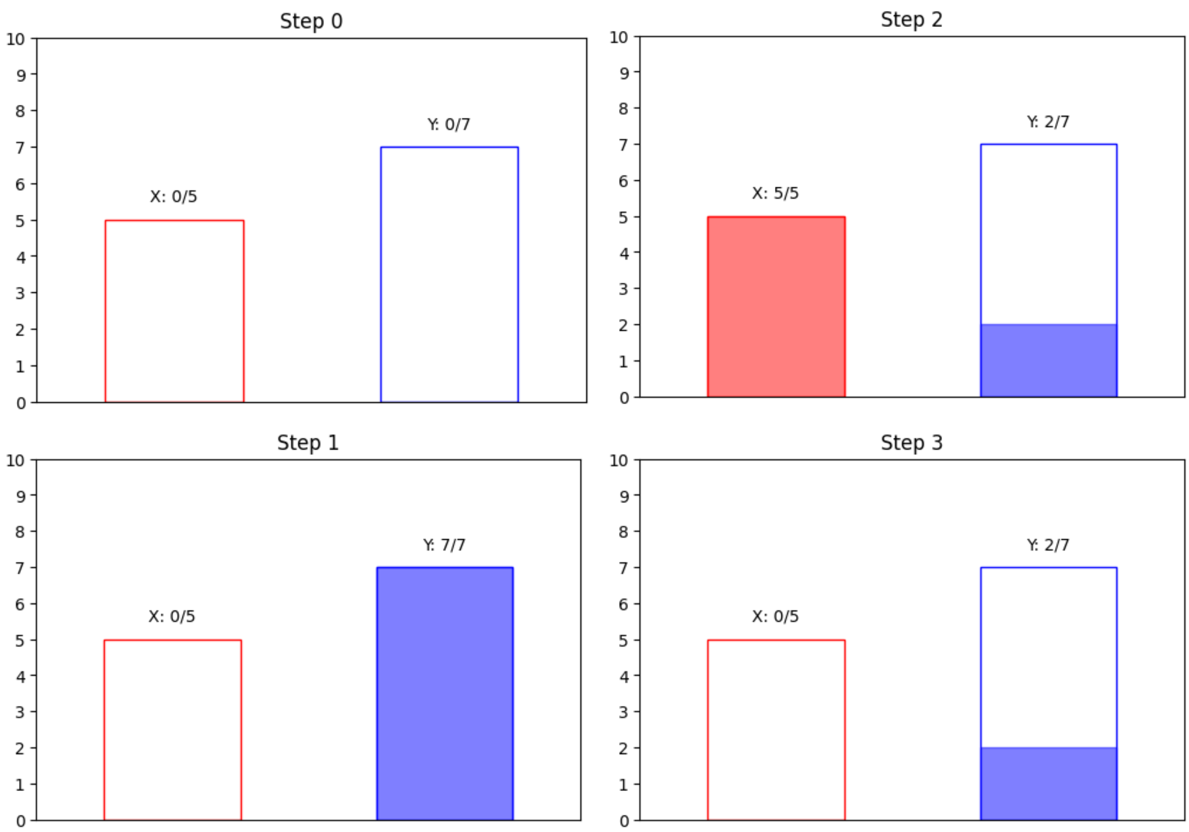
Exp 1: Implementation of Breadth First Search & Depth First Search for Water Jug Problem

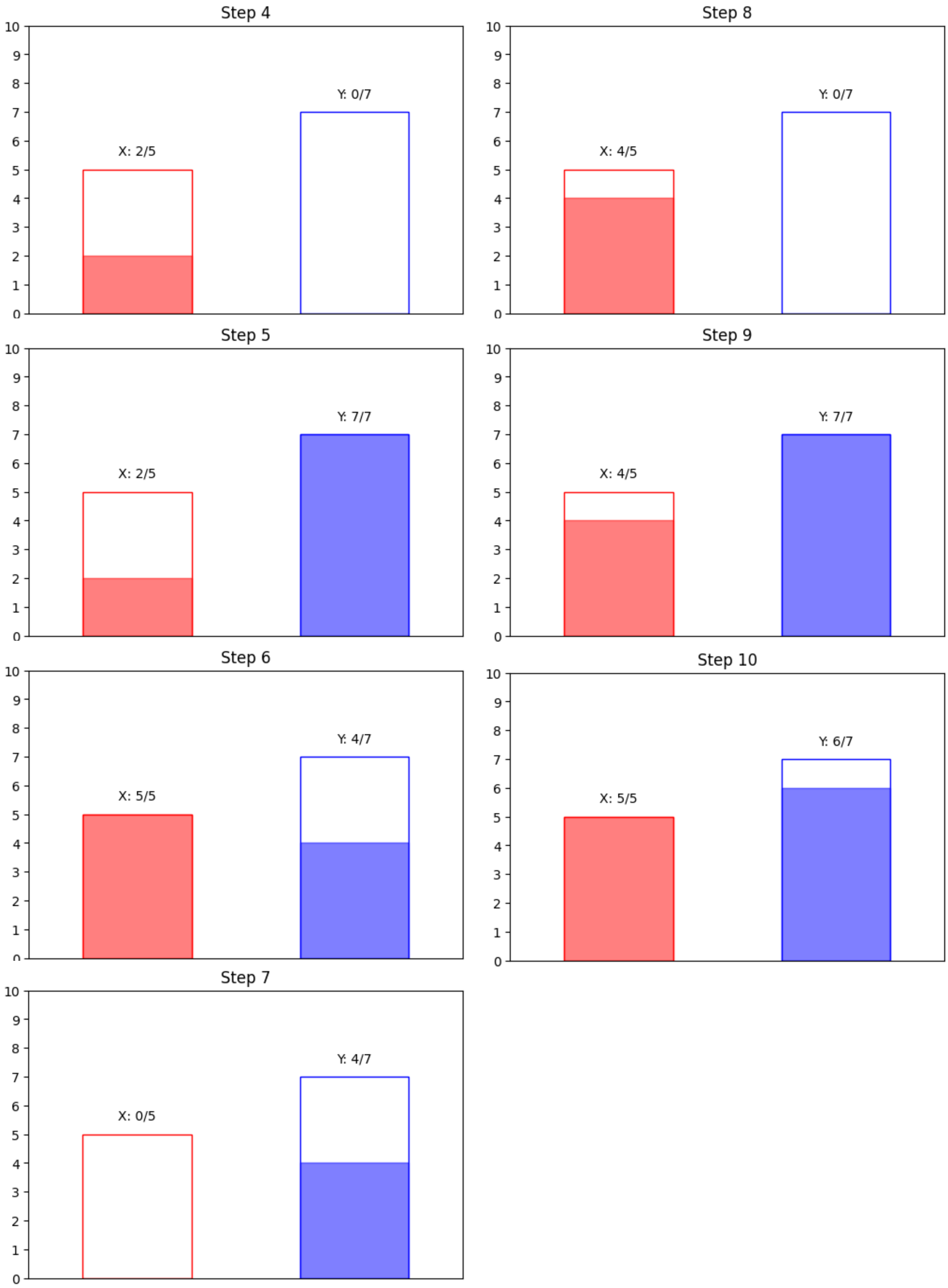
Aim: To determine a sequence of actions to measure a specific target amount of water using two jugs of given capacities. The challenge is to use only filling, emptying, and transferring operations between the jugs to achieve the target volume.

Program: Using BFS

```
import matplotlib.pyplot as plt

from matplotlib.patches import Rectangle
from collections import deque
%matplotlib inline

def solve(capacity_x, capacity_y, target):
    queue = deque([(0, 0, [])])
    visited = set()

    while queue:
        x, y, path = queue.popleft()

        if x == target or y == target:
            return path + [(x, y)]

        if (x, y) in visited:
            continue
        visited.add((x, y))

        next_states = [
            (capacity_x, y), (x, capacity_y),
            (0, y), (x, 0),
            (min(x + y, capacity_x), max(0, x + y - capacity_x)),
            (max(0, x + y - capacity_y), min(x + y, capacity_y))
        ]

        for next_x, next_y in next_states:
            queue.append((next_x, next_y, path + [(x, y)]))

    return None

def visualize_step(capacity_x, capacity_y, x, y, step):
    fig, ax = plt.subplots(figsize=(6, 4))
    ax.set_xlim(0, 2)
    ax.set_ylim(0, 10)
    ax.set_xticks([])
    ax.set_yticks(range(0, 11))

    # Draw jug X
```

```
    ax.add_patch(Rectangle((0.25, 0), 0.5, capacity_x, fill=False, color='red'))
    ax.add_patch(Rectangle((0.25, 0), 0.5, x, fill=True, color='red', alpha=0.5))

    # Draw jug Y
    ax.add_patch(Rectangle((1.25, 0), 0.5, capacity_y, fill=False, color='blue'))
    ax.add_patch(Rectangle((1.25, 0), 0.5, y, fill=True, color='blue', alpha=0.5))

    ax.text(0.5, capacity_x + 0.5, f"X: {x}/{capacity_x}", ha='center')
    ax.text(1.5, capacity_y + 0.5, f"Y: {y}/{capacity_y}", ha='center')
    ax.set_title(f"Step {step}")

    plt.show()

# Set up the problem
capacity_x, capacity_y, target = 5, 7, 6
# Solve the problem
solution = solve(capacity_x, capacity_y, target)

if solution:
    print(f"Solution found in {len(solution) - 1} steps:")
    for step, (x, y) in enumerate(solution):
        print(f"Step {step}: X: {x}, Y: {y}")
        visualize_step(capacity_x, capacity_y, x, y, step)
else:
    print("No solution found.")
```





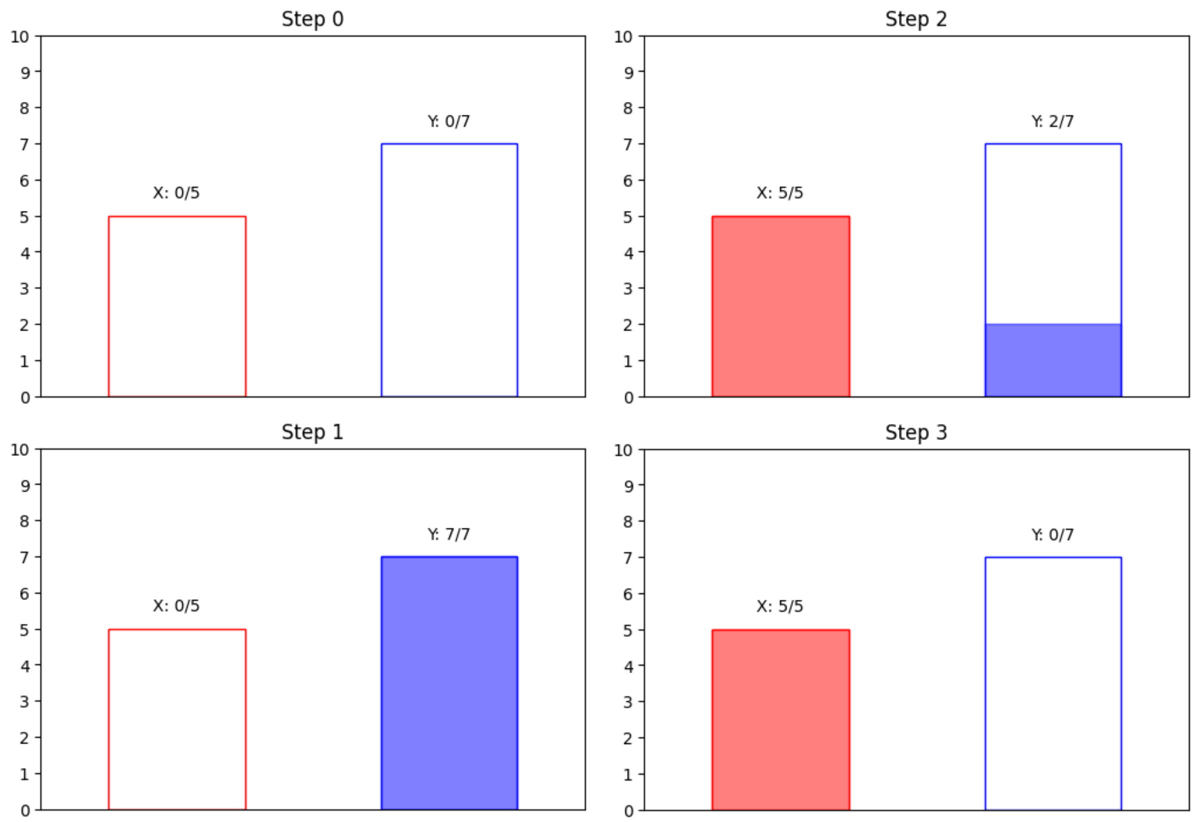
Step 4
X: 2/5
Y: 0/7
Step 8
X: 4/5
Y: 0/7
Step 5
X: 2/5
Y: 7/7
Step 9
X: 4/5
Y: 7/7
Step 6
X: 5/5
Y: 4/7
Step 10
X: 5/5
Y: 6/7
Step 7
X: 0/5
Y: 4/7

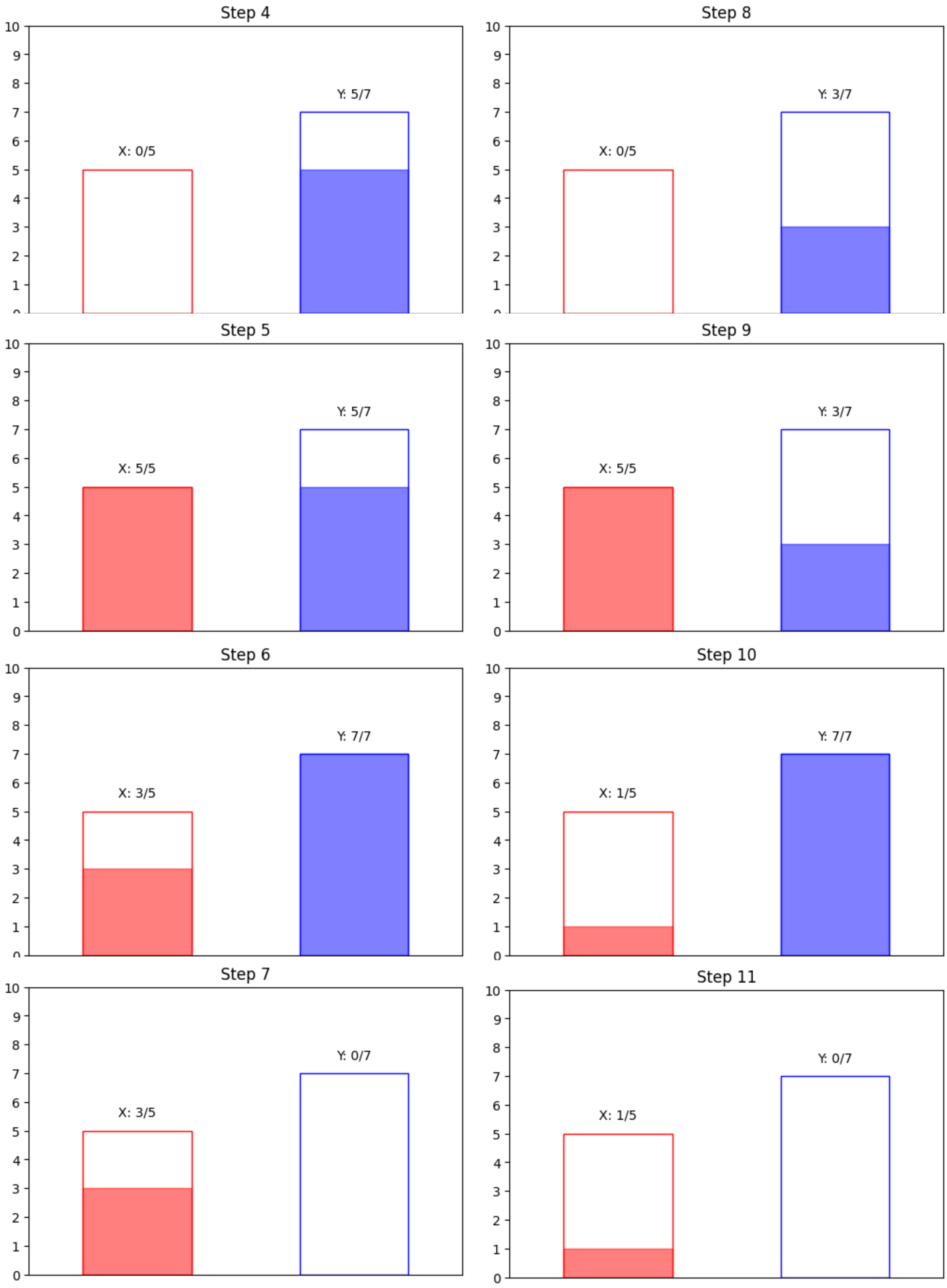
```
Let Jug X = P where 0 <= P <= 5
    Jug Y = Q where 0 <= Q <= 7

Initial State: (P,Q) = (0,0)

Production Rules (BFS Order):
1. (P,Q) -> (5, Q)     ; Fill jug X
2. (P,Q) -> (P, 7)     ; Fill jug Y
3. (P,Q) -> (0, Q)     ; Empty jug X
4. (P,Q) -> (P, 0)     ; Empty jug Y
5. (P,Q) -> (min(P+Q, 5), max(0, P+Q-5))  ; Pour Y into X
6. (P,Q) -> (max(0, P+Q-7), min(P+Q, 7))  ; Pour X into Y
7. (P,Q) -> (P, Q)     ; Do nothing (terminal state check)

Goal State: P = 6 or Q = 6
```

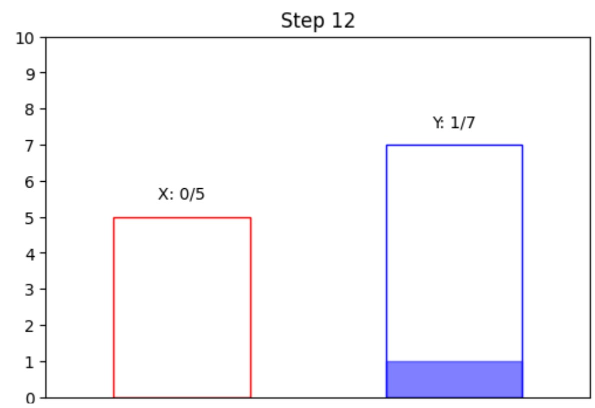
Program: Using DFS

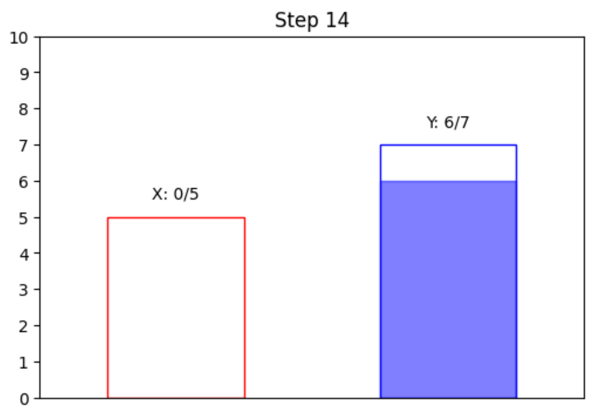
```
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle

def solve_dfs(capacity_x, capacity_y, target):
    stack = [(0, 0, [])]
    visited = set()

    while stack:
        x, y, path = stack.pop()

        if x == target or y == target:
            return path + [(x, y)]

        if (x, y) in visited:
            continue
        visited.add((x, y))

        next_states = [
            (capacity_x, y), (x, capacity_y),
            (0, y), (x, 0),
            (min(x + y, capacity_x), max(0, x + y - capacity_x)),
            (max(0, x + y - capacity_y), min(x + y, capacity_y))
        ]

        for next_x, next_y in next_states:
            stack.append((next_x, next_y, path + [(x, y)]))

    return None

def visualize_step(capacity_x, capacity_y, x, y, step):
    fig, ax = plt.subplots(figsize=(6, 4))
    ax.set_xlim(0, 2)
    ax.set_ylim(0, 10)
    ax.set_xticks([])
    ax.set_yticks(range(0, 11))

    # Draw jug X
    ax.add_patch(Rectangle((0.25, 0), 0.5, capacity_x, fill=False,
color='red'))
    ax.add_patch(Rectangle((0.25, 0), 0.5, x, fill=True, color='red',
alpha=0.5))

    # Draw jug Y
    ax.add_patch(Rectangle((1.25, 0), 0.5, capacity_y, fill=False,
color='blue'))
    ax.add_patch(Rectangle((1.25, 0), 0.5, y, fill=True, color='blue',
alpha=0.5))
```

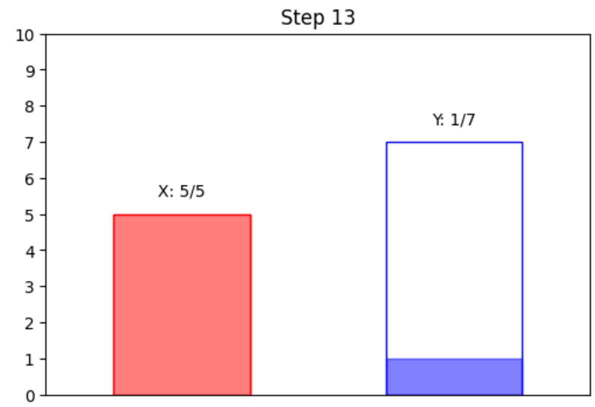
```
    ax.text(0.5, capacity_x + 0.5, f"X: {x}/{capacity_x}", ha='center')
    ax.text(1.5, capacity_y + 0.5, f"Y: {y}/{capacity_y}", ha='center')
    ax.set_title(f"Step {step}")

    plt.show()

# Set up the problem
capacity_x, capacity_y, target = 5, 7, 6
# Solve the problem using DFS
solution = solve_dfs(capacity_x, capacity_y, target)

if solution:
    print(f"Solution found in {len(solution) - 1} steps:")
    for step, (x, y) in enumerate(solution):
        print(f"Step {step}: X: {x}, Y: {y}")
        visualize_step(capacity_x, capacity_y, x, y, step)
else:
    print("No solution found.")
```




Step 4
X: 0/5
Y: 5/7
Step 8
X: 0/5
Y: 3/7
Step 5
X: 5/5
Y: 5/7
Step 9
X: 5/5
Y: 3/7
Step 6
X: 3/5
Y: 7/7
Step 10
X: 1/5
Y: 7/7
Step 7
X: 3/5
Y: 0/7
Step 11
X: 1/5
Y: 0/7








```
Let Jug X = P where 0 <= P <= 5
    Jug Y = Q where 0 <= Q <= 7

Initial State: (P,Q) = (0,0)

Goal State: P = 6 or Q = 6

Production Rules (DFS order):
1. (P,Q) -> (5, Q)     ; Fill jug X
   If (5,Q) is goal state, terminate
   Else, apply rule 1 to (5,Q)

2. (P,Q) -> (P, 7)     ; Fill jug Y
   If (P,7) is goal state, terminate
   Else, apply rule 1 to (P,7)
```

```
3. (P,Q) -> (0, Q)     ; Empty jug X
   If (0,Q) is goal state, terminate
   Else, apply rule 1 to (0,Q)

4. (P,Q) -> (P, 0)     ; Empty jug Y
   If (P,0) is goal state, terminate
   Else, apply rule 1 to (P,0)

5. (P,Q) -> (min(P+Q, 5), max(0, P+Q-5))  ; Pour Y into X
   Let (P',Q') = (min(P+Q, 5), max(0, P+Q-5))
   If (P',Q') is goal state, terminate
   Else, apply rule 1 to (P',Q')

6. (P,Q) -> (max(0, P+Q-7), min(P+Q, 7))  ; Pour X into Y
   Let (P',Q') = (max(0, P+Q-7), min(P+Q, 7))
   If (P',Q') is goal state, terminate
   Else, apply rule 1 to (P',Q')

7. If all rules have been applied and backtracked to initial state,
terminate with no solution.
```