PROGRAM TITLE-3

WATER JUG PROBLEM

AIM:

To write and execute the python program for Water Jug Problem.

PROCEDURE:

1. State Representation:
   - A state in the problem is represented by the current amount of water in each jug.
2. Initialization:
   - Define the initial state (usually both jugs are empty) and the target state (the desired amount of water).
3. Generate Successors:
   - From the current state, generate all possible successor states by applying the allowed operations (filling a jug, emptying a jug, or pouring water from one jug to another).
4. Depth-First Search:
   - Use a depth-first search to explore the state space.
   - At each step, choose one of the successor states.
   - Recursively apply the algorithm to the chosen successor until the target state is reached or no more valid moves are possible.
5. Backtracking:
   - If the target state is reached, backtrack to the initial state to obtain the sequence of moves that lead to the solution.

CODING:

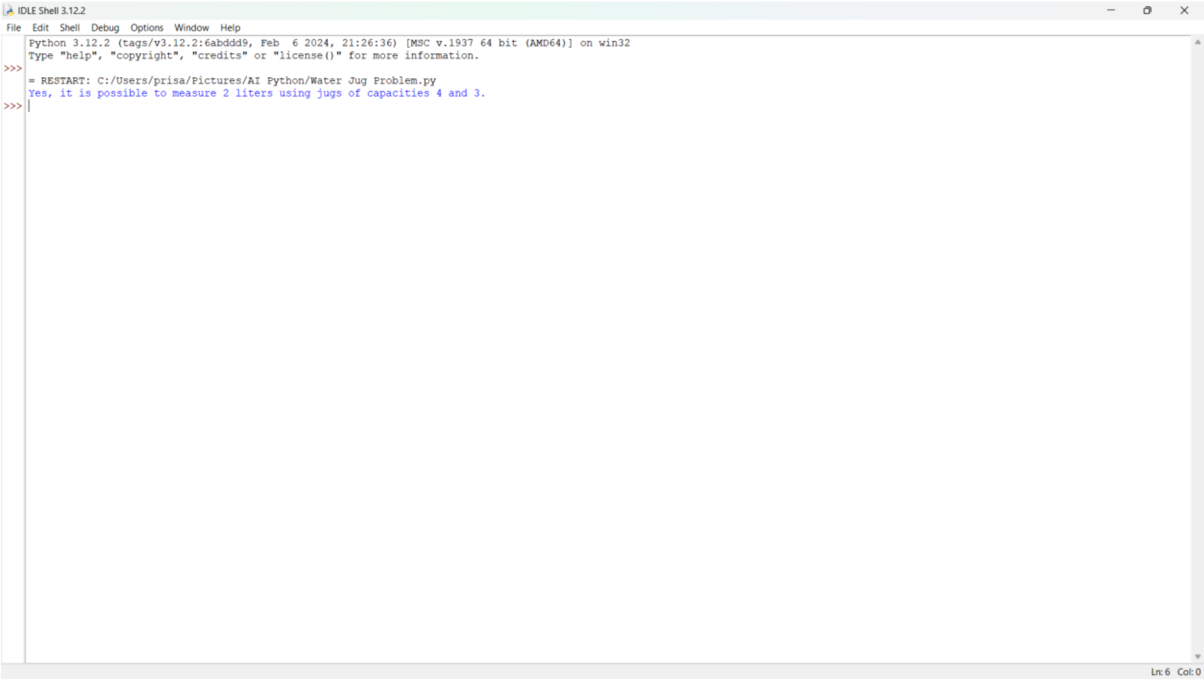
```
from collections import deque

def BFS(a, b, target):

    m = {}

    isSolvable = False

    path = []
```

```
q = deque()
q.append((0, 0))
while (len(q) > 0):
        u = q.popleft()# If this state is already visited
        if ((u[0], u[1]) in m):
                continue
        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):
                        if (u[1] != 0):
                                # Fill final state
                                path.append([u[0], 0])
                else:
                        if (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
    # then, start developing intermediate
    # states to reach solution state
    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")

# Driver code
if __name__ == '__main__':

    Jug1, Jug2, target = 4, 3, 2
    print("Path from initial state "
        "to solution state ::")

    BFS(Jug1, Jug2, target)
```

OUTPUT:



RESULT:

Thus the program has been written and verified successfully.