The Superior University, Lahore

Assignment-I (Fall 2023)

Course Title:	Programming for AI				Course Code:	CAI601410	Credit Hours: 4
Instructor:	Prof. Rasikh Ali				Programme Name:	BSDS	
Semester:	4th	Batch:	F23	Section:	BSDSM-4A	Date:	1st February, 2025
Time Allowed:					Maximum Marks:		
Student's Name:	Asaad Iqbal				Reg. No.	SU92-BSDSM-F23-020	

Lab-Task 3

1: Water Jug Problem

Task 1

Water Jug Problem Documentation

Code:

```
from collections import deque
```
[1] ✓ 0.1s

```
def waterJugProblem(capacity1, capacity2, goal):
    stack = deque()
    visited = set()

    stack.append((0, 0, []))
    visited.add((0, 0))

    while stack:
        jug1, jug2, path = stack.pop()

        if jug1 == goal or jug2 == goal:
            print("Solution Found!")
            print("\nSteps:")
            for step in path:
                print(step)
            print(f"Final State: ({jug1}, {jug2})")
            return True

        next_states = []

        if jug1 < capacity1:
            new_state = (capacity1, jug2)
            action = "Fill 4-gallon jug (Rule 1)"
            next_states.append((new_state, action))

        if jug2 < capacity2:
            new_state = (jug1, capacity2)
            action = "Fill 3-gallon jug (Rule 2)"
            next_states.append((new_state, action))
```

```
        if jug1 > 0:
            new_state = (0, jug2)
            action = "Empty 4-gallon jug (Rule 5)"
            next_states.append((new_state, action))

        if jug2 > 0:
            new_state = (jug1, 0)
            action = "Empty 3-gallon jug (Rule 6)"
            next_states.append((new_state, action))

        if (jug1 + jug2 >= capacity1) and (jug2 > 0):
            amount_poured = min(jug2, capacity1 - jug1)
            new_j1 = jug1 + amount_poured
            new_j2 = jug2 - amount_poured
            new_state = (new_j1, new_j2)
            action = f"Pour {amount_poured} from 3-gallon to 4-gallon jug until full (Rule 7)"
            next_states.append((new_state, action))

        if (jug1 + jug2 >= capacity2) and (jug1 > 0):
            amount_poured = min(jug1, capacity2 - jug2)
            new_j1 = jug1 - amount_poured
            new_j2 = jug2 + amount_poured
            new_state = (new_j1, new_j2)
            action = f"Pour {amount_poured} from 4-gallon to 3-gallon jug until full (Rule 8)"
            next_states.append((new_state, action))

        if (jug1 + jug2 <= capacity1) and (jug2 > 0):
            new_state = (jug1 + jug2, 0)
            action = "Pour all from 3-gallon to 4-gallon jug (Rule 9)"
            next_states.append((new_state, action))

        if (jug1 + jug2 <= capacity2) and (jug1 > 0):
            new_state = (0, jug1 + jug2)
            action = "Pour all from 4-gallon to 3-gallon jug (Rule 10)"
            next_states.append((new_state, action))

        next_states.reverse()
```

```
        next_states.reverse()

        for state, action in next_states:
            if state not in visited:
                new_path = path.copy()
                new_path.append(action)
                visited.add(state)
                stack.append((state[0], state[1], new_path))

    print("No solution exists.")
    return False
```

[2] ✓ 0.0s

```
jug1Capacity = 7
jug2Capacity = 5
target = 1
waterJugProblem(jug1Capacity, jug2Capacity, target)
```

[3] ✓ 0.0s

```
Solution Found!

Steps:
Fill 4-gallon jug (Rule 1)
Pour 5 from 4-gallon to 3-gallon jug until full (Rule 8)
Empty 3-gallon jug (Rule 6)
Pour all from 4-gallon to 3-gallon jug (Rule 10)
Fill 4-gallon jug (Rule 1)
Pour 3 from 4-gallon to 3-gallon jug until full (Rule 8)
Empty 3-gallon jug (Rule 6)
Pour all from 4-gallon to 3-gallon jug (Rule 10)
Fill 4-gallon jug (Rule 1)
Pour 1 from 4-gallon to 3-gallon jug until full (Rule 8)
Empty 3-gallon jug (Rule 6)
Pour 5 from 4-gallon to 3-gallon jug until full (Rule 8)
Final State: (1, 5)
```

This Python code uses Depth-First Search (DFS) to solve the Water Jug Problem. It aims to measure a specific quantity of water using two jugs of constant capacities. The code tries out all possible combinations of filling, draining, and pouring water from one jug to the other without repeating any state in order to prevent infinite loops.

The strategy is to:

- Employ a stack (deque) to try out states in LIFO (Last-In-First-Out) fashion (DFS strategy).
- Tracking visited states to prevent redundant calculations.
- Verifying if either jug is filled with the desired quantity of water at some point.
- In case a solution is discovered, the steps in the sequence resulting in the solution are printed.

Code Explanation

Initialize Data Structures

- A stack is employed to hold (jug1, jug2, path), where path records the moves made.
- A visited set is used to mark visited states and prevent redundancy.

Begin with Both Jugs Emptied

- The initial state is (0, 0), i.e., both jugs are empty.
- This state is pushed onto the stack, and the search procedure starts.

Check If the Goal is Reached

- If one jug holds the goal amount, the solution is found, and the program prints the steps taken.

Generate Possible Next States

- The program performs valid operations to derive new states:
- Fill a jug to the maximum (if it is not already full).
- Empty a jug to the minimum (if it is not already empty).
- Move water from one jug to another without overflowing.
- Each new state is pushed onto the stack to be explored later.

Avoid Revisiting States

- A new state is checked before pushing it onto the stack.
- If the state is visited, it is not pushed to avoid loops.

Repeat Until Solution or No More States

- The algorithm repeats until the stack is empty.
- If no solution is found, the program outputs "No solution exists."

Steps of the Water Jug Problem (General Strategy)

The Water Jug Problem uses these permissible operations (rules) to achieve a solution:

1. Fill the first jug to full (Rule 1).
2. Fill the second jug to full (Rule 2).
3. Empty the first jug to full (Rule 5).

4. Empty the second jug to full (Rule 6).
5. Fill the first jug from the second jug until the first jug is full or the second jug is empty (Rule 7).
6. Fill the second jug with water from the first jug until the second jug is full or the first jug is empty (Rule 8).
7. Empty the second jug into the first jug (if it fits) (Rule 9).
8. Empty the first jug into the second jug (if it fits) (Rule 10).
9. Applying these operations in an orderly fashion, the program determines the steps needed to measure the target quantity. In case no feasible sequence is available, the program indicates that there is no possible solution.