

The Superior University

|  |  |  |
| --- | --- | --- |
| Name: Abdul Moeez | Roll No: 017 | Course: PAI Lab |
| Semester: 4th | Section: BSAI 4A | Department: Artificial Intelligence |
| Submitted To: Sir Rasikh | Total Marks: | Date: |

**Lab 3**

**Task: Water Jug Problem: Documentation**

#### **Overview**

The Water Jug Problem is a classic puzzle where the goal is to measure a specific amount of water using two jugs of given capacities. This problem is often solved using a systematic approach, such as a state-space search, to explore all possible states of the jugs until the desired amount of water is achieved.

The provided Python code implements a solution to this problem using a **Depth-First Search (DFS)** approach. It systematically explores all possible states of the two jugs and finds a sequence of steps to reach the target amount of water.

#### **How the Code Works**

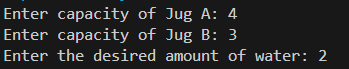
1. **Function: water\_jug(capacity\_a, capacity\_b, target)**
   1. This function solves the Water Jug Problem using a DFS-based approach.
   2. **Input Parameters:**
      1. capacity\_a: Capacity of Jug A.
      2. capacity\_b: Capacity of Jug B.
      3. target: Desired amount of water to measure.
   3. **Output:**
      1. Returns a list of states (tuples representing the amount of water in each jug) that lead to the target amount.
2. **Helper Function: get\_next\_states(state)**
   1. This function generates all possible next states from the current state of the jugs.
   2. **Possible Actions:**
      1. Fill Jug A to its full capacity.
      2. Fill Jug B to its full capacity.
      3. Empty Jug A.
      4. Empty Jug B.
      5. Pour water from Jug A to Jug B until Jug B is full or Jug A is empty.
      6. Pour water from Jug B to Jug A until Jug A is full or Jug B is empty.
   3. **Output:**
      1. Returns a list of all possible next states.
3. **DFS Implementation:**
   1. A stack is used to implement the DFS algorithm.
   2. The algorithm starts with both jugs empty ((0, 0)) and explores all possible states by applying the actions defined in get\_next\_states.
   3. Each state is marked as visited to avoid revisiting the same state.
   4. If the target amount is reached in either jug, the function returns the sequence of steps taken to achieve it.
4. **Function: solution(solution)**
   1. This function prints the solution steps in a user-friendly format.
   2. **Input:**
      1. solution: A list of states representing the sequence of steps to reach the target.
   3. **Output:**
      1. Prints each step with the amount of water in Jug A and Jug B.
      2. If no solution exists, it prints "No solution exists."
5. **User Input and Validation:**
   1. The user is prompted to input the capacities of Jug A, Jug B, and the target amount.
   2. Input validation ensures that all values are positive integers.

#### **Why This Approach?**

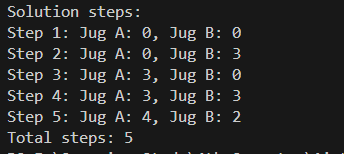
* **Systematic Exploration:** The DFS algorithm ensures that all possible states are explored systematically, guaranteeing a solution if one exists.
* **State-Space Representation:** The problem is modeled as a state-space search, where each state represents the amount of water in the two jugs.
* **Efficiency:** While DFS may not always find the shortest path, it is effective for exploring all possible solutions in a manageable state space.

#### **Example Usage**

**Input:**



**Output:**



#### **Key Points**

* The code handles all edge cases, such as invalid inputs or unsolvable scenarios.
* The solution is presented in a step-by-step manner, making it easy to understand and follow.
* The algorithm is flexible and can be adapted to similar state-space search problems.

#### **Conclusion**

This implementation provides an efficient and clear solution to the Water Jug Problem using a DFS-based approach. It demonstrates the power of systematic state-space exploration in solving classic puzzles and can serve as a foundation for solving similar problems.