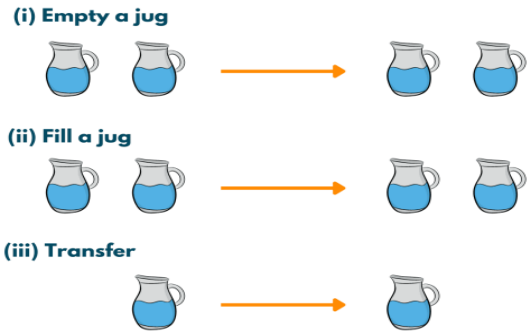
Program 5

Write a Program to Implement Water-Jug problem using Python.



Problem Description:

The Water Jug problem is a classic problem in computer science and artificial intelligence. It involves two jugs with different capacities and the task is to measure a specific amount of water using these two jugs. The challenge is to use a series of allowed operations to achieve this goal.

Step-by-Step Demonstration with BFS:

Let's continue with the Breadth-First Search (BFS) approach to solving the Water Jug Problem. In this example, we have a 3-liter jug and a 5-liter jug, and we want to measure exactly 4 liters of water. We'll use BFS to find the optimal solution. You are allowed to perform the following operations:

- **Fill a jug**: You can fill either jug to its full capacity.
- **Empty a jug**: You can empty either jug.
- **Pour water from one jug to another**: You can pour water from one jug to the other until one of the jugs is either full or empty.

1. **Start with an Empty State**: (0, 0)
   Both jugs are initially empty.

2. **Apply All Possible Actions from the Current State: (0, 0)**
   Fill the 3-liter jug: (3, 0)
   Fill the 5-liter jug: (0, 5)

3. **Expand to the Next Level**:
   We now have two new states to explore, (3, 0) and (0, 5).

4. **Continue Expanding**:

   From (3, 0), we can:

   Pour from the 3-liter jug to the 5-liter jug: (0, 3)

Fill the 3-liter jug again: (3, 3)

From (0, 5), we can:

Pour from the 5-liter jug to the 3-liter jug: (3, 2)

Empty the 5-liter jug: (0, 0)

5. **Explore Further:**

Continue expanding the states at the next level:

From (0, 3), we can pour to reach (3, 0).

From (3, 3), we can reach (0, 3) or (3, 5).

6. **Goal State Achieved:**

In our search, we've reached the goal state (0, 4).

7. **Backtrack to Find the Solution:**

To find the solution path, we backtrack from the goal state to the initial state:

(0, 4) -> (3, 1) -> (0, 1) -> (1, 0) -> (1, 5) -> (3, 4) -> (0, 4)

This step-by-step demonstration shows how Breadth-First Search systematically explores the state space to find the optimal solution to the Water Jug Problem. It ensures that we find the shortest path to the goal state while examining all possible actions. While BFS guarantees optimality, it may not always be the most efficient choice for larger problem spaces.

SOURCE CODE :

```
 # jug1 and jug2 contain the value
jug1, jug2, goal = 3, 5, 4

# Initialize a 2D list for visited states
# The list will have dimensions (jug1+1) x (jug2+1) to cover all possible
states
visited = [[False for _ in range(jug2 + 1)] for _ in range(jug1 + 1)]

def waterJug(vol1, vol2):
      # Check if we reached the goal state
      if (vol1 == goal and vol2 == 0) or (vol2 == goal and vol1 == 0):
            print(vol1,"\t", vol2)
            print("Solution Found")
            return True

    # If this state has been visited, return False
      if visited[vol1][vol2]:
```

```
            return False
        # Mark this state as visited
        visited[vol1][vol2] = True
        # Print the current state
        print(vol1,"\t", vol2)
        # Try all possible moves:
        return (
          waterJug(0, vol2) or  # Empty jug1
          waterJug(vol1, 0) or  # Empty jug2
          waterJug(jug1, vol2) or  # Fill jug1
          waterJug(vol1, jug2) or  # Fill jug2
          waterJug(vol1 + min(vol2, (jug1 - vol1)), vol2 - min(vol2, (jug1 -
vol1))) or  # Pour water from jug2 to jug1
          waterJug(vol1 - min(vol1, (jug2 - vol2)), vol2 + min(vol1, (jug2 -
vol2)))  # Pour water from jug1 to jug2
      )

print("Steps: ")
print("Jug1 \t Jug2 ")
print("----- \t ------")
waterJug(0, 0)
```

OUTPUT :

```
Steps:
Jug1    Jug2
-----   ------
0       0
4       0
4       3
0       3
3       0
3       3
4       2
0       2
Solution Found
```