**PROGRAM TITLE-3**

**WATER JUG PROBLEM**

**AIM:**

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

**PROCEDURE:**

1. **Initialization:**
   * Start with two empty jugs of different capacities: Jug A and Jug B.
   * Define the capacities of Jug A and Jug B (capacity\_A, capacity\_B).
   * Define the desired target volume of water to be measured (target\_volume).
2. **Fill Jug:**
   * Fill Jug A or Jug B with water from a water source until it's full.
   * Repeat until one of the jugs contains the desired target volume of water, or until the target cannot be achieved:
     + Fill Jug A with water from the water source if it's not full.
     + Fill Jug B with water from the water source if it's not full.
3. **Transfer Water:**
   * Pour water from one jug to the other, following these rules:
     + If Jug A is full, pour water from Jug A to Jug B until Jug B is full or until Jug A is empty.
     + If Jug B is full, pour water from Jug B to Jug A until Jug A is full or until Jug B is empty.
     + Repeat these steps until the desired target volume is reached, or until it's not possible to reach the target.
4. **Check Solution:**
   * After each pouring action, check if the target volume is reached in either Jug A or Jug B.
     + If the target volume is reached in either jug, stop the process and consider the solution found.
     + If the target volume cannot be reached, continue pouring water and transferring between jugs.
5. **End:**
   * Once the target volume is reached in either Jug A or Jug B, the algorithm terminates, and the solution is obtained.

**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\_\_':

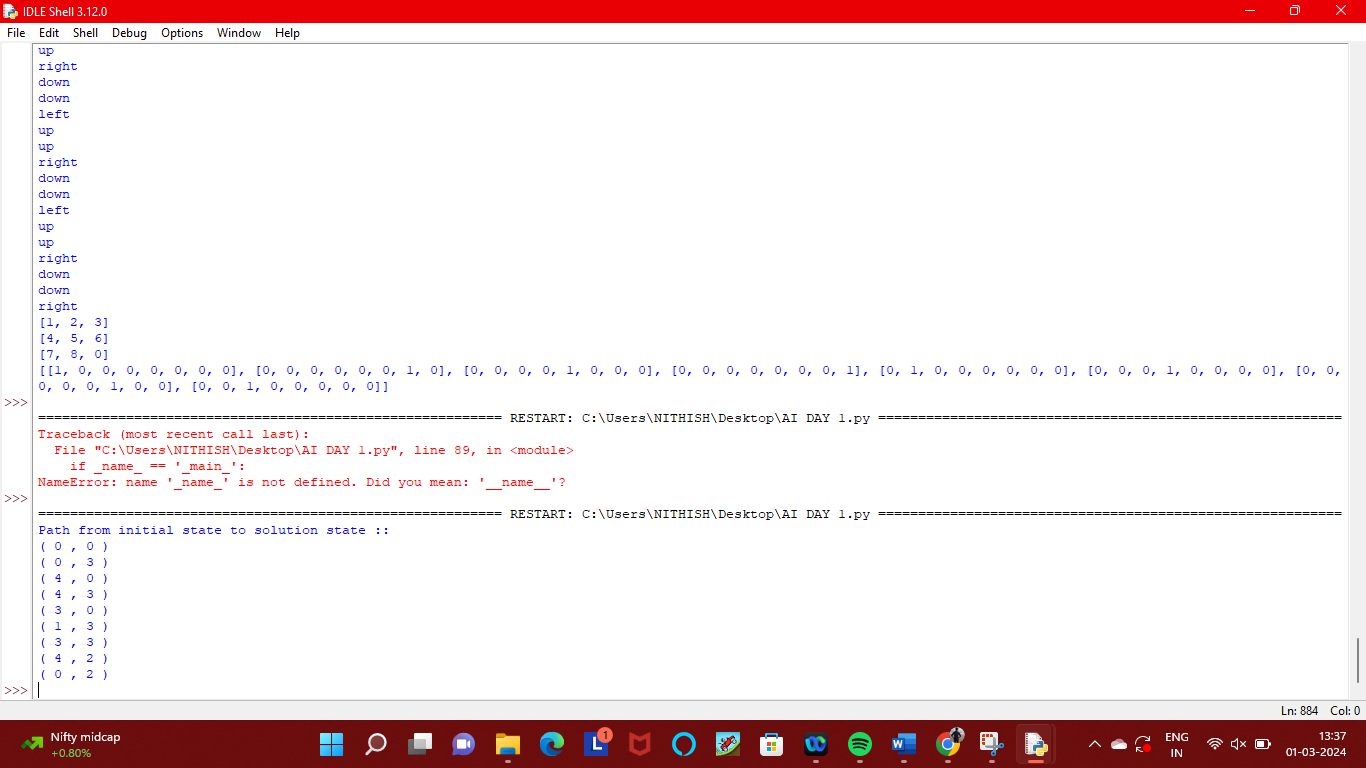
Jug1, Jug2, target = 4, 3, 2

print("Path from initial state "

"to solution state ::")

BFS(Jug1, Jug2, target)

**OUTPUT:**

****

**RESULT:**

Thus the program has been written and verified successfully.