Experiment No. 1

Title: Implementation of Depth First Search for Water Jug problem.

Aim: To implement the Depth First Search (DFS) algorithm to solve the Water Jug problem using python.

Objectives:

1. To study the DFS algorithm.
2. To explore the application of DFS in finding a solution to the classic problem of measuring a certain quantity of water using two jugs of different capacities. 3. To implement DFS for water jug problems using python.

Problem Statement:
Implement Depth First Search to find a sequence of operations enabling measurement of exactly C liters of water using two given jugs of capacities A and B, where A, B, C are positive integers.

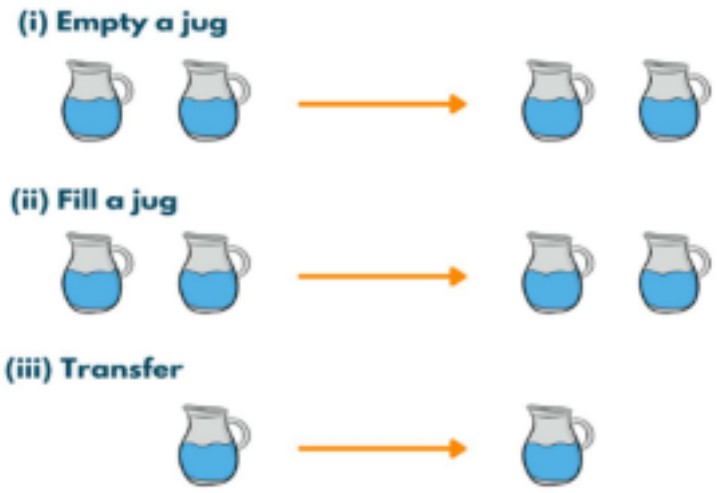
Software/s: Python (version 3.x)

Theory:

The Water Jug problem is a classic example in problem-solving and algorithms. The scenario involves two jugs with different capacities, and the goal is to measure a specific quantity of water using these jugs. The problem is defined by the capacities of the jugs, the desired quantity to be measured, and the operations allowed (filling, emptying, pouring.

Department of Artificial Intelligence and Data Science

AISSMS
INSTITUTE OF INFORMATION TECHNOLOGY
ADDING VALUE TO ENGINEERING



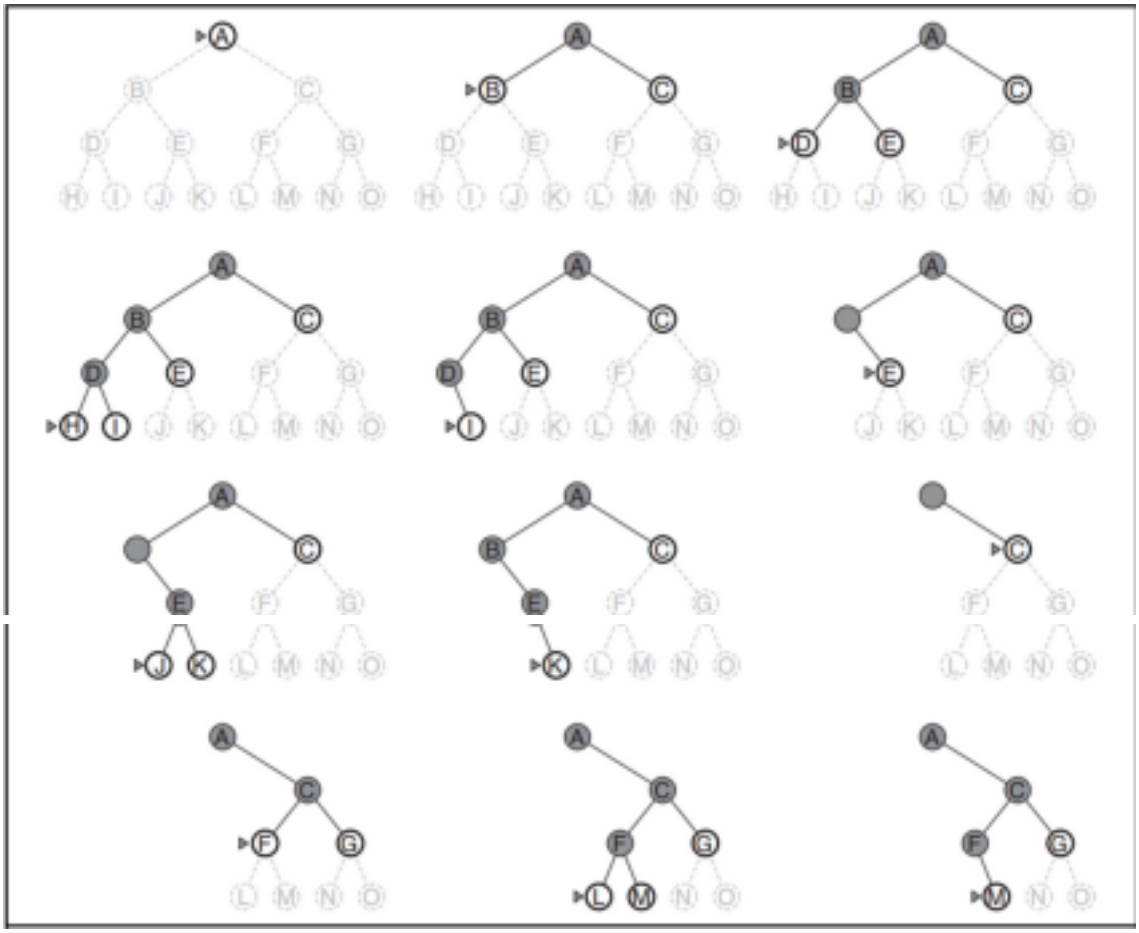


Figure: Depth-first search on a binary tree. The unexplored region is shown in light gray. Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors and M is the only goal node.

Depth First Search (DFS) is a graph traversal algorithm that explores as far as possible along each branch before backtracking. In the context of the Water Jug problem, the state space can be represented as a graph, where each node represents a particular state (configuration of water in the jugs), and edges represent possible operations to transition between states.

Depth-first search always expands the deepest node in the current frontier of the search tree. The progress of the search is illustrated in Figure. The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. As those nodes are expanded, they are dropped from the frontier, so then the search “backs up” to the next deepest node that still has unexplored successors.

Pseudo code:

function DEPTH-FIRST-SEARCH(problem) returns a solution, or failure node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0 if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
 frontier ← a LIFO stack with node as the only element
 explored ← an empty set

 loop do
 if EMPTY?(frontier) then return failure
 node ← POP(froner) /* chooses the deepest node in froner */
add node.STATE to explored

 for each action in problem.ACTIONS(node.STATE) do
 child ← CHILD-NODE(problem, node, action)

 if child.STATE is not in explored or frontier then
 if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
frontier ← PUSH(child, froner) /* LIFO enqueue for depth-first search */

Procedure:
State Representation:

- ○ Define a representation for the states in the problem. In the Water Jug problem, a state can be described by the water levels in both Jug A and Jug B.

- DFS Algorithm:
  - ○ Start with an initial state, typically where both jugs are empty. ○ Explore possible operations from the current state, such as filling Jug A, filling Jug B, emptying Jug A, emptying Jug B, pouring water from Jug A to Jug B, and pouring water from Jug B to Jug A.
  - ○ For each operation, create a new state reflecting the outcome of that operation.

- Recursively apply DFS to explore the next states generated by each operation.
- Keep track of visited states to avoid infinite loops and revisit the same configuration.

- Goal Check:
  - At each step, check if the current state satisfies the goal condition. For the Water Jug problem, the goal is typically to have a specific amount of water in one of the jugs.
- Backtracking:
  - If the goal is not reached from the current state, backtrack to the previous state and explore alternative paths. This involves undoing the last operation to try a different one.
- Termination:
  - Terminate the search when a solution is found or when all possible paths have been explored.
- Complexity and Optimization:
  - Analyze the time and space complexity of the DFS algorithm for the Water Jug problem. Consider optimizations such as memoization to avoid revisiting already explored states, which can significantly improve efficiency.
- Visualization (Optional):
  - Optionally, visualize the state space and the exploration process. This could help in understanding how DFS traverses through different configurations of the jugs.
- Experimentation:
  - Test the DFS algorithm with different capacities for Jug A and Jug B, as well as various goal quantities. This allows you to observe how the algorithm finds solutions for different problem instances.

By following these steps, you can systematically implement the DFS algorithm for the Water Jug problem, gaining insights into how the algorithm explores the state space and identifies solutions through a depth-first exploration strategy.

Conclusion:

_____

_____

References:

S. Russel, P. Norvig, “Artificial Intelligence – A Modern Approach”, Third Edition, Pearson Education, 2015.