MANIPAL
UNIVERSITY JAIPUR
(University under Section 2(f) of the UGC Act)

School of Computer Science and Engineering

Department

of

Computer Science and Engineering

Artificial Intelligence & Soft Computing Lab

CS3131

Name	: CHARU RAMNANI
Registration No.	: 229302213
Semester	: 5
Section	: CSE-D
Faculty Instructor Name	: MRS. BALI DEVI GUPTA

Content

S. No.	List of Programs	Remarks	Date	Signature
1	Write production rules to solve Water jug problem and basic introduction.			
2	Program to implement best first search and depth first search.			
3	Program to implement the concept of hill climbing.			
4	Program to Implement A* Algorithm.			
5	Write a program to implement Perceptron neural network.			
6	Write a program to implement Perceptron neural network for Classification/ Regression.			
7	Write a program to implement Backpropagation neural network.			
8	Write a program to implement Backpropagation neural network for Classification/ Regression.			
9	Implementation and execution of allotted research-based problems			
10	Implementation and execution of allotted research-based problems			
11	Implement Fuzzy operations Union, Intersections, Complement.			
12	Write a Program to implement the concept of Genetic Algorithm			

LAB1- WATER JUG (RULE BASED) PROBLEM

AIM - Write production rules to solve Water jug problem and basic introduction.

My Problem Statement- Jug1=13L, Jug2=11L capacity. How can you get exactly 8 gallons of water in the 13-gallon jug?

Production Rule:

State Representation and Initial State:

- State Representation: Each state is represented as a tuple (x, y), where:
  - x represents the amount of water in Jug 1 (with a capacity of 13 liters).
  - y represents the amount of water in Jug 2 (with a capacity of 11 liters).
- Initial State: The problem starts with both jugs empty: (0, 0).
- Final State: The problem ends with both jug: (8, 0)

Operators:

1. **Fill Jug 1:**
   - Action: Fill Jug 1 to its full capacity.
   - State Transition: From (x,y) to (13,y) where $x<13$.
2. **Fill Jug 2:**
   - Action: Fill Jug 2 to its full capacity.
   - State Transition: From (x,y) to (x,11) where $y<11$.
3. **Empty Jug 1:**
   - Action: Empty Jug 1 completely.
   - State Transition: From (x,y) to (0,y) where $x>0$.
4. **Empty Jug 2:**
   - Action: Empty Jug 2 completely.
   - State Transition: From (x,y) to (x,0) where $y>0$.
5. **Pour Water from Jug 1 to Jug 2:**
   - Action: Pour water from Jug 1 into Jug 2 until Jug 2 is full or Jug 1 is empty.
   - State Transition: From (x,y) to $(x-\min(x,11-y), y+\min(x,11-y))$ where $x>0$ and $y<11$.
6. **Pour Water from Jug 2 to Jug 1:**
   - Action: Pour water from Jug 2 into Jug 1 until Jug 1 is full or Jug 2 is empty.

- State Transition: From (x,y) to (x+min(y,13−x),y−min(y,13−x)) where y>0 and x<13.

Solution Path:

Here is the sequence of steps (states) to obtain exactly 8 liters in the 13-liter jug:

1. (0, 0): Initial state (both jugs empty).
2. (13, 0): Fill Jug 1 to full capacity.
3. (2, 11): Pour water from Jug 1 into Jug 2 until Jug 2 is full (Jug 2 is now full with 11 liters, and 2 liters remain in Jug 1).
4. (2, 0): Empty Jug 2.
5. (0, 2): Pour water from Jug 1 into Jug 2 (Jug 2 now has 2 liters, and Jug 1 is empty).
6. (13, 2): Fill Jug 1 to full capacity again.
7. (4, 11): Pour water from Jug 1 into Jug 2 until Jug 2 is full (4 liters remain in Jug 1, and Jug 2 is full with 11 liters).
8. (4, 0): Empty Jug 2.
9. (0, 4): Pour water from Jug 1 into Jug 2 (Jug 2 now has 4 liters, and Jug 1 is empty).
10. (13, 4): Fill Jug 1 to full capacity again.
11. (6, 11): Pour water from Jug 1 into Jug 2 until Jug 2 is full (6 liters remain in Jug 1, and Jug 2 is full with 11 liters).
12. (6, 0): Empty Jug 2.
13. (0, 6): Pour water from Jug 1 into Jug 2 (Jug 2 now has 6 liters, and Jug 1 is empty).
14. (13, 6): Fill Jug 1 to full capacity again.
15. (8, 11): Pour water from Jug 1 into Jug 2 until Jug 2 is full (8 liters remain in Jug 1, and Jug 2 is full with 11 liters).

Final State:

(8, 11): Jug 1 contains 8 liters, meeting the goal.

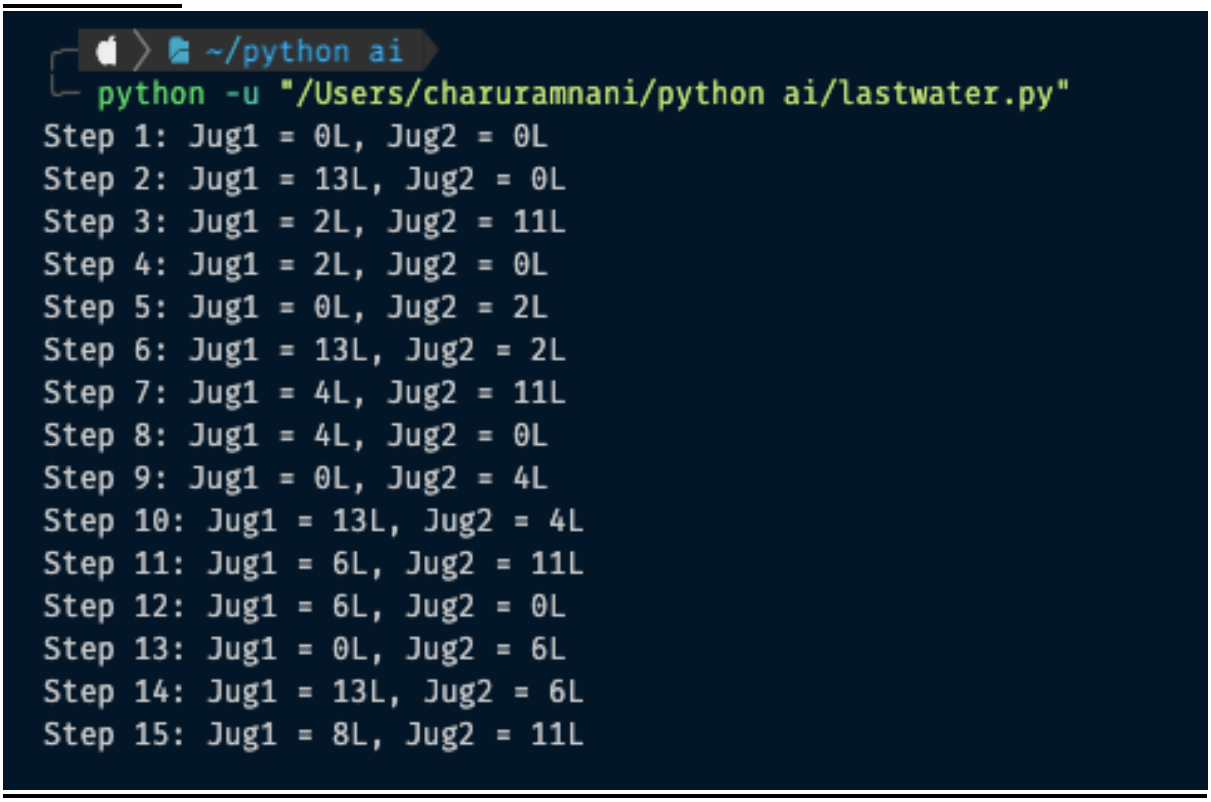
CODE FOR WATER JUG PROBLEM:

```
from collections import deque

class WaterJugProblem:
    def __init__(self, jug1_capacity, jug2_capacity, target_amount):
```

```
        self.jug1_capacity = jug1_capacity
        self.jug2_capacity = jug2_capacity
        self.target_amount = target_amount
        self.visited = set()
        self.solution = []

    def is_solved(self, state):
        return state[0] == self.target_amount or state[1] == self.target_amount

    def bfs(self):
        initial_state = (0, 0)
        queue = deque([(initial_state, [])])
        self.visited.add(initial_state)

        while queue:
            (jug1, jug2), path = queue.popleft()

            if self.is_solved((jug1, jug2)):
                self.solution = path + [(jug1, jug2)]
                return True

            next_states = [
                (self.jug1_capacity, jug2),  # Fill Jug 1
                (jug1, self.jug2_capacity),  # Fill Jug 2
                (0, jug2),  # Empty Jug 1
                (jug1, 0),  # Empty Jug 2
                (jug1 - min(jug1, self.jug2_capacity - jug2), jug2 + min(jug1, self.jug2_capacity - jug2)),  # Pour Jug 1 into Jug 2
                (jug1 + min(jug2, self.jug1_capacity - jug1), jug2 - min(jug2, self.jug1_capacity - jug1))  # Pour Jug 2 into Jug 1
            ]

            for state in next_states:
                if state not in self.visited:
                    self.visited.add(state)
                    queue.append((state, path + [(jug1, jug2)]))

        return False

    def print_solution(self):
        if not self.solution:
            print("No solution found")
            return

        for i, (jug1, jug2) in enumerate(self.solution):
```

```
        print(f"Step {i + 1}: Jug1 = {jug1}L, Jug2 = {jug2}L")

def solve_water_jug(jug1_capacity, jug2_capacity, target_amount):
    problem = WaterJugProblem(jug1_capacity, jug2_capacity, target_amount)
    if problem.bfs():
        problem.print_solution()
    else:
        print("No solution exists")

# Example Usage:
solve_water_jug(13, 11, 8)
```

OUTPUT:



CODE FOR ANIMATION:

```
import matplotlib.pyplot as plt
import matplotlib.animation as animation

class WaterJugAnimation:
    def __init__(self, jug1_capacity, jug2_capacity, required_amount):
        self.jug1_capacity = jug1_capacity
        self.jug2_capacity = jug2_capacity
        self.required_amount = required_amount
        self.steps = self.solve_jug_problem()
```

```
        self.step_number = 0

        self.fig, self.ax = plt.subplots()
        self.update_plot()
        self.fig.canvas.mpl_connect('key_press_event', self.on_key)

    def draw_jugs(self, jug1, jug2):
        self.ax.clear()
        self.ax.set_xlim(0, 10)
        self.ax.set_ylim(0, max(self.jug1_capacity, self.jug2_capacity) + 2)

        # Draw Jug 1
        self.ax.add_patch(plt.Rectangle((2, 0), 2, self.jug1_capacity, edgecolor='black',
facecolor='none'))
        self.ax.add_patch(plt.Rectangle((2, 0), 2, jug1, color='lightpink'))

        # Draw Jug 2
        self.ax.add_patch(plt.Rectangle((6, 0), 2, self.jug2_capacity, edgecolor='black',
facecolor='none'))
        self.ax.add_patch(plt.Rectangle((6, 0), 2, jug2, color='lavender'))

        # Label the jugs and step number
        self.ax.text(3, self.jug1_capacity + 0.2, f'{self.jug1_capacity}L', ha='center', va='bottom',
fontsize=12)
        self.ax.text(7, self.jug2_capacity + 0.2, f'{self.jug2_capacity}L', ha='center', va='bottom',
fontsize=12)
        self.ax.text(5, max(self.jug1_capacity, self.jug2_capacity) + 1, f'Step {self.step_number
+ 1}', ha='center', fontsize=16)

    def solve_jug_problem(self):
        steps = [(0, 0)]
        jug1, jug2 = 0, 0

        while jug1 != self.required_amount:
            if jug1 == 0:
                jug1 = self.jug1_capacity
            elif jug2 != self.jug2_capacity:
                transfer_amount = min(jug1, self.jug2_capacity - jug2)
                jug1 -= transfer_amount
                jug2 += transfer_amount
            else:
                jug2 = 0

            if (jug1, jug2) not in steps:  # Avoid adding redundant steps
                steps.append((jug1, jug2))
```

```
            if jug1 == self.required_amount:
                break

            if jug2 > 0 and (jug1, 0) not in steps:
                steps.append((jug1, 0))

        return steps

    def update_plot(self):
        jug1, jug2 = self.steps[self.step_number]
        self.draw_jugs(jug1, jug2)
        plt.draw()

    def on_key(self, event):
        if event.key == 'right':
            if self.step_number < len(self.steps) - 1:
                self.step_number += 1
                self.update_plot()
        elif event.key == 'left':
            if self.step_number > 0:
                self.step_number -= 1
                self.update_plot()

    def show(self):
        plt.show()

# Set Jug capacities and required amount
jug1_capacity = 13
jug2_capacity = 11
required_amount = 8

# Create animation instance and show
animation = WaterJugAnimation(jug1_capacity, jug2_capacity, required_amount)
animation.show()

# Print the number of steps
print(f"Total number of steps: {len(animation.steps)}")
```

OUTPUT:

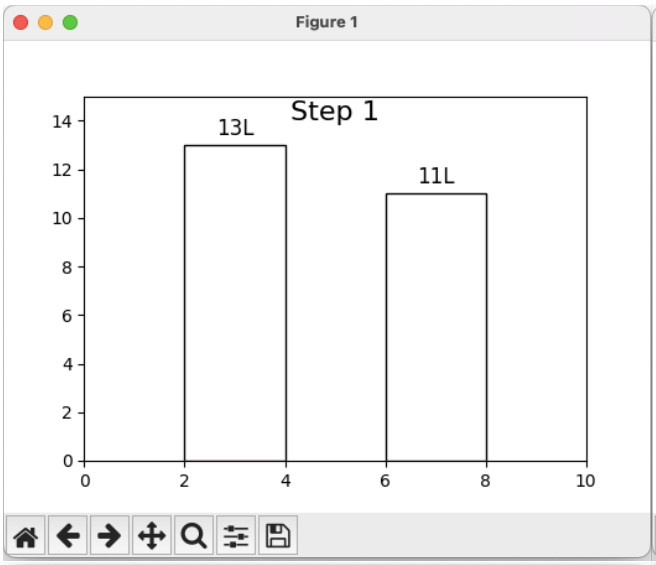


Figure 1
Step 1
13L
11L

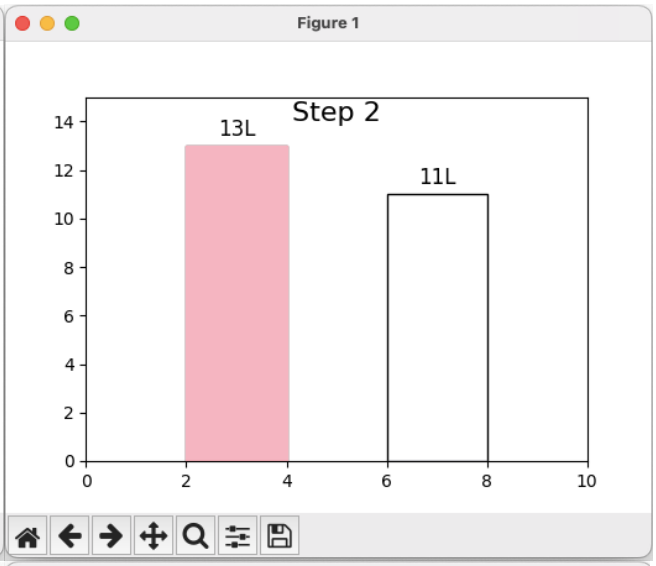


Figure 1
Step 2
13L
11L

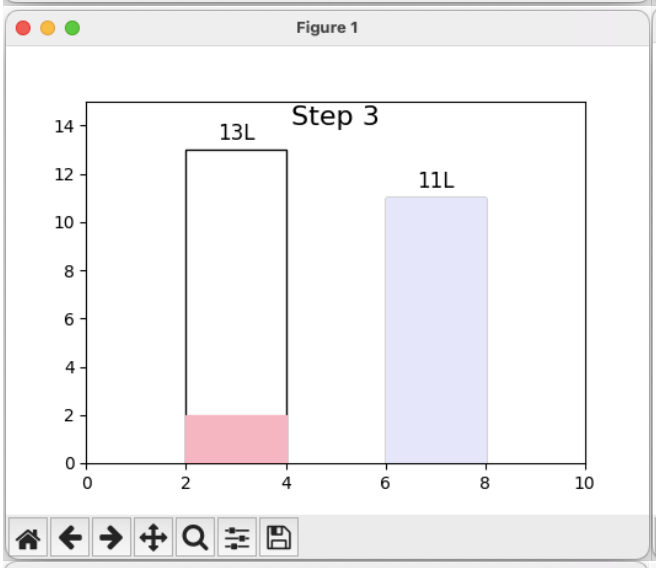


Figure 1
Step 3
13L
11L

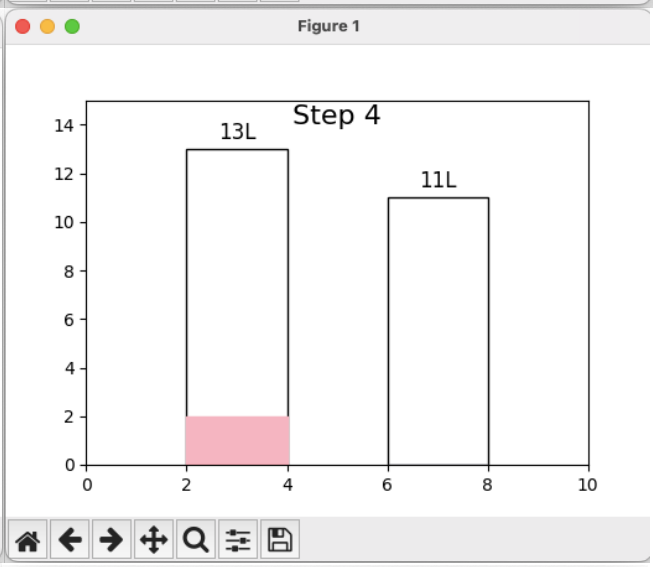


Figure 1
Step 4
13L
11L

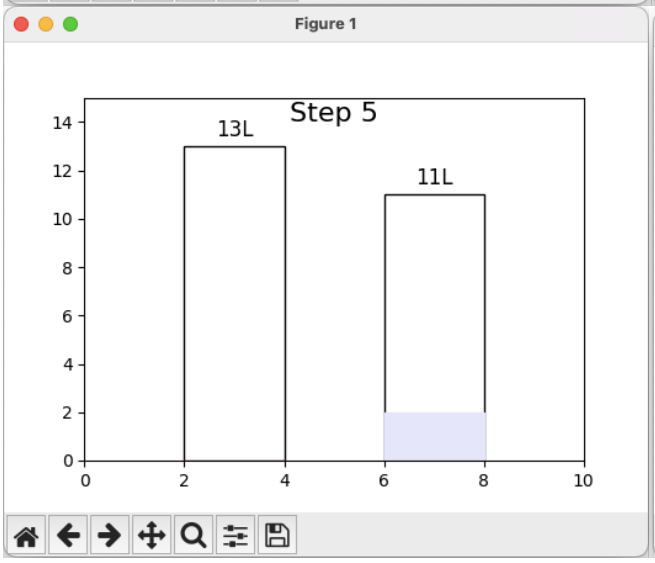


Figure 1
Step 5
13L
11L

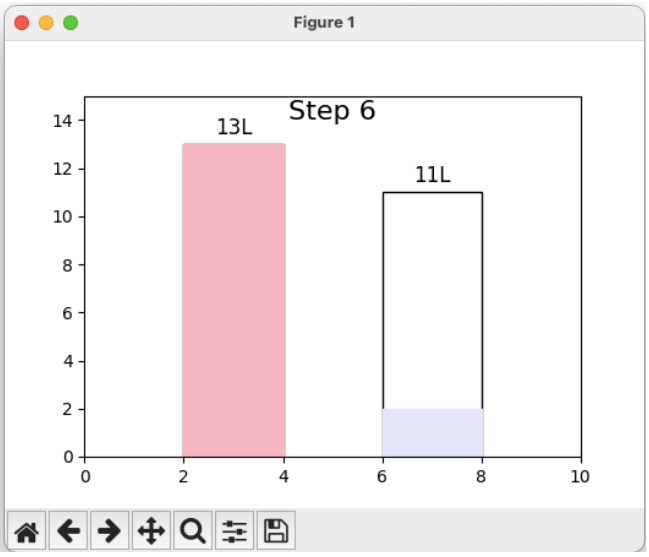


Figure 1
Step 6
13L
11L

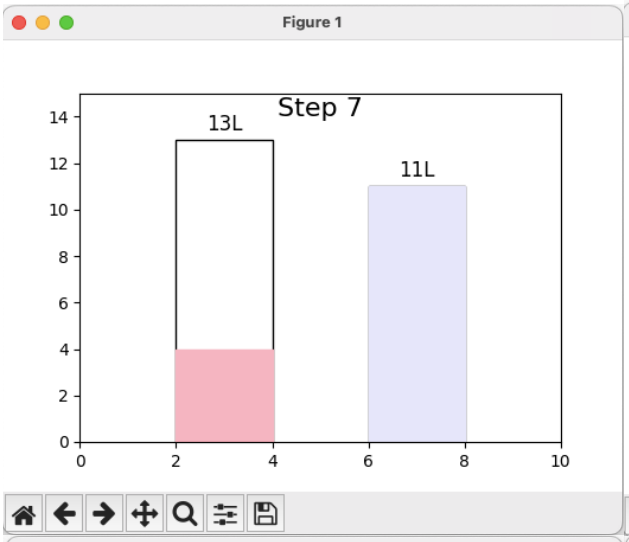


Figure 1
Step 7
13L
11L

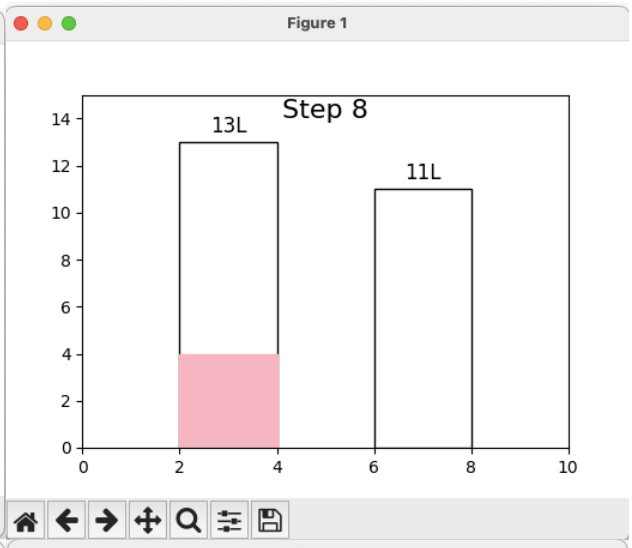


Figure 1
Step 8
13L
11L

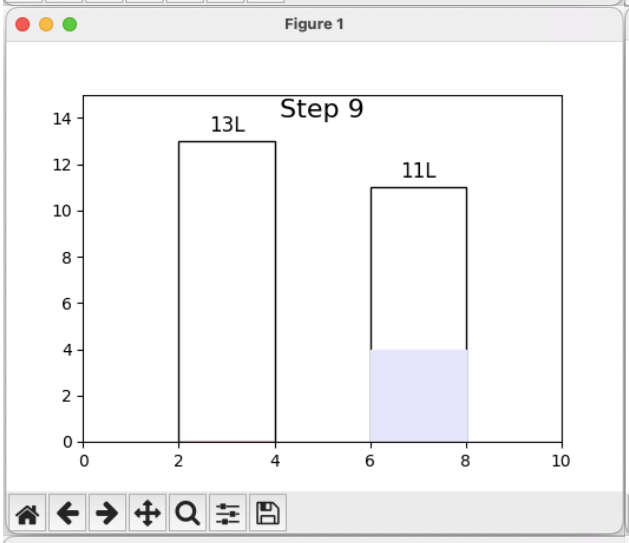


Figure 1
Step 9
13L
11L

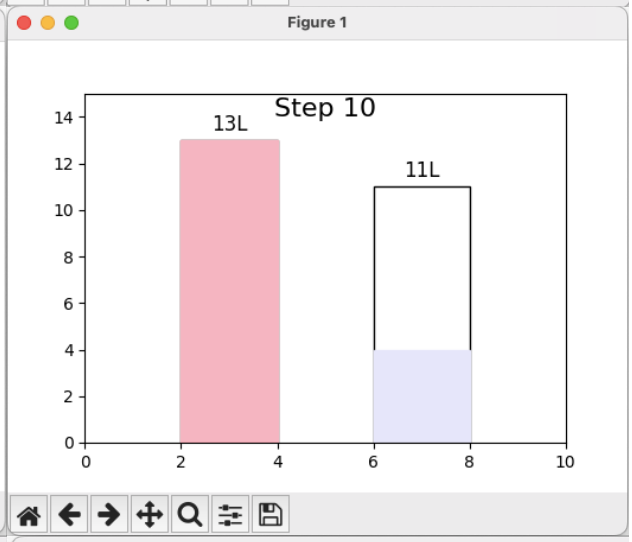


Figure 1
Step 10
13L
11L

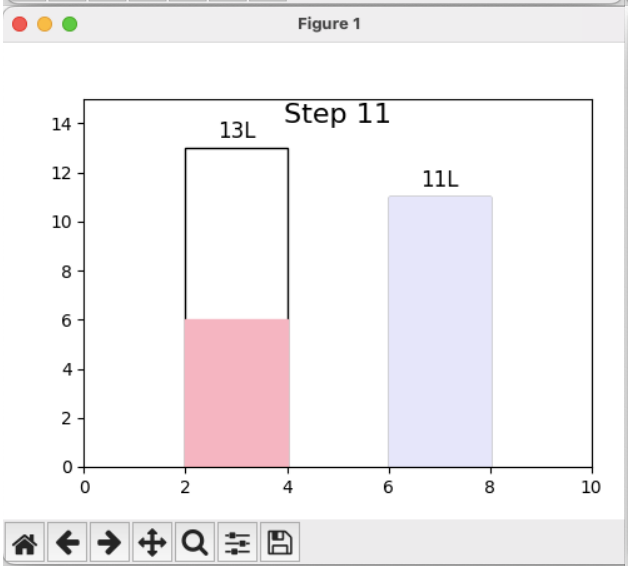


Figure 1
Step 11
13L
11L

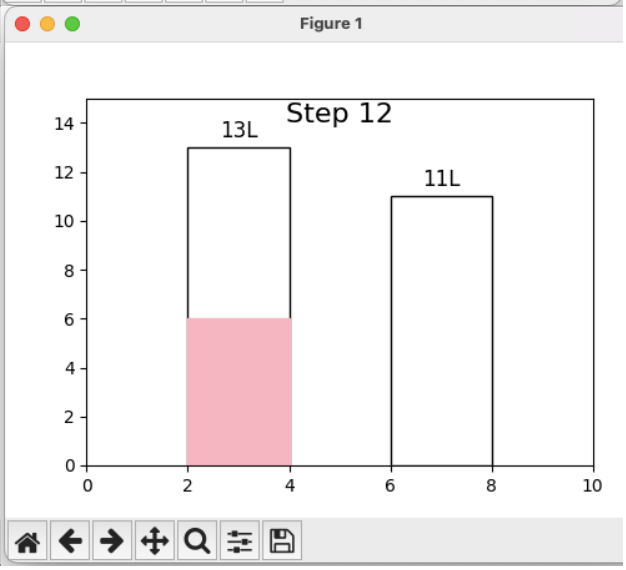


Figure 1
Step 12
13L
11L

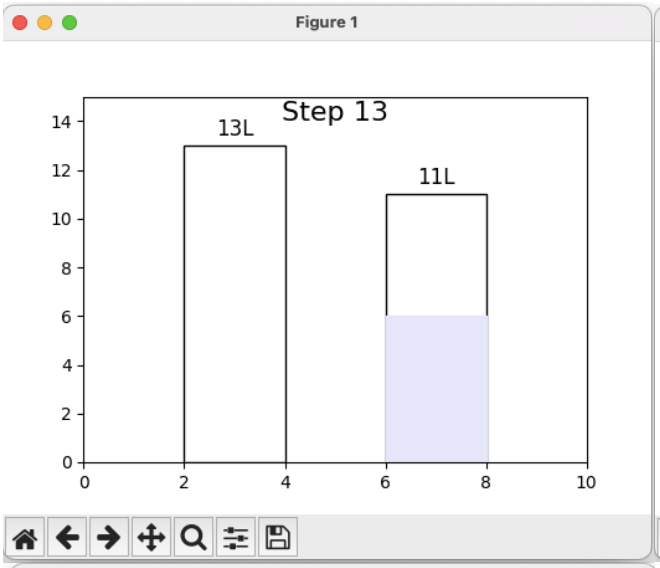


Figure 1
Step 13
13L
11L
0
2
4
6
8
10
0
2
4
6
8
10
12
14

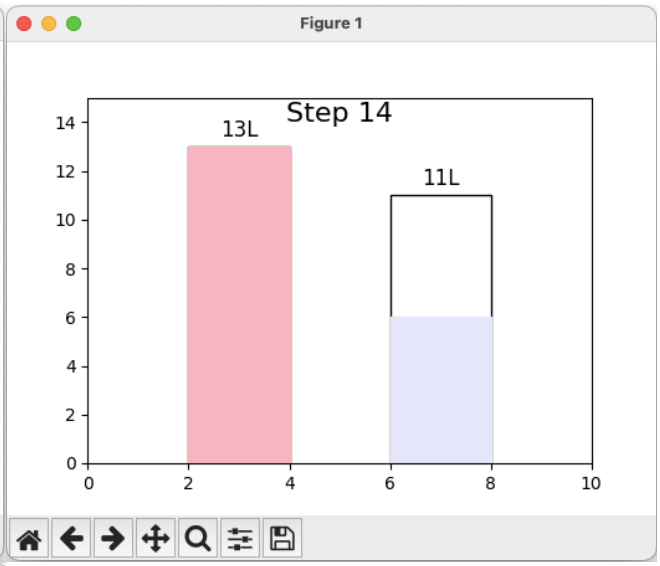


Figure 1
Step 14
13L
11L
0
2
4
6
8
10
0
2
4
6
8
10
12
14

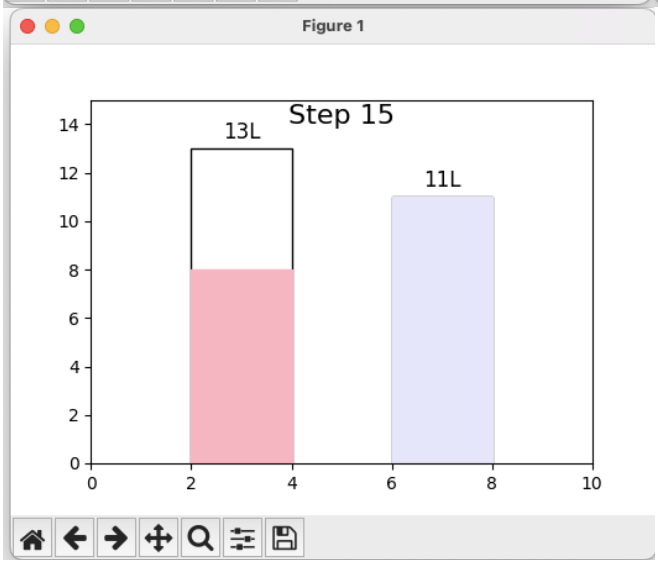


Figure 1
Step 15
13L
11L
0
2
4
6
8
10
0
2
4
6
8
10
12
14