EXPERIMENT-2

Name: Sakshi Jadhav **Roll No: 49**
Class: BEITB **Batch: B3**
Grade: **Sign:**

Problem Definition with State Space Representation

2.1 Implement Water Jug Problem Using Problem Formulation

Aim: Implement water jug problem using BFS or DFS (Un-Informed Search).

Theory:
Problem Statement
In the **water jug problem in Artificial Intelligence**, we are provided with two jugs: one having the capacity to hold 3 gallons of water and the other has the capacity to hold 4 gallons of water. There is no other measuring equipment available and the jugs also do not have any kind of marking on them. So, the agent's task here is to fill the 4-gallon jug with 2 gallons of water by using only these two jugs and no other material. Initially, both our jugs are empty.

So, to solve this problem, following set of rules were proposed:

Production rules for solving the water jug problem
Here, let ***x*** denote the 4-gallon jug and ***y*** denote the 3-gallon jug.

S.No.	Initial State	Condition	Final state	Description of action taken
1.	(x,y)	If x<4	(4,y)	Fill the 4 gallon jug completely
2.	(x,y)	if y<3	(x,3)	Fill the 3 gallon jug completely
3.	(x,y)	If x>0	(x-d,y)	Pour some part from the 4 gallon jug
4.	(x,y)	If y>0	(x,y-d)	Pour some part from the 3 gallon jug
5.	(x,y)	If x>0	(0,y)	Empty the 4 gallon jug
6.	(x,y)	If y>0	(x,0)	Empty the 3 gallon jug
7.	(x,y)	If (x+y)<7	(4, y-[4-x])	Pour some water from the 3 gallon jug to fill the four gallon jug
8.	(x,y)	If (x+y)<7	(x-[3-y],y)	Pour some water from the 4 gallon jug to fill the 3 gallon jug.
9.	(x,y)	If (x+y)<4	(x+y,0)	Pour all water from 3 gallon jug to the 4 gallon jug
10.	(x,y)	if (x+y)<3	(0, x+y)	Pour all water from the 4 gallon jug to the 3 gallon jug

The listed production rules contain all the actions that could be performed by the agent in transferring the contents of jugs. But, to solve the water jug problem in a minimum number of moves, following set of rules in the given sequence should be performed:

Solution of water jug problem according to the production rules:

S.No.	4 gallon jug contents	3 gallon jug contents	Rule followed
1.	0 gallon	0 gallon	Initial state
2.	0 gallon	3 gallons	Rule no.2
3.	3 gallons	0 gallon	Rule no. 9
4.	3 gallons	3 gallons	Rule no. 2
5.	4 gallons	2 gallons	Rule no. 7
6.	0 gallon	2 gallons	Rule no. 5
7.	2 gallons	0 gallon	Rule no. 9

On reaching the 7th attempt, we reach a state which is our goal state. Therefore, at this state, our problem is solved.

Program Code in Python:
Problem Statement: There are two jugs (supposed capacity of 3 and 5) and we need to fill the jug in such a way that a 5 liters capacity jug should contain 4 liters of water.
from collections import deque

```
def BFS(a, b, target):
    m = {}
    isSolvable = False
    path = []
    q = deque()

    # Initialing with initial state
    q.append((0, 0))

    while (len(q) > 0):
        # Current state
        u = q.popleft()

        # If this state is already visited
        if ((u[0], u[1]) in m):
            continue

        # Doesn't met jug constraints
        if ((u[0] > a or u[1] > b or
            u[0] < 0 or u[1] < 0)):
            continue
```

```
# Filling the vector for constructing the solution path
path.append([u[0], u[1]])

# Marking current state as visited
m[(u[0], u[1])] = 1

# If we reach solution state, put ans=1
if (u[0] == target or u[1] == target):
    isSolvable = True

    if (u[0] == target) and if (u[1] != 0):
        # Fill final state
        path.append([u[0], 0])
    elif (u[0] != 0):
        # Fill final state
        path.append([0, u[1]])

    # Print the solution path
    sz = len(path)
    for i in range(sz):
        print("(", path[i][0], ",",
                path[i][1], ")")
    break

# If we have not reached final state
# start developing intermediate  states
q.append([u[0], b]) # Fill Jug2
q.append([a, u[1]]) # Fill Jug1

for ap in range(max(a, b) + 1):

    # Pour amount ap from Jug2 to Jug1
    c = u[0] + ap
    d = u[1] - ap

    # Check if this state is possible or not
    if (c == a or (d == 0 and d >= 0)):
        q.append([c, d])

    # Pour amount ap from Jug 1 to Jug2
    c = u[0] - ap
    d = u[1] + ap

    # Check if this state is possible or not
    if ((c == 0 and c >= 0) or d == b):
        q.append([c, d])
```

```
            # Empty Jug2
            q.append([a, 0])

            # Empty Jug1
            q.append([0, b])

        # No, solution exists if ans=0
        if (not isSolvable):
            print ("No solution")

#Calling the method
Jug1, Jug2, target = 4, 3, 2
print("Path from initial state to solution state ::")
BFS(Jug1, Jug2, target)
```

Output:

```
Path from initial state to solution state :-
( 0 , 0 )
( 0 , 3 )
( 4 , 0 )
( 4 , 3 )
( 3 , 0 )
( 1 , 3 )
( 3 , 3 )
( 4 , 2 )
( 0 , 2 )
```

Conclusion: Thus, we have successfully implemented the water jug problem.