NAME: NAKHIREDDI BHARATH CHANDRA

REG NO: 20BCE0770

DIGITAL ASSESSMENT

Merkle Trees (Hash Tree):

Introduction:

Merkle trees is an implementation of binary trees where each non-leaf node is a hash of the two child nodes. The leaves can either be the data itself or a hash/signature of the data.

If we found any difference in root hash between the systems, we can use binary search to detect the defective subtree. Thus by simple log(N) complexity, we can find the problematic area or say problematic subtree.

Usages:

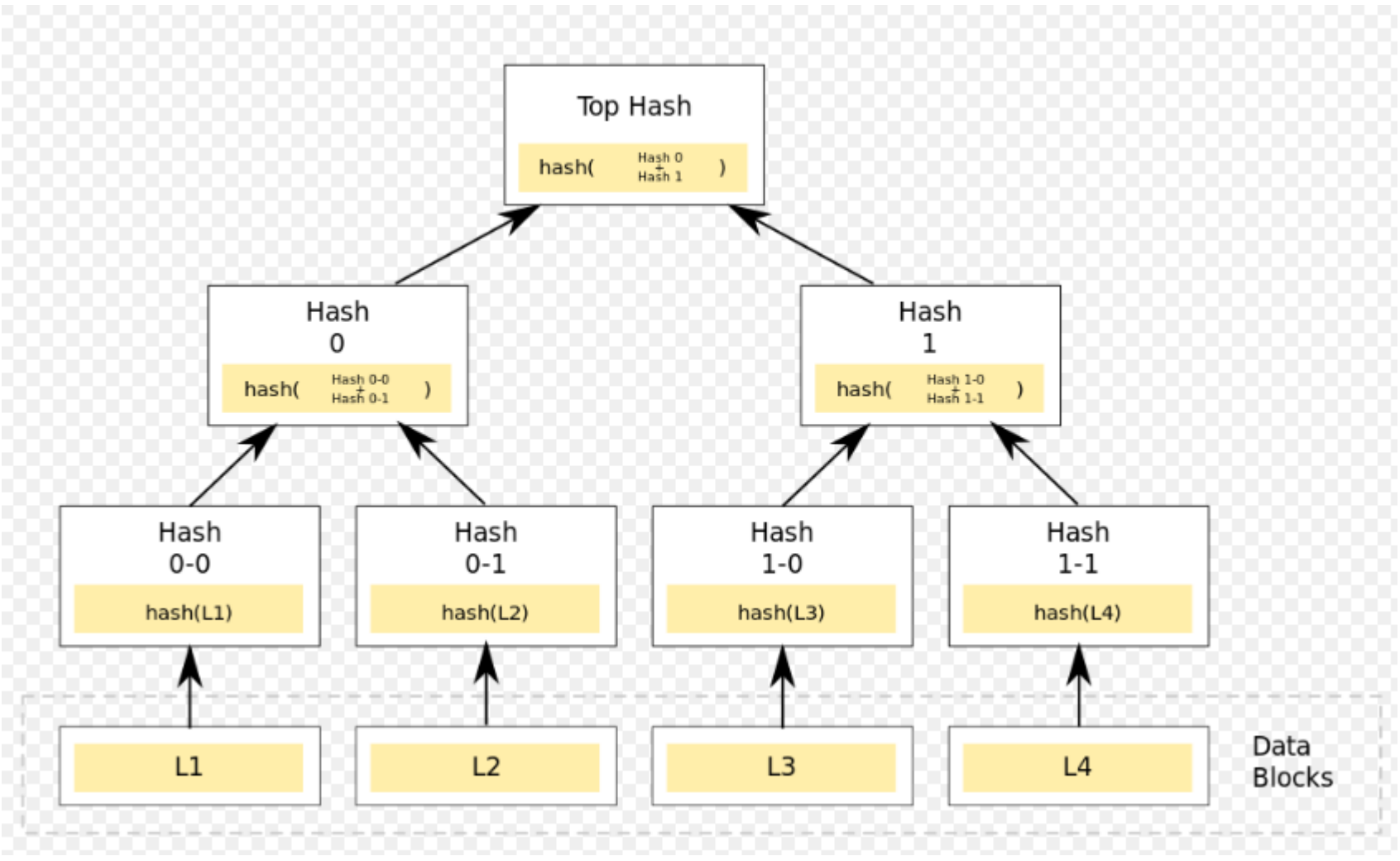
Merkle tree(Hash tree) is used to verify any kind of data stored, handled and transferred in and between computers.

Currently, the main use of Merkle tree is to make sure that data blocks received from other peers in a peer-to-peer network are received undamaged and unaltered, and even to check that the other peers do not lie and send fake blocks.

Merkle tree is used in git, Amazon's Dynamo, Cassandra as well as BitCoin.

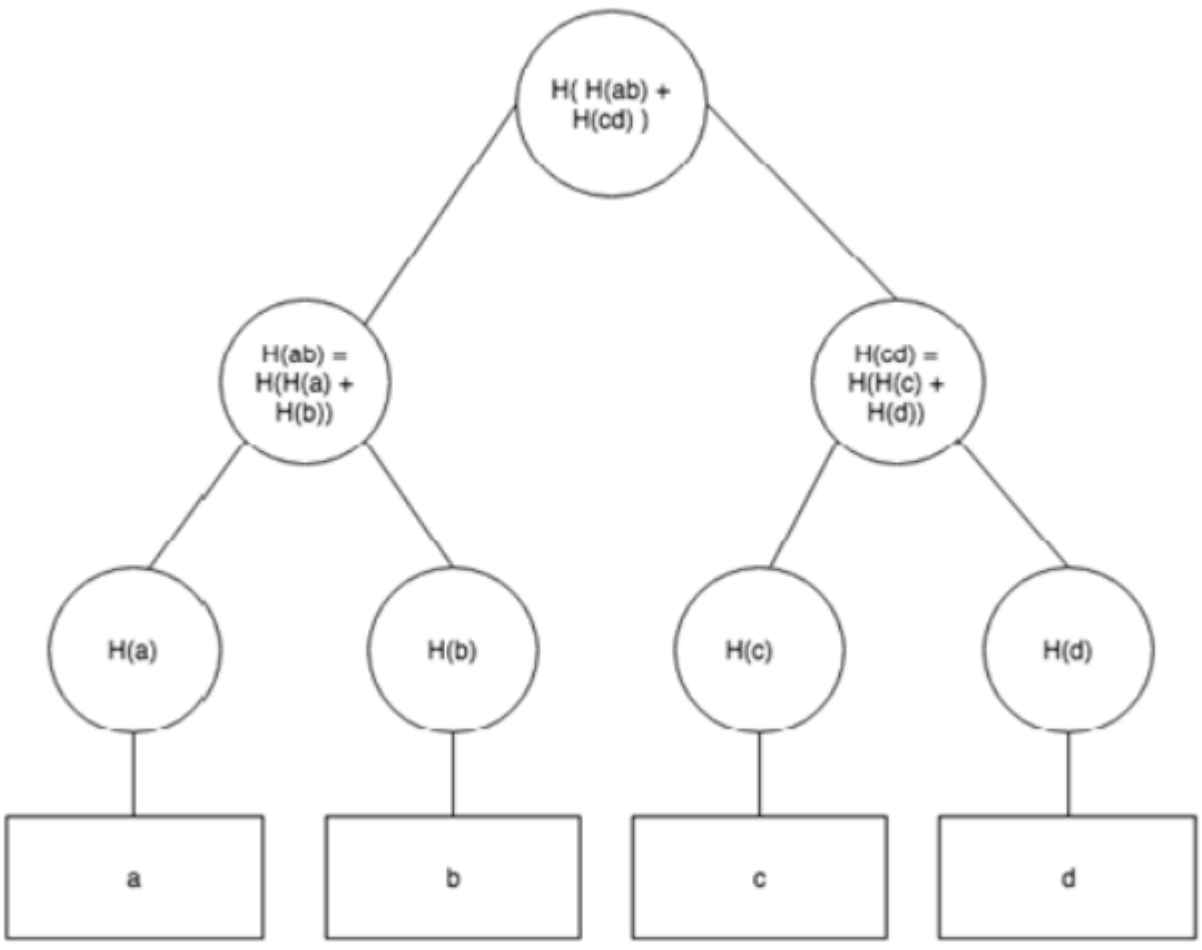
Architecture of Merkle tree:

It is a tree structure in which each leaf node is a hash of a block of data, and each non-leaf node is a hash of its children. Typically, Merkle trees have a branching factor of 2, meaning that each node has up to 2 children.



Algorithm and Implementation:

A Merkle tree is constructed by recursively hashing pairs of nodes until there is only one hash.



a, *b*, *c*, and *d* are some data elements (files, JSON, etc) and *H* is a hash function.

a hash function acts as a “digital fingerprint” of some piece of data by mapping it to a simple string with a low probability that any other piece of data will map to the same string.

Each node is created by hashing the concatenation of its “parents” in the tree.

Note: Merkle tree are mostly a binary tree but there are also Trees. Platforms like Ethereum use non binary tree.

Implementation In Java:

We are going to implement binary merkle tree. As the first step, let's define the node. Like a regular tree, it has a data field to store the hash and left and right pointers to point to left child and right child of the binary tree.

```
package com.merkle.tree.implementation;


import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;


public class MerkleTree {
    public static Node generateTree(ArrayList<String> dataBlocks) {
        ArrayList<Node> childNodes = new ArrayList<>();

        for (String message : dataBlocks) {
            childNodes.add(new Node(null, null, HashAlgorithm.generateHash(message)));
        }

        return buildTree(childNodes);
    }

    private static Node buildTree(ArrayList<Node> children) {
        ArrayList<Node> parents = new ArrayList<>();

        while (children.size() != 1) {
            int index = 0, length = children.size();
            while (index < length) {
                Node leftChild = children.get(index);
                Node rightChild = null;

                if ((index + 1) < length) {
                    rightChild = children.get(index + 1);
                } else {
                    rightChild = new Node(null, null, leftChild.getHash());
                }

                String parentHash = HashAlgorithm.generateHash(leftChild.getHash() + rightChild.getHash());
                parents.add(new Node(leftChild, rightChild, parentHash));
                index += 2;
            }
```

```
            children = parents;
            parents = new ArrayList<>();
        }
        return children.get(0);
    }

    private static void printLevelOrderTraversal(Node root) {
        if (root == null) {
            return;
        }

        if ((root.getLeft() == null && root.getRight() == null)) {
            System.out.println(root.getHash());
        }
        Queue<Node> queue = new LinkedList<>();
        queue.add(root);
        queue.add(null);

        while (!queue.isEmpty()) {
            Node node = queue.poll();
            if (node != null) {
                System.out.println(node.getHash());
            } else {
                System.out.println();
                if (!queue.isEmpty()) {
                    queue.add(null);
                }
            }

            if (node != null && node.getLeft() != null) {
                queue.add(node.getLeft());
            }

            if (node != null && node.getRight() != null) {
                queue.add(node.getRight());
            }

        }

    }
```

```
    public static void main(String[] args) {
        ArrayList<String> dataBlocks = new ArrayList<>();
        dataBlocks.add("Captain America");
        dataBlocks.add("Iron Man");
        dataBlocks.add("God of thunder");
        dataBlocks.add("Doctor strange");
        Node root = generateTree(dataBlocks);
        printLevelOrderTraversal(root);
    }
}
```

References:

1. https://www.linkedin.com/pulse/merkle-tree-its-implementation-java-nikhil-goyal#:~:text=Merkle%20trees%20is%20an%20implementation,to%20detect%20the%20defective%20subtree.
2. https://en.wikipedia.org/wiki/Merkle_tree