path.append(state)  
                return True  
        #Pour a to c case  
        if a+c<=z:  
            if wjp((0,b,a+c)):  
                path.append(state)  
                return True  
        else:  
            if wjp((a-(z-c),b,z)):  
                path.append(state)  
                return True  
   #Try to pour from B to A,C  
    if b>0:  
        #Pour a to b case  
        if b+a<=x:  
            if wjp((a+b,0,c)):  
                path.append(state)  
                return True  
        else:  
            if wjp((x,b-(x-a),c)):  
                path.append(state)  
                return True  
        #Pour a to c case  
        if b+c<=z:  
            if wjp((a,0,b+c)):  
                path.append(state)  
                return True  
        else:  
            if wjp((a,b-(z-c),z)):  
                path.append(state)  
                return True  
   #Try to pour from C to A,B  
    if c>0:  
        #Pour a to b case  
        if c+a<=x:  
            if wjp((a+c,b,0)):  
                path.append(state)  
                return True  
        else:  
            if wjp((x,b,c-(x-a))):  
                path.append(state)  
                return True  
        #Pour a to c case  
        if c+b<=y:  
            if wjp((a,b+c,0)):  
                path.append(state)  
                return True  
        else:  
            if wjp((a,y,c-(y-b))):  
                path.append(state)  
                return True  
  
    return False  
if wjp(initial\_state):  
   path.reverse()  
   print("Path is: ")  
   for p in path:  
      print(p)     
else:  
   print("No solution exists")

***Output:***

**Enter capacity of j1: 8**

**Enter capacity of j2: 5**

**Enter capacity of j3: 3**

**Enter final state of j1: 4**

**Enter final state of j2: 4**

**Enter final state of j3: 0**

**Path is:**

**(8, 0, 0)**

**(3, 5, 0)**

**(0, 5, 3)**

**(5, 0, 3)**

**(5, 3, 0)**

**(2, 3, 3)**

**(2, 5, 1)**

**(7, 0, 1)**

**(7, 1, 0)**

**(4, 1, 3)**

**(4, 4, 0)**