PROGRAM TITLE-3

WATER JUG PROBLEM

AIM:

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

PROCEDURE:

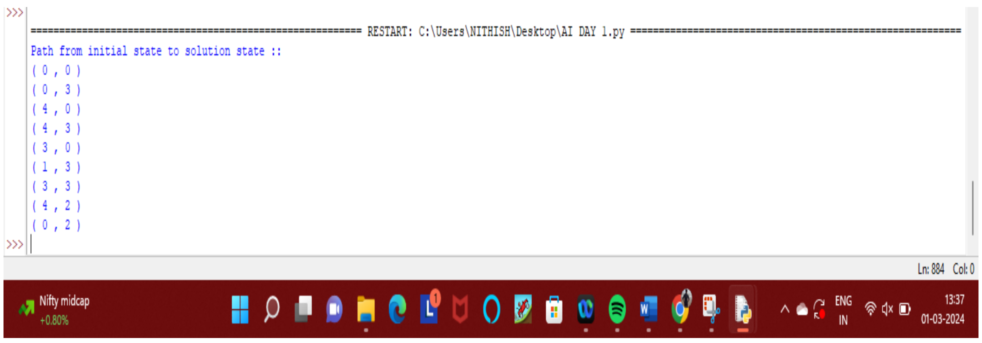
1. **Variable Initialization:**
   - Define variables to represent the capacities of the two jugs and the desired target amount of water. Initialize the jugs with water amounts (initial state).
2. **Define Operations:**
   - Identify and define the possible operations that can be performed on the jugs. This typically includes filling a jug, emptying a jug, or pouring water from one jug to another.
3. **State Representation:**
   - Choose a suitable data structure to represent the state of the system, including the current water amounts in each jug.
4. **Implement Breadth-First Search (BFS):**
   - Use a BFS algorithm to explore all possible states of the system. Start with the initial state and perform valid operations to generate new states. Keep track of visited states to avoid infinite loops.
5. **Goal Test and Solution Output:**
   - Define a goal test to check whether the current state matches the target amount of water. If a solution is found, print or store the sequence of operations to reach that 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:



```
>>>
=========================================================== RESTART: C:\Users\NITHISH\Desktop\AI DAY 1.py ==========================================================
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 )
>>>
```

RESULT:

Thus the program has been written and verified successfully.