Lab3

Water Jug Problem DFS Implementation

Code:

def water\_jug\_dfs(capacity\_a, capacity\_b, target\_amount):

    """

    Capacity

    Jug A: 3L and Jug B: 5L

    soucre Unlimited

    """

    # The set to keep track of visited states (jug\_a\_amount, jug\_b\_amount)

    visited = set()

    # The initial state is (0, 0)

    initial\_state = (0, 0)

    # The path stores the sequence of states

    # It's a list of (state, previous\_state\_index) tuples in the stack

    # For a recursive DFS, we'll pass the current path

    def dfs\_recursive(current\_state, current\_path):

        """

        Recursive helper function for DFS.

        """

        jug\_a, jug\_b = current\_state

        # 1. Check for Target Condition

        # The condition is to have 2 liters in \*one\* of the jugs.

        if jug\_a == target\_amount or jug\_b == target\_amount:

            return current\_path + [current\_state]  # Found solution, return path

        # 2. Check for Visited State

        if current\_state in visited:

            return None

        # Mark the current state as visited and add it to the path

        visited.add(current\_state)

        new\_path = current\_path + [current\_state]

        # 3. Explore Possible Moves (Transitions)

        # List to hold potential next states (jug\_a\_amount, jug\_b\_amount)

        next\_states = []

        # Rule 1: Fill Jug A from source (unlimited source for A as per prompt)

        next\_states.append((capacity\_a, jug\_b))

        # Rule 2: Fill Jug B (from source - we assume both can be filled)

        next\_states.append((jug\_a, capacity\_b))

        # Rule 3: Empty Jug A

        next\_states.append((0, jug\_b))

        # Rule 4: Empty Jug B

        next\_states.append((jug\_a, 0))

        # Rule 5: Pour A to B

        transfer\_amount = min(jug\_a, capacity\_b - jug\_b)

        next\_states.append((jug\_a - transfer\_amount, jug\_b + transfer\_amount))

        # Rule 6: Pour B to A

        transfer\_amount = min(jug\_b, capacity\_a - jug\_a)

        next\_states.append((jug\_a + transfer\_amount, jug\_b - transfer\_amount))

        # Recurse on the next states

        for next\_state in next\_states:

            if next\_state not in visited:

                result = dfs\_recursive(next\_state, new\_path)

                if result is not None:

                    return result  # Solution found

        # If no solution found from this path, backtrack (implicitly done by recursion)

        # and remove current state from visited for other potential paths (only

        return None

    # Start the DFS from the initial state

    return dfs\_recursive(initial\_state, [])

# --- Problem Parameters ---

JUG\_A\_CAPACITY = 3 # Liters

JUG\_B\_CAPACITY = 5 # Liters

TARGET = 2         # Liters

# --- Execute and Print Result ---

solution\_path = water\_jug\_dfs(JUG\_A\_CAPACITY, JUG\_B\_CAPACITY, TARGET)

print(f"Goal: Get {TARGET} liters in a jug (A: {JUG\_A\_CAPACITY}L, B: {JUG\_B\_CAPACITY}L)")

print("---")

if solution\_path:

    print("✅ Solution Found via DFS:")

    # Define a helper function to describe the move

    def describe\_move(prev\_state, curr\_state):

        a1, b1 = prev\_state

        a2, b2 = curr\_state

        if a1 < JUG\_A\_CAPACITY and a2 == JUG\_A\_CAPACITY and b1 == b2:

            return "Fill Jug A (3L)"

        if b1 < JUG\_B\_CAPACITY and b2 == JUG\_B\_CAPACITY and a1 == a2:

            return "Fill Jug B (5L)"

        if a1 > 0 and a2 == 0 and b1 == b2:

            return "Empty Jug A"

        if b1 > 0 and b2 == 0 and a1 == a2:

            return "Empty Jug B"

        if a1 > 0 and b2 > b1 and a2 < a1:

            return "Pour A to B"

        if b1 > 0 and a2 > a1 and b2 < b1:

            return "Pour B to A"

        return "Initial State"

    print(f"| Step | Action | Jug A (3L) | Jug B (5L) |")

    print(f"|---|---|---|---|")

    for i, state in enumerate(solution\_path):

        jug\_a, jug\_b = state

        if i == 0:

            action = describe\_move(state, state)

        else:

            action = describe\_move(solution\_path[i-1], state)

        # Mark the final state where the condition is met

        if jug\_a == TARGET or jug\_b == TARGET:

            action = f"\*\*{action} (GOAL REACHED)\*\*"

        print(f"| {i} | {action} | {jug\_a} | {jug\_b} |")

else:

    print(" No solution found.")

