**Name:** Swapnil Thorat

**PRN:** 121B1C055

**Assignment No. 2**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract MerkleTree {

    bytes32[] public leaves;  // Array of leaf hashes

    bytes32 public merkleRoot; // Store the Merkle root

    // Add new leaf (hashed) to the leaves array

    function addLeaf(bytes32 \_leaf) public {

        leaves.push(\_leaf);

    }

    // Function to construct the Merkle Tree and compute Merkle Root

    function generateMerkleRoot() public {

        require(leaves.length > 0, "No leaves available");

        uint n = leaves.length;

        bytes32[] memory currentLevel = leaves;

        // Continue hashing pairs of nodes until one hash (the root) remains

        while (n > 1) {

            uint nextLevelLength = n / 2;

            if (n % 2 == 1) {

                nextLevelLength++;

            }

            bytes32[] memory nextLevel = new bytes32[](nextLevelLength);

            for (uint i = 0; i < n / 2; i++) {

                nextLevel[i] = keccak256(abi.encodePacked(currentLevel[2 \* i], currentLevel[2 \* i + 1]));

            }

            // If there is an odd number of elements, promote the last one

            if (n % 2 == 1) {

                nextLevel[nextLevelLength - 1] = currentLevel[n - 1];

            }

            currentLevel = nextLevel;

            n = nextLevelLength;

        }

        // The final root hash

        merkleRoot = currentLevel[0];

    }

    // Verify whether a leaf belongs to the Merkle Tree using Merkle proof

    function verify(bytes32 leaf, bytes32[] memory proof, bytes32 root) public pure returns (bool) {

        bytes32 computedHash = leaf;

        // Rebuild the hash from the proof

        for (uint i = 0; i < proof.length; i++) {

            bytes32 proofElement = proof[i];

            if (computedHash <= proofElement) {

                // Hash current computed hash with the proof element

                computedHash = keccak256(abi.encodePacked(computedHash, proofElement));

            } else {

                // Hash proof element with current computed hash

                computedHash = keccak256(abi.encodePacked(proofElement, computedHash));

            }

        }

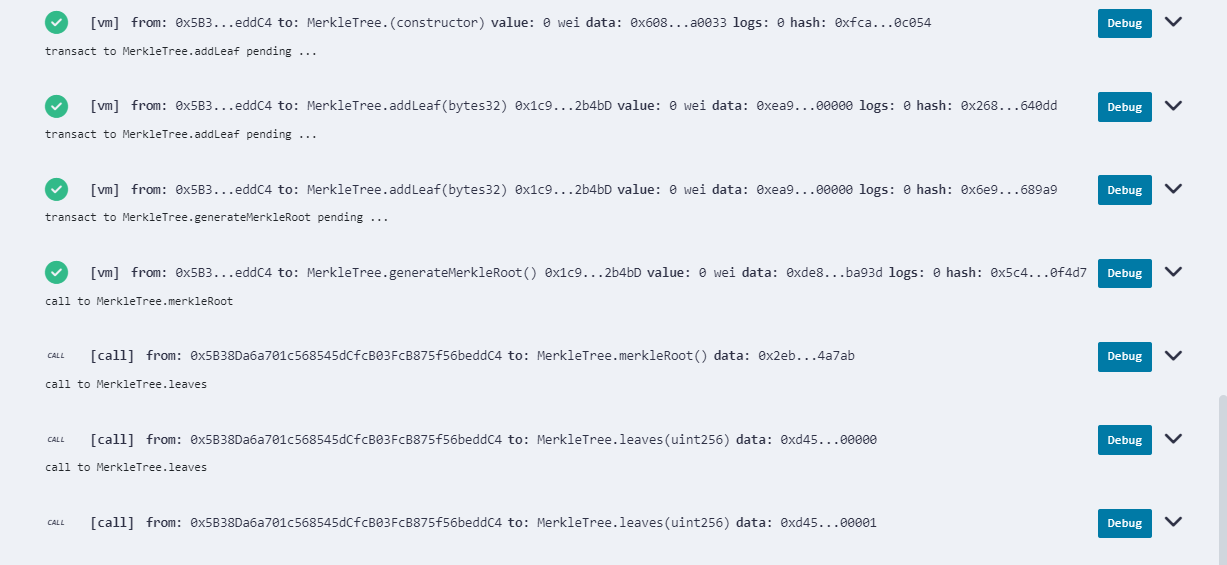
        // Check if the rebuilt hash matches the root

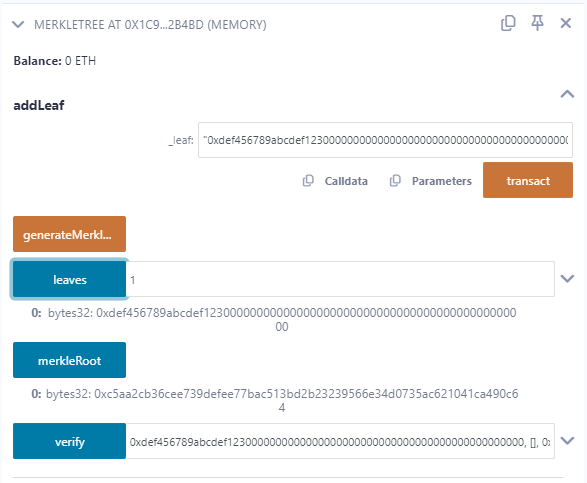
        return computedHash == root;

    }

}

**Output:**

****

****