**Aim:**

To write and execute the python program for the water jug problem

**Procedure:**

1. **Initialize Variables:**
   * **visited**: A set to keep track of visited states to avoid duplicates.
   * **queue**: A queue to store the state of the jugs and the actions taken to reach that state. It starts with the initial state of both jugs being empty (0, 0, []).
2. **Breadth-First Search (BFS) Loop:**
   * The program enters a while loop that continues as long as the queue is not empty.
   * In each iteration:
     + Dequeue the front element from the queue, representing the current state of the jugs (**jug\_4**, **jug\_3**), and the actions taken so far (**actions**).
     + Check if the current state has already been visited. If it has, skip to the next iteration.
     + If the 4-gallon jug (**jug\_4**) contains exactly 2 gallons, return the list of actions, indicating a successful solution.
3. **Possible Moves:**
   * For each possible move, represented by the tuples **(x, y)** where **x** is the new state of the 4-gallon jug, and **y** is the new state of the 3-gallon jug:
     + **(4, jug\_3)**: Fill the 4-gallon jug.
     + **(jug\_4, 3)**: Fill the 3-gallon jug.
     + **(0, jug\_3)**: Empty the 4-gallon jug.
     + **(jug\_4, 0)**: Empty the 3-gallon jug.
     + **(jug\_4 - min(jug\_4, 3 - jug\_3), jug\_3 + min(jug\_4, 3 - jug\_3))**: Pour water from the 4-gallon jug to the 3-gallon jug.
     + **(jug\_4 + min(jug\_3, 4 - jug\_4), jug\_3 - min(jug\_3, 4 - jug\_4))**: Pour water from the 3-gallon jug to the 4-gallon jug.
     + Check if the new state is within the valid range and has not been visited before. If so, enqueue the new state and update the actions.
4. **Print Solution or No Solution:**
   * If a solution is found, print the steps and actions taken to achieve exactly 2 gallons in the 4-gallon jug.
   * If no solution is found, print a message indicating that there is no solution.

**Code:**

def water\_jug\_problem():

visited = set()

queue = [(0, 0, [])]

while queue:

jug\_4, jug\_3, actions = queue.pop(0)

if jug\_4 == 2:

return actions

visited.add((jug\_4, jug\_3))

for x, y in [(4, jug\_3), (jug\_4, 3), (0, jug\_3), (jug\_4, 0), (jug\_4 - min(jug\_4, 3 - jug\_3), jug\_3 + min(jug\_4, 3 - jug\_3)), (jug\_4 + min(jug\_3, 4 - jug\_4), jug\_3 - min(jug\_3, 4 - jug\_4))]:

if 0 <= x <= 4 and 0 <= y <= 3 and (x, y) not in visited:

queue.append((x, y, actions + [(x, y)]))

return None

solution = water\_jug\_problem()

if solution:

print("Steps to get exactly 2 gallons of water into the 4-gallon jug:")

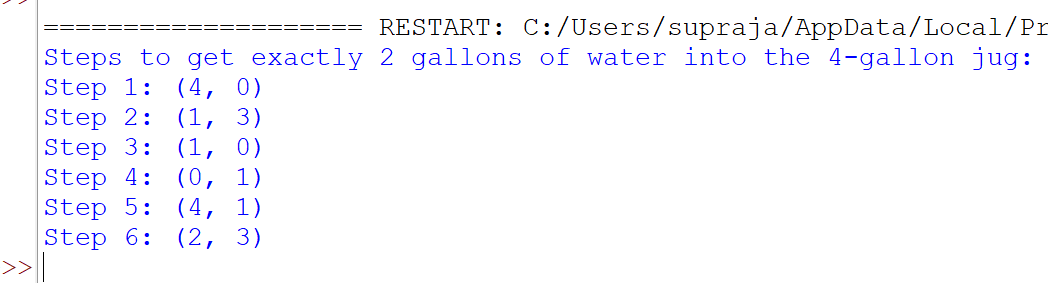
for step, action in enumerate(solution):

print(f"Step {step + 1}: {action}")

else:

print("No solution found.")

output:



Result:

Hence the program has been successfully executed and verified.