LAB1- RULE BASED PROBLEM

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

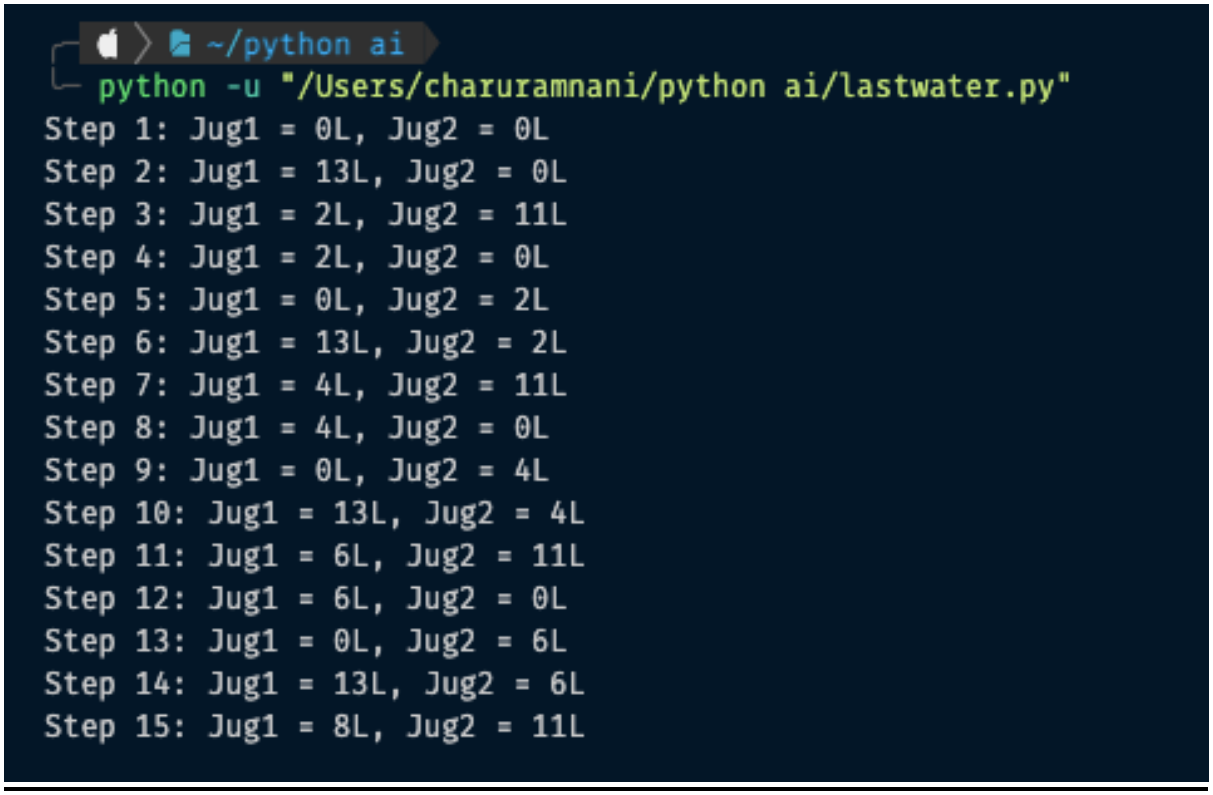
My question- Jug1=13L, Jug2=11L capacity. How can you get exactly 8 gallons of water in the 13-gallon jug?

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
import numpy as np
from matplotlib.patches import FancyBboxPatch, Rectangle

def water_jug_problem_animation(jug1_capacity, jug2_capacity, target):
    visited = set()
    queue = [(0, 0)]
    visited.add((0, 0))

    states = []  # To store all states for animation

    while queue:
        current_jug1, current_jug2 = queue.pop(0)
        states.append((current_jug1, current_jug2))

        if current_jug1 == target or current_jug2 == target:
            states.append((current_jug1, current_jug2))
```

```
            break

        possible_states = [
            (jug1_capacity, current_jug2),  # Fill Jug 1
            (current_jug1, jug2_capacity),  # Fill Jug 2
            (0, current_jug2),  # Empty Jug 1
            (current_jug1, 0),  # Empty Jug 2
            (current_jug1 - min(current_jug1, jug2_capacity - current_jug2),
current_jug2 + min(current_jug1, jug2_capacity - current_jug2)),  # Pour Jug 1
into Jug 2
            (current_jug1 + min(current_jug2, jug1_capacity - current_jug1),
current_jug2 - min(current_jug2, jug1_capacity - current_jug1)),  # Pour Jug 2
into Jug 1
        ]

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

    fig, ax = plt.subplots(figsize=(12, 8))

    def draw_jug(ax, x, y, jug_width, jug_height, water_height, color1, color2,
label, capacity, water_transfer, handle_color='grey'):
        # Jug body
        body = FancyBboxPatch((x, y), jug_width, jug_height,
boxstyle="round,pad=0.1", edgecolor='black', facecolor='none', lw=2)
        ax.add_patch(body)

        # Handle (outside the jug)
        handle = FancyBboxPatch((x + jug_width + 0.1, y + jug_height * 0.2), 0.5,
jug_height * 0.6, boxstyle="round,pad=0.2", edgecolor='black',
facecolor=handle_color, lw=2)
        ax.add_patch(handle)

        # Water in the jug
        if water_transfer:
            # Show both colors during transfer
            water1_height = water_height * (1 - water_transfer)  # Original water
            water2_height = water_height * water_transfer  # Transferred water
```

```
            ax.add_patch(Rectangle((x, y), jug_width, water1_height, color=color1,
zorder=2))
            ax.add_patch(Rectangle((x, y + water1_height), jug_width,
water2_height, color=color2, zorder=2))
        else:
            ax.add_patch(Rectangle((x, y), jug_width, water_height, color=color1,
zorder=2))

        # Capacity label
        ax.text(x + jug_width / 2, y - 0.5, f'{label}/{capacity}', ha='center',
va='bottom', fontsize=12)

    def update(frame):
        ax.clear()
        ax.set_xlim(0, 14)
        ax.set_ylim(0, 10)
        ax.set_aspect('equal')
        ax.axis('off')

        current_jug1, current_jug2 = states[frame]

        # Determine if water is being transferred
        if frame > 0:
            prev_jug1, prev_jug2 = states[frame-1]
            water_transfer = True if (prev_jug1 != current_jug1 and prev_jug2 !=
current_jug2) else False
        else:
            water_transfer = False

        # Jug 1 (Red Water)
        water_height_jug1 = (current_jug1 / jug1_capacity) * 6  # Scale water
height
        color1_jug1 = 'red'
        color2_jug1 = 'blue' if water_transfer and prev_jug2 > current_jug2 else
'red'
        draw_jug(ax, 1, 2, 4, 6, water_height_jug1, color1_jug1, color2_jug1,
current_jug1, jug1_capacity, water_transfer, handle_color='darkred')

        # Jug 2 (Blue Water)
```

```
    water_height_jug2 = (current_jug2 / jug2_capacity) * 6  # Scale water
height
    color1_jug2 = 'blue'
    color2_jug2 = 'red' if water_transfer and prev_jug1 > current_jug1 else
'blue'
    draw_jug(ax, 8, 2, 4, 6, water_height_jug2, color1_jug2, color2_jug2,
current_jug2, jug2_capacity, water_transfer, handle_color='darkblue')

    # Title and Target Achievement
    ax.set_title(f'Step {frame+1}: Jug 1 = {current_jug1}, Jug 2 =
{current_jug2}', fontsize=16)
    if current_jug1 == target or current_jug2 == target:
        ax.text(7, 9, 'Target Achieved!', ha='center', va='center', fontsize=20,
color='green', fontweight='bold')

# Key press event handling
class IndexTracker:
    def __init__(self, ax, frames):
        self.ax = ax
        self.frames = frames
        self.slices = len(frames)
        self.ind = 0
        self.update()

    def update(self):
        update(self.ind)
        fig.canvas.draw_idle()

    def on_key(self, event):
        if event.key == 'right':
            self.ind = (self.ind + 1) % self.slices
        elif event.key == 'left':
            self.ind = (self.ind - 1) % self.slices
        self.update()

tracker = IndexTracker(ax, states)

fig.canvas.mpl_connect('key_press_event', tracker.on_key)

plt.show()
```

```
# Example usage:
jug1_capacity = 13
jug2_capacity = 11
target = 8

water_jug_problem_animation(jug1_capacity, jug2_capacity, target)
```

OUTPUT: