

**The Superior University Lahore**

**Faculty of Computer Science & Information**

**Technology**

**Name: Alishba Haroon**

**Roll No: BSAI-116-4C**

**Date: 27 Feb 2025**

**Subject: PAI LAB**

**Water Jug Problem**

This Python program solves the Water Jug Problem using Breadth-First Search (BFS). The goal is to measure an exact amount of water using two jugs of given capacities

**Rules for the Water Jug Problem:**

**Eight rules** to manipulate the water in two jugs:

1. **Fill jug1:** Completely fill jug1 to its maximum capacity.

* **New state:** (jug1, y)

1. **Fill jug2:** Completely fill jug2 to its maximum capacity.

* **New state:** (x, jug2)

1. **Empty jug1:** Completely empty jug1.

* **New state:** (0, y)

1. **Empty jug2:** Completely empty jug2.

* **New state:** (x, 0)

1. **Pour water from jug1 into jug2** (without overflowing jug2).

* Water is transferred **until jug1 is empty** or **jug2 is full**.
* **New state:** (max(0, x - (jug2 - y)), min(jug2, y + x))

1. **Pour water from jug2 into jug1** (without overflowing jug1).

* Water is transferred **until jug2 is empty** or **jug1 is full**.
* **New state:** (min(jug1, x + y), max(0, y - (jug1 - x)))

1. **Pour all water from jug1 into jug2** (ignoring capacity limits).

* **New state:** (0, y + x)

1. **Pour all water from jug2 into jug1** (ignoring capacity limits).

* **New state:** (x + y, 0)

**1. water\_jug(jug1, jug2, goal) Function**

* This function performs BFS to find a sequence of steps that measure the target amount (goal).

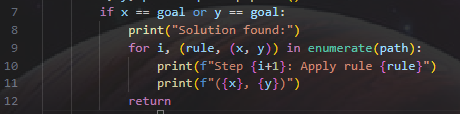
****

* **queue:** Stores tuples (x, y, path), where x and y are water amounts in jug1 and jug2, respectively, and path records steps taken.
* **visited:** Keeps track of previously explored states to avoid redundant processing.

**2. Loop Until Solution is Found:**

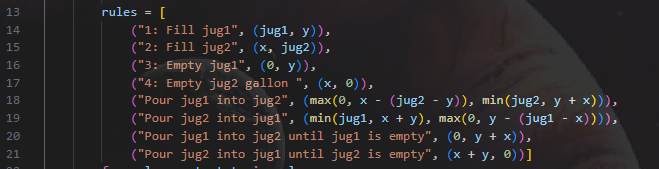
****

* Extracts the front element from the queue (x, y, path).
* If either jug holds the goal amount, the solution is printed:

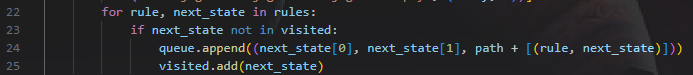


* The program then terminates successfully.

**3. Defining Possible Moves (Rules):**

****

**4. Exploring Next States:**

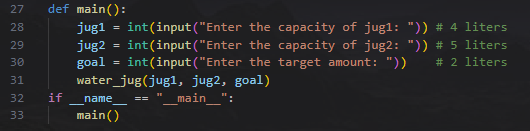
****

* **Loop through rules:** Each rule represents an action (e.g., fill, empty, pour), and next\_state is the new water level after applying the rule.
* Check if **next\_state is visited:** Prevents revisiting states, avoiding infinite loops.
* Add new state to **queue:**
* The new state (next\_state[0], next\_state[1]) is added to explore further.
* path + [(rule, next\_state)] tracks the sequence of steps leading to this state.
* Mark next\_state as visited: Ensures it won't be processed again.

**5. If No Solution Exists:**

****

**6. main() Function:**

****

* Users enter values for jug1, jug2, and goal.
* The program then attempts to find a solution.

**OUTPUT:**

