SUPERIOR UNIVERSITY

Programming for Artificial Intelligence – Lab

Task 3

Name: Minhaj Asghar

Roll no: SU92-BSAIM-F23-108

Section: BSAI-4B

Submitted to: Sir Rasikh Ali

Water Jug Problem - Report

Introduction

The Water Jug Problem is a classic problem in artificial intelligence and computer science, where the goal is to measure a specific amount of water using two jugs with different capacities. The challenge lies in the ability to use only a limited set of operations to reach the desired amount of water. This problem can be efficiently solved using search algorithms such as Depth-First Search (DFS).

Problem Statement

Given two jugs with capacities c1 and c2, and a goal amount of water goal, the task is to determine a sequence of operations that will result in one of the jugs containing exactly goal units of water. The operations allowed are:

- Fill a jug to its full capacity.
- Empty a jug.
- Pour water from one jug into the other until one is either full or empty.

The problem is solved by applying DFS to explore the possible states of the jugs and finding the sequence of actions that lead to the goal.

Approach

The approach used to solve the Water Jug Problem in this implementation is **Depth-First Search (DFS)**. The steps are as follows:

1. **State Representation**: The state of the problem is represented as a tuple (jug1, jug2), where jug1 and jug2 are the current amounts of water in the two jugs.
2. **DFS Search**: Starting from the initial state (0, 0) (both jugs are empty), DFS explores all possible states by applying the allowed operations (rules) to the current state.
3. **Goal Check**: If at any point, the amount of water in either jug matches the goal, the solution is found.
4. **Backtracking**: The algorithm keeps track of the parent of each state to reconstruct the path from the initial state to the solution.

Code Implementation:

```python
def waterJugProblemDFS(c1, c2, goal):
    stack = [(0, 0)]
    visited = set()
    parent = {}
    actions = []

    visited.add((0, 0))
    parent[(0, 0)] = None

    while stack:
        jug1, jug2 = stack.pop()
        actions.append((jug1, jug2))

        if jug1 == goal or jug2 == goal:
            print("Solution found")
            return reconstruct_path(parent, (jug1, jug2))

        rules = [
            (c1, jug2),
            (jug1, c2),
            (0, jug2),
            (jug1, 0),
            (jug1 - min(jug1, c2 - jug2), jug2 + min(jug1, c2 - jug2)),
            (jug1 + min(jug2, c1 - jug1), jug2 - min(jug2, c1 - jug1))
        ]

        for rule in rules:
            if rule not in visited:
                visited.add(rule)
                stack.append(rule)
                parent[rule] = (jug1, jug2)

    print("No solution found")
    return None
```

Explanation of Code:

1. **waterJugProblemDFS(c1, c2, goal)**: This function performs the Depth-First Search to find the solution to the Water Jug problem. It uses a stack to explore states and a set (visited) to ensure each state is explored only once.
    - The function starts with both jugs empty (0, 0).
    - It applies all possible operations to the current state and pushes the new states onto the stack.
    - If the goal is reached, it reconstructs and returns the sequence of actions that lead to the solution.
2. **reconstruct_path(parent, state)**: This function backtracks from the goal state to the initial state, reconstructing the path of operations taken.

State Transition Rules:

- **Fill Jug 1**: Set jug1 = c1 (fill jug1).
- **Fill Jug 2**: Set jug2 = c2 (fill jug2).
- **Empty Jug 1**: Set jug1 = 0 (empty jug1).
- **Empty Jug 2**: Set jug2 = 0 (empty jug2).
- **Pour Jug 1 into Jug 2**: Pour water from jug1 to jug2 until jug2 is full or jug1 is empty.
- **Pour Jug 2 into Jug 1**: Pour water from jug2 to jug1 until jug1 is full or jug2 is empty.

Example Output

For jug1capacity = 4, jug2capacity = 3, and goal = 2, the output will be:

```
Solution found
Steps to reach the goal:
(0, 0)
(0, 3)
(3, 0)
(3, 3)
(4, 2)
```

In this example:

- Each step shows the state of the two jugs after performing an operation.
- The goal is to have exactly 2 units of water in one of the jugs.

Time Complexity

The time complexity of this algorithm is dependent on the number of states and the number of operations per state:

- In the worst case, DFS explores all possible states. Since there are at most c1 * c2 states (where c1 and c2 are the capacities of the jugs), the complexity is O(c1 * c2).
- The number of possible operations per state is constant (6 operations), so the time complexity is O(c1 * c2).

Conclusion

The Water Jug Problem is a typical example of a search problem that can be solved using Depth -First Search (DFS). By exploring the possible states and backtracking when necessary, we can find a solution to the problem. The DFS approach is effective for problems with a manageable number of states, and it provides an intuitive way to explore all possibilities systematically.