3. Write the python program for Water Jug Problem

**AIM :** program for Water Jug Problem

**ALGORITHM :**

1. **Input:** Capacities of two jugs (x and y), and the target amount of water to be measured (target).
2. **Check Validity:** Check if the target amount is achievable using the given jug capacities. If the target amount is greater than the maximum capacity of both jugs or if the target amount is not divisible by the greatest common divisor (GCD) of the jug capacities, then no solution is possible.
3. **Initialize:** Create an empty set visited\_states to keep track of visited states and a queue queue to store states and actions.
4. **Enqueue Initial State:** Enqueue the initial state (0, 0) along with an empty list of actions into the queue.
5. **While Queue is Not Empty:**

* Dequeue the front state and associated actions from the queue.
* Check if the state has already been visited. If so, skip this iteration.
* Mark the state as visited by adding it to the visited\_states set.
* Check if either jug has the desired amount of water. If so, return the list of actions as the solution.
* Perform the following actions for each possible operation (fill, empty, pour) on both jugs:
* Fill the first jug.
* Fill the second jug.
* Empty the first jug.
* Empty the second jug.
* Pour water from the first jug to the second jug.
* Pour water from the second jug to the first jug.
* Enqueue the resulting states along with the updated list of actions

1. **No Solution Found:** If the queue becomes empty and no solution is found, return None to indicate that no solution is possible.

**PROGRAM :**

def gcd(a, b):

while b:

a, b = b, a % b

return a

def water\_jug\_problem(x, y, target):

if target > max(x, y) or target % gcd(x, y) != 0:

return None

visited\_states = set()

stack = [(0, 0)]

while stack:

current\_state = stack.pop()

if current\_state in visited\_states:

continue

visited\_states.add(current\_state)

if current\_state[0] == target or current\_state[1] == target:

return visited\_states

stack.append((x, current\_state[1]))

stack.append((current\_state[0], y))

stack.append((0, current\_state[1]))

stack.append((current\_state[0], 0))

pour\_amount = min(current\_state[0], y - current\_state[1])

stack.append((current\_state[0] - pour\_amount, current\_state[1] + pour\_amount))

pour\_amount = min(x - current\_state[0], current\_state[1])

stack.append((current\_state[0] + pour\_amount, current\_state[1] - pour\_amount))

return None

x = int(input("Enter the capacity of the first jug: "))

y = int(input("Enter the capacity of the second jug: "))

target = int(input("Enter the target amount of water: "))

solution = water\_jug\_problem(x, y, target)

if solution:

print("Solution found!")

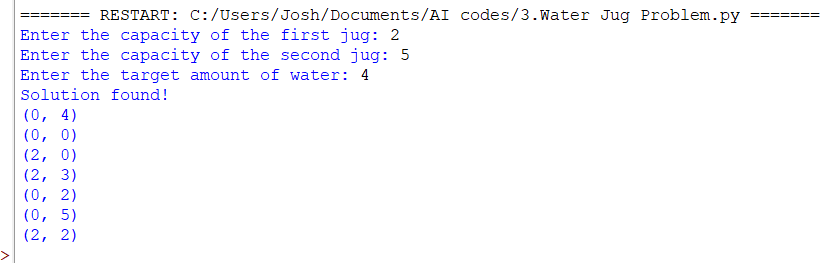
for state in solution:

print(state)

else:

print("No solution found.")

**OUT PUT :**

****