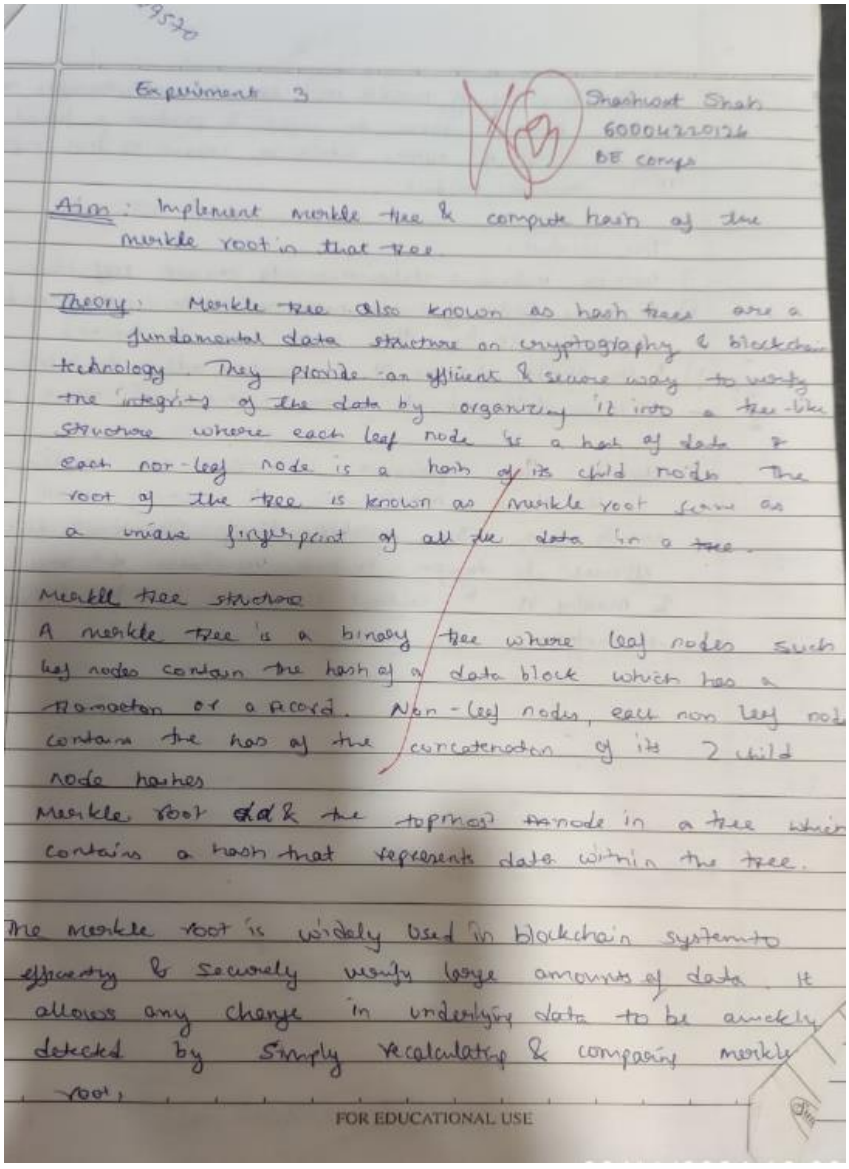
BLOCKCHAIN TECHNOLOGY

EXPERIMENT NO.03



Experiment 3

Shashwat Shah
60004220126
BE comps

Aim: Implement merkle tree & compute hash of the merkle root in that tree.

Theory: Merkle tree also known as hash trees are a fundamental data structure in cryptography & blockchain technology. They provide an efficient & secure way to verify the integrity of the data by organizing it into a tree-like structure where each leaf node is a hash of data & each non-leaf node is a hash of its child nodes. The root of the tree is known as merkle root serve as a unique fingerprint of all the data in a tree.

Merkle Tree structure

A merkle tree is a binary tree where leaf nodes such leaf nodes contain the hash of a data block which has a transaction or a record. Non-leaf nodes, each non leaf node contains the has of the concatenation of its 2 child node hashes

Merkle root is the topmost node in a tree which contains a hash that represents data within the tree.

The merkle root is widely used in blockchain systems to efficiently & securely verify large amounts of data. It allows any change in underlying data to be quickly detected by simply recalculating & comparing merkle root.

```
from typing import List
import hashlib

class Node:
    def __init__(self, left, right, value: str, content, is_copied=False) -> None:
        self.left: Node = left
        self.right: Node = right
        self.value = value
        self.content = content
        self.is_copied = is_copied

    @staticmethod
    def hash(val: str) -> str:
        return hashlib.sha256(val.encode('utf-8')).hexdigest()

    def __str__(self):
        return (str(self.value))

    def copy(self):
        return Node(self.left, self.right, self.value, self.content, True)

class MerkleTree:
    def __init__(self, values: List[str]) -> None:
        self.__buildTree(values)

    def __buildTree(self, values: List[str]) -> None:
        leaves: List[Node] = [Node(None, None, Node.hash(e), e) for e in values]
        if len(leaves) % 2 == 1:
            leaves.append(leaves[-1].copy())
        self.root: Node = self.__buildTreeRec(leaves)

    def __buildTreeRec(self, nodes: List[Node]) -> Node:
        if len(nodes) % 2 == 1:
            nodes.append(nodes[-1].copy())
        half: int = len(nodes) // 2

        if len(nodes) == 2:
            return Node(nodes[0], nodes[1], Node.hash(nodes[0].value + nodes[1].value),
nodes[0].content+"+"+nodes[1].content)

        left: Node = self.__buildTreeRec(nodes[:half])
        right: Node = self.__buildTreeRec(nodes[half:])
        value: str = Node.hash(left.value + right.value)
        content: str = f'{left.content}+{right.content}'
        return Node(left, right, value, content)

    def printTree(self) -> None:
        self.__printTreeRec(self.root)

    def __printTreeRec(self, node: Node) -> None:
        if node != None:
            if node.left != None:
                print("Left: "+str(node.left))
                print("Right: "+str(node.right))
            else:
                print("Input")

            if node.is_copied:
                print('(Padding)')
            print("Value: "+str(node.value))
            print("Content: "+str(node.content))
            print("")
            self.__printTreeRec(node.left)
```

```
            self.__printTreeRec(node.right)

    def getRootHash(self) -> str:
        return self.root.value

def mixmerkletree() -> None:
    n = int(input("Enter the number of blocks: "))
    elems = []
    print("Enter the blocks: ")
    for i in range(n):
        elems.append(input())
    print("Inputs: ")
    print(*elems, sep=" | ")
    print("")
    mtree = MerkleTree(elems)
    print("Root Hash: "+mtree.getRootHash()+"\n")
    mtree.printTree()

mixmerkletree()
```