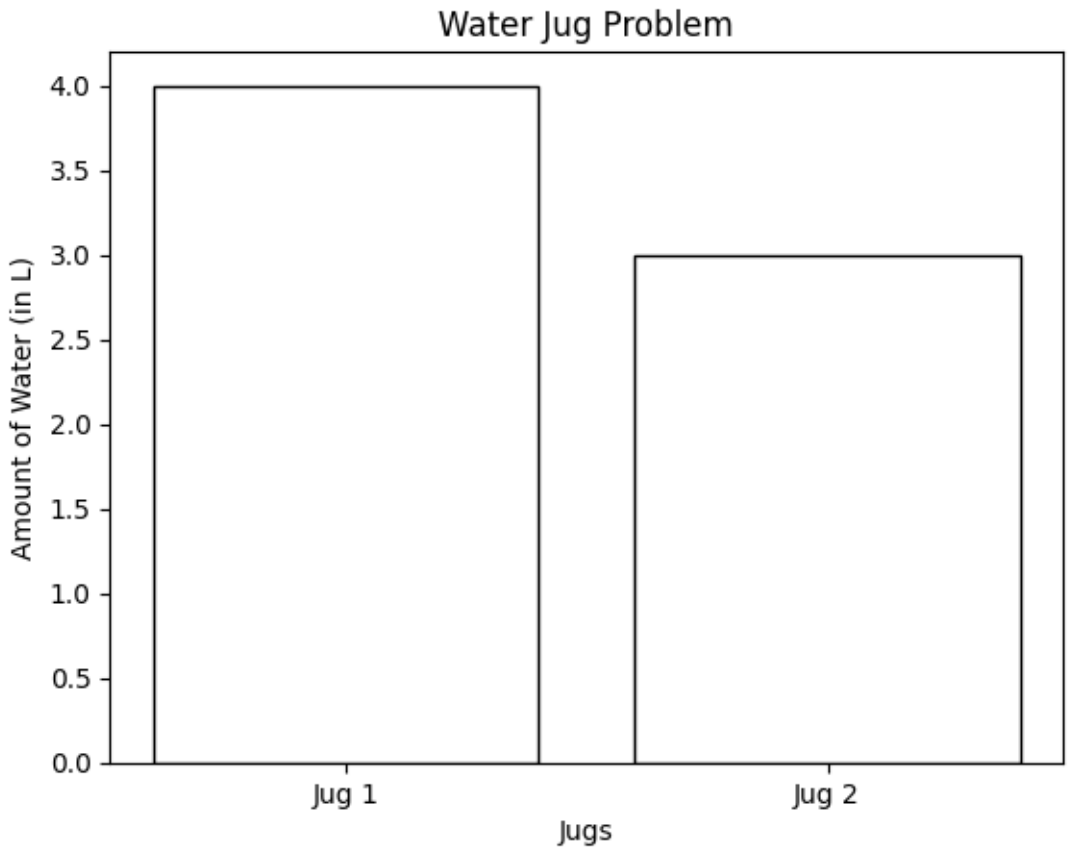
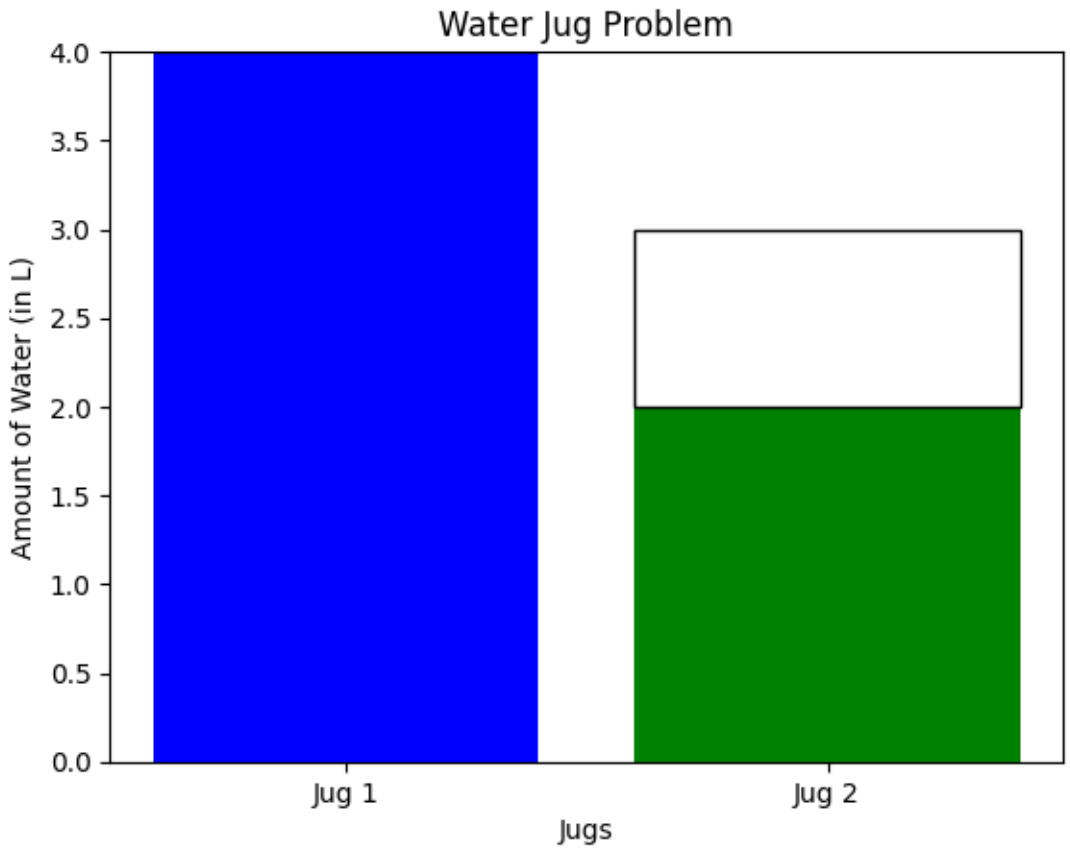
Exp 2: Implementation of Depth First Search and Breath First Search for Water Jug problem.

Program: Using DFS

```
import matplotlib.pyplot as plt
import numpy as np

def jug_diagram_visualize(a, b, jug1, jug2):
    finalx = jug1 - a
    finaly = jug2 - b
    key = ['Jug 1', 'Jug 2']
    list1 = [a, b]
    list2 = [finalx, finaly]

    plt.bar(key, list1, color=['blue', 'green'])
    plt.bar(key, list2, bottom=list1, color=['white', 'white'],
edgecolor='black')
    plt.xlabel("Jugs")
    plt.ylabel("Amount of Water (in L)")
    plt.title("Water Jug Problem")
    plt.show()

def water_jug_solver_visualize(jug1, jug2, goal):
    visited = set()
    stack = [(0, 0)]

    while stack:
        current_state = stack.pop()
        jug_diagram_visualize(current_state[0], current_state[1], jug1,
jug2)

        if current_state[0] == goal or current_state[1] == goal:
            print("Goal achieved!")
            break

        visited.add(current_state)

        next_states = [
            (jug1, current_state[1]),  # Fill Jug1
            (current_state[0], jug2),  # Fill Jug2
            (0, current_state[1]),  # Empty Jug1
            (current_state[0], 0),  # Empty Jug2
            (max(0, current_state[0] - (jug2 - current_state[1])),
min(jug2, current_state[1] + current_state[0])),  # Pour Jug1 to Jug2
            (min(jug1, current_state[0] + current_state[1]), max(0,
current_state[1] - (jug1 - current_state[0])))  # Pour Jug2 to Jug1
        ]
```

```
        for state in next_states:
            if state not in visited:
                stack.append(state)

if __name__ == "__main__":
    try:
        jug1_capacity = int(input("Enter capacity of Jug 1: "))
        jug2_capacity = int(input("Enter capacity of Jug 2: "))
        goal_amount = int(input("Enter the desired amount to measure:
"))

        if jug1_capacity <= 0 or jug2_capacity <= 0 or goal_amount < 0:
            raise ValueError("Capacity and goal must be positive
integers.")

        print("\nSteps:")
        water_jug_solver_visualize(jug1_capacity, jug2_capacity,
goal_amount)

    except ValueError as e:
        print(f"Error: {e}. Please enter valid inputs.")
```

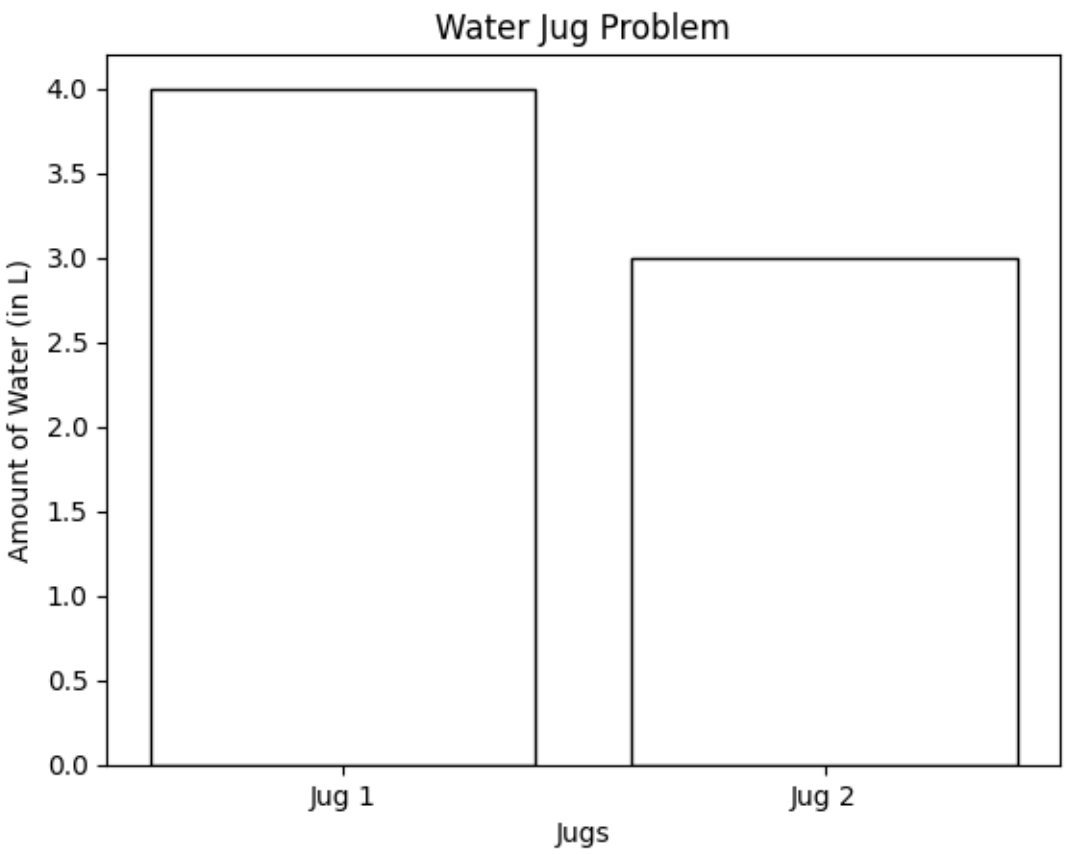


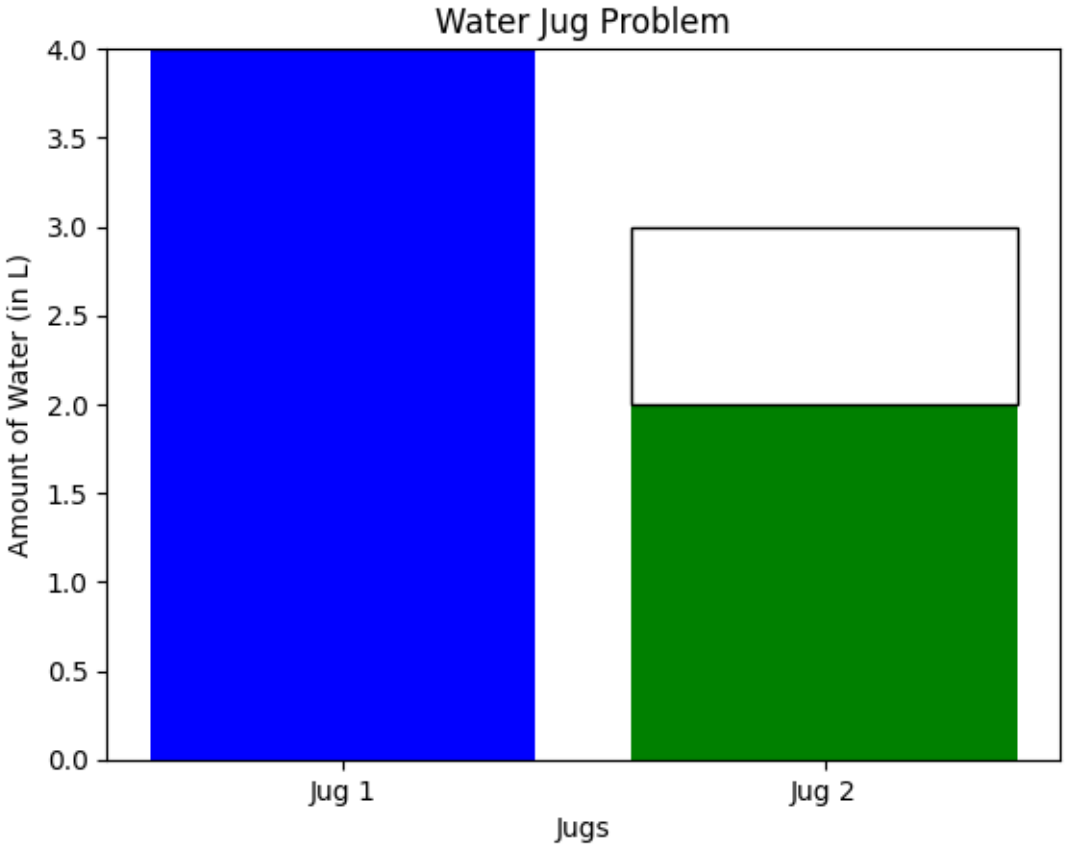


Program: Using BFS

```
from collections import deque
import matplotlib.pyplot as plt

def jug_diagram_visualize(a, b, jug1, jug2):
    finalx = jug1 - a
    finaly = jug2 - b
    key = ['Jug 1', 'Jug 2']
    list1 = [a, b]
    list2 = [finalx, finaly]

    plt.bar(key, list1, color=['blue', 'green'])
    plt.bar(key, list2, bottom=list1, color=['white', 'white'],
edgecolor='black')
    plt.xlabel("Jugs")
    plt.ylabel("Amount of Water (in L)")
    plt.title("Water Jug Problem")
    plt.show()

def water_jug_solver_visualize_bfs(jug1, jug2, goal):
    visited = set()
```

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

    while queue:
        current_state = queue.popleft()
        jug_diagram_visualize(current_state[0], current_state[1], jug1,
jug2)

        if current_state[0] == goal or current_state[1] == goal:
            print("Goal achieved!")
            break

        visited.add(current_state)

        next_states = [
            (jug1, current_state[1]),  # Fill Jug1
            (current_state[0], jug2),  # Fill Jug2
            (0, current_state[1]),  # Empty Jug1
            (current_state[0], 0),  # Empty Jug2
            (max(0, current_state[0] - (jug2 - current_state[1])),
min(jug2, current_state[1] + current_state[0])),  # Pour Jug1 to Jug2
            (min(jug1, current_state[0] + current_state[1]), max(0,
current_state[1] - (jug1 - current_state[0])))  # Pour Jug2 to Jug1
        ]

        for state in next_states:
            if state not in visited:
                queue.append(state)
                visited.add(state)

if __name__ == "__main__":
    try:
        jug1_capacity = int(input("Enter capacity of Jug 1: "))
        jug2_capacity = int(input("Enter capacity of Jug 2: "))
        goal_amount = int(input("Enter the desired amount to measure:
"))

        if jug1_capacity <= 0 or jug2_capacity <= 0 or goal_amount < 0:
            raise ValueError("Capacity and goal must be positive
integers.")

        print("\nSteps:")
        water_jug_solver_visualize_bfs(jug1_capacity, jug2_capacity,
goal_amount)

    except ValueError as e:
        print(f"Error: {e}. Please enter valid inputs.")
```


Water Jug Problem
Amount of Water (in L)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Jug 1
Jug 2
Jugs


Water Jug Problem
Amount of Water (in L)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Jug 1
Jug 2
Jugs

Program: Without visualization

```
from collections import deque

def water_jug_solver_bfs(jug1, jug2, goal):
    visited = set()
    queue = deque([(0, 0, 0)])  # Add a third element to represent the
steps

    while queue:
        current_state = queue.popleft()
        print(f"Jug 1: {current_state[0]}, Jug 2: {current_state[1]},
Steps: {current_state[2]}")

        if current_state[0] == goal or current_state[1] == goal:
            print("Goal achieved!")
            break

        visited.add((current_state[0], current_state[1]))

        next_states = [
            (jug1, current_state[1], current_state[2] + 1),  # Fill
Jug1
            (current_state[0], jug2, current_state[2] + 1),  # Fill
Jug2
            (0, current_state[1], current_state[2] + 1),  # Empty Jug1
            (current_state[0], 0, current_state[2] + 1),  # Empty Jug2
            (max(0, current_state[0] - (jug2 - current_state[1])),
min(jug2, current_state[1] + current_state[0]), current_state[2] +
1),  # Pour Jug1 to Jug2
            (min(jug1, current_state[0] + current_state[1]), max(0,
current_state[1] - (jug1 - current_state[0])), current_state[2] + 1)  #
Pour Jug2 to Jug1
        ]

        for state in next_states:
            if (state[0], state[1]) not in visited:
                queue.append(state)

if __name__ == "__main__":
    try:
        jug1_capacity = int(input("Enter capacity of Jug 1: "))
        jug2_capacity = int(input("Enter capacity of Jug 2: "))
        goal_amount = int(input("Enter the desired amount to measure:
"))
```

```
        if jug1_capacity <= 0 or jug2_capacity <= 0 or goal_amount < 0:
            raise ValueError("Capacity and goal must be positive
integers.")

        print("\nSteps:")
        water_jug_solver_bfs(jug1_capacity, jug2_capacity, goal_amount)

    except ValueError as e:
        print(f"Error: {e}. Please enter valid inputs.")
```

```
Enter capacity of Jug 1: 4
Enter capacity of Jug 2: 3
Enter the desired amount to measure: 2

Steps:
Jug 1: 0, Jug 2: 0, Steps: 0
Jug 1: 4, Jug 2: 0, Steps: 1
Jug 1: 0, Jug 2: 3, Steps: 1
Jug 1: 4, Jug 2: 3, Steps: 2
Jug 1: 1, Jug 2: 3, Steps: 2
Jug 1: 4, Jug 2: 3, Steps: 2
Jug 1: 3, Jug 2: 0, Steps: 2
Jug 1: 1, Jug 2: 0, Steps: 3
Jug 1: 3, Jug 2: 3, Steps: 3
Jug 1: 0, Jug 2: 1, Steps: 4
Jug 1: 4, Jug 2: 2, Steps: 4
Goal achieved!
```

Example: Water Jug Problem

Consider the following problem:

> A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in the 4-gallon jug?

State Representation and Initial State – we will represent a state of the problem as a tuple (x, y) where x represents the amount of water in the 4-gallon jug and y represents the amount of water in the 3-gallon jug. Note $0 \leq x \leq 4$, and $0 \leq y \leq 3$. Our initial state: (0,0)

Goal Predicate – state = (2,y) where $0 \leq y \leq 3$.

Operators – we must define a set of operators that will take us from one state to another:

1. Fill 4-gal jug	(x,y) $x < 4$	→	(4,y)
2. Fill 3-gal jug	(x,y) $y < 3$	→	(x,3)
3. Empty 4-gal jug on ground	(x,y) $x > 0$	→	(0,y)
4. Empty 3-gal jug on ground	(x,y) $y > 0$	→	(x,0)
5. Pour water from 3-gal jug to fill 4-gal jug	(x,y) $0 < x+y \geq 4$ and $y > 0$	→	(4, y - (4 - x))
6. Pour water from 4-gal jug to fill 3-gal-jug	(x,y) $0 < x+y \geq 3$ and $x > 0$	→	(x - (3-y), 3)
7. Pour all of water from 3-gal jug into 4-gal jug	(x,y) $0 < x+y \leq 4$ and $y \geq 0$	→	(x+y, 0)
8. Pour all of water from 4-gal jug into 3-gal jug	(x,y) $0 < x+y \leq 3$ and $x \geq 0$	→	(0, x+y)

Through Graph Search, the following solution is found :

Gals in 4-gal jug	**Gals in 3-gal jug**	**Rule Applied**
0	0	
		1. Fill 4
4	0	
		6. Pour 4 into 3 to fill
1	3	
		4. Empty 3
1	0	
		8. Pour all of 4 into 3
0	1	
		1. Fill 4
4	1	
		6. Pour into 3
2	3	