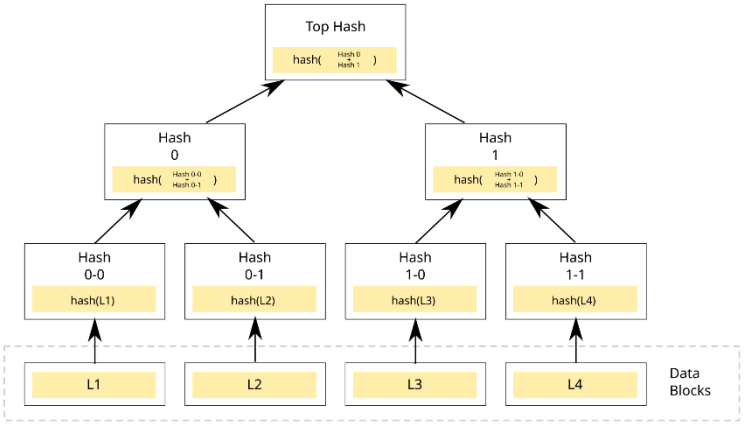
PRACTICAL 3

Name:	Harsh Shah	**Semester:**	VII	**Division:**	6
Roll No.:	21BCP359	**Date:**	08-08-24	**Batch:**	G11
Aim:	To Implement Merkle Tree using any programming language				

Merkle Tree

A Merkle tree is a data encryption structure used in data management applications where data is sent through a hashing algorithm in different ways to create a hash that represents all of the data in a file.



Complexity	**Average**	**Worst**
Space	O(n)	O(n)
Search	$O(\log_2(n))$	$O(\log_k(n))$
Insert	$O(\log_2(n))$	$O(\log_k(n))$
Delete	$O(\log_2(n))$	$O(\log_k(n))$

- Merkle tree also known as hash tree is a data structure used for data verification and synchronization.
- It is a tree data structure where each non-leaf node is a hash of its child nodes.
- All the leaf nodes are at the same depth and are as far left as possible.
- It maintains data integrity and uses hash functions for this purpose.

Program

```
import hashlib

class MerkleTreeNode:
    def __init__(self, value):
        self.left = None
        self.right = None
        self.value = value
        self.hashValue = hashlib.sha256(value.encode("utf-8")).hexdigest()

def buildTree(leaves, f):
    nodes = []
    for i in leaves:
        nodes.append(MerkleTreeNode(i))

    while len(nodes) != 1:
        temp = []
```

```
        for i in range(0, len(nodes), 2):
            node1 = nodes[i]
            if i + 1 < len(nodes):
                node2 = nodes[i + 1]
            else:
                temp.append(nodes[i])
                break
            f.write(
                "Left child : " + node1.value + " | Hash : " + node1.hashValue + " \n"
            )
            f.write(
                "Right child : " + node2.value + " | Hash : " + node2.hashValue + " \n"
            )
            concatenatedHash = node1.hashValue + node2.hashValue
            parent = MerkleTreeNode(concatenatedHash)
            parent.left = node1
            parent.right = node2
            f.write(
                "Parent(concatenation of "
                + node1.value
                + " and "
                + node2.value
                + ") : "
                + parent.value
                + " | Hash : "
                + parent.hashValue
                + " \n"
            )
            temp.append(parent)
        nodes = temp
    return nodes[0]

inputString = input("Enter the leaves as a comma-separated list (e.g., 'a,b,c,d'): ")
leaves = inputString.split(",")

with open("merkle.tree", "w") as f:
    root = buildTree(leaves, f)
```

Output

```
PS C:\Users\harsh\OneDrive - pdpu.ac.in\HARSH\_PDEU\SEM 7\Blockchain\Blockchain Lab\practical3>
lockchain\Blockchain Lab\practical3\test.py"
Enter the leaves as a comma-separated list (e.g., 'a,b,c,d'): a,b,c,d,e
```

merkle.tree

```
Left child : a | Hash : ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb
Right child : b | Hash : 3e23e8160039594a33894f6564e1b1348bbd7a0088d42c4acb73eeaed59c009d
Parent(concatenation of a and b) :
ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb3e23e8160039594a33894f6564e1b1348bbd7a0088d42c4acb73eeaed59c009d | Hash :
62af5c3cb8da3e4f25061e829ebeea5c7513c54949115b1acc225930a90154da
Left child : c | Hash : 2e7d2c03a9507ae265ecf5b5356885a53393a2029d241394997265a1a25aefc6
Right child : d | Hash : 18ac3e7343f016890c510e93f935261169d9e3f565436429830faf0934f4f8e4
Parent(concatenation of c and d) :
2e7d2c03a9507ae265ecf5b5356885a53393a2029d241394997265a1a25aefc618ac3e7343f016890c510e93f935261169d9e3f565436429830faf0934f4f8e4 | Hash :
d3a0f1c792ccf7f1708d5422696263e35755a86917ea76ef9242bd4a8cf4891a
Left child : ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb3e23e8160039594a33894f6564e1b1348bbd7a0088d42c4acb73eeaed59c009d | Hash :
62af5c3cb8da3e4f25061e829ebeea5c7513c54949115b1acc225930a90154da
Right child : 2e7d2c03a9507ae265ecf5b5356885a53393a2029d241394997265a1a25aefc618ac3e7343f016890c510e93f935261169d9e3f565436429830faf0934f4f8e4 | Hash :
d3a0f1c792ccf7f1708d5422696263e35755a86917ea76ef9242bd4a8cf4891a
Parent(concatenation of ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb3e23e8160039594a33894f6564e1b1348bbd7a0088d42c4acb73eeaed59c009d
and 2e7d2c03a9507ae265ecf5b5356885a53393a2029d241394997265a1a25aefc618ac3e7343f016890c510e93f935261169d9e3f565436429830faf0934f4f8e4) :
62af5c3cb8da3e4f25061e829ebeea5c7513c54949115b1acc225930a90154dad3a0f1c792ccf7f1708d5422696263e35755a86917ea76ef9242bd4a8cf4891a | Hash :
58c89d709329eb37285837b042ab6ff72c7c8f74de0446b091b6a0131c102cfd
Left child : 62af5c3cb8da3e4f25061e829ebeea5c7513c54949115b1acc225930a90154dad3a0f1c792ccf7f1708d5422696263e35755a86917ea76ef9242bd4a8cf4891a | Hash :
58c89d709329eb37285837b042ab6ff72c7c8f74de0446b091b6a0131c102cfd
Right child : e | Hash : 3f79bb7b435b05321651daefd374cdc681dc06faa65e374e38337b88ca046dea
Parent(concatenation of 62af5c3cb8da3e4f25061e829ebeea5c7513c54949115b1acc225930a90154dad3a0f1c792ccf7f1708d5422696263e35755a86917ea76ef9242bd4a8cf4891a
and e) : 58c89d709329eb37285837b042ab6ff72c7c8f74de0446b091b6a0131c102cfd3f79bb7b435b05321651daefd374cdc681dc06faa65e374e38337b88ca046dea | Hash :
dea979f026a014fcb2300d6300e73ae1ccfb0dd238835d33895286d610eb7c4f
```