Bhartiya Vidya Bhavan's
Sardar Patel Institute of Technology, Mumbai-400058
Department of Electronics and Telecommunication Engineering
IT424:Blockchain Technology and Applications

Lab3A and 3B: Blockchain Programming

Objective: Develop a simple Blockchain

Outcomes: After successful completion of lab students should be able to
Develop and demonstrate Blockchain fundamentals
Write a code in python to build a Blockchain
Demonstrate the Blockchain basic working
Solve cryptographic puzzles for the desired prefix of the hash value of a blockchain.

System Requirements:
PC (C2D, 4GB RAM, 100GB HDD space and NIC)
Ubuntu Linux 14.04/20.04
Internet connectivity
Python Cryptography and Pycrypto

Part-3A: Implementing Simple Blockchain using Python

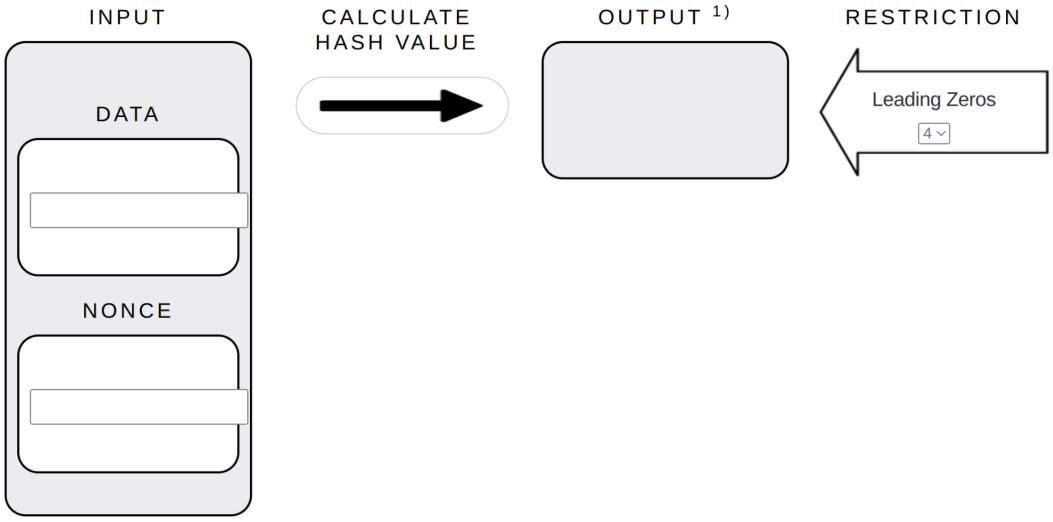

Figure-1: Hash puzzle

Write a code for a simple blockchain concept.
Compute the hash.
Solve the cryptographic puzzle (nonce) with prefix four zeros to hash value of a block.

Code:

```
import datetime

import hashlib


class Block:

    blockNo = 0

    data = None

    next = None

    hash = None

    nonce = 0

    previous_hash = 0x0

    proof=0

    timestamp = datetime.datetime.now()


    def __init__(self, data):

        self.data = data


    def hash(self):

        h = hashlib.sha256()

        h.update(

        str(self.nonce).encode('utf-8') +

        str(self.data).encode('utf-8') +
```

```
        str(self.previous_hash).encode('utf-8') +
        str(self.timestamp).encode('utf-8') +
        str(self.blockNo).encode('utf-8')
        )
        return h.hexdigest()

    def __str__(self):
        return "Block Hash: " + str(self.hash()) +"\nPrevious Hash: "+
str(self.previous_hash)+"\nBlockNo: " + str(self.blockNo) + "\nBlock Data: " + str(self.data) +
"\nNonce: " + str(self.nonce) +"\nProof: "+str(self.proof)+ "\n--------------"


class Blockchain:

    block = Block("Genesis")
    dummy = head = block

    def add(self, block):

        block.previous_hash = self.block.hash()
        block.blockNo = self.block.blockNo + 1

        self.block.next = block
        self.block = self.block.next

    def proof_of_work(self, block, difficulty):
```

```
        block.nonce = 0
        computed_hash = block.hash()
        while not computed_hash.startswith('0' * difficulty):
            block.nonce += 1
            computed_hash = block.hash()
        return computed_hash

    def mine(self,new_block):
        new_block.proof = self.proof_of_work(new_block,4)
        self.add(new_block)




blockchain = Blockchain()

for n in range(5):
    data=input('Enter the data to be stored in block '+str(n+1)+' ')
    blockchain.mine(Block(data))

while blockchain.head != None:
    print(blockchain.head)
    blockchain.head = blockchain.head.next
```

Output:

```
PS C:\Users\Nupur Gupte> python -u "c:\Users\Nupur Gupte\Desktop\files\Blockchain\simple_blockchain.py"
Enter the data to be stored in block 1 hello
Enter the data to be stored in block 2 txn1
Enter the data to be stored in block 3 world
Enter the data to be stored in block 4 txn2
Enter the data to be stored in block 5 txn 3
Block Hash: d218d990fdd9cd0085f1506b330bfcd1400504c218a00644dd9afc428b123e68
Previous Hash: 0
BlockNo: 0
Block Data: Genesis
Nonce: 0
Proof: 0
--------------
Block Hash: 25931154e4ea36519179ba2766e429711fc8388da55908bfa663b906282c15eb
Previous Hash: d218d990fdd9cd0085f1506b330bfcd1400504c218a00644dd9afc428b123e68
BlockNo: 1
Block Data: hello
Nonce: 4717
Proof: 00000edf2756394a2c18220634154a84245dc68b2f07a2ead03703f7903d799d
--------------
Block Hash: f7d0b8f60055ad5f18a618151a2dcb12926c7b27fb6c0ceb9d52bcd5d450d50f
Previous Hash: 25931154e4ea36519179ba2766e429711fc8388da55908bfa663b906282c15eb
BlockNo: 2
Block Data: txn1
Nonce: 1784
Proof: 00009d4c937c3efed42621581b04159dd01d619d992c0a9f90939fe729de8324
--------------
Block Hash: 96c05d476d9587d6287dd5b19d12c0136ee4a2d113123f132cbd75ca347b9c11
```

```
--------------
Block Hash: 96c05d476d9587d6287dd5b19d12c0136ee4a2d113123f132cbd75ca347b9c11
Previous Hash: f7d0b8f60055ad5f18a618151a2dcb12926c7b27fb6c0ceb9d52bcd5d450d50f
BlockNo: 3
Block Data: world
Nonce: 77690
Proof: 0000510d67f5979527e6337a8a3bd8cc8b47a7923497ffb07693f1827bbe5a09
--------------
Block Hash: 0158d06a4716b621484396bb8fffb758b23e3203deb625a15cb8236209fe6f45
Previous Hash: 96c05d476d9587d6287dd5b19d12c0136ee4a2d113123f132cbd75ca347b9c11
BlockNo: 4
Block Data: txn2
Nonce: 45170
Proof: 000033e9eb8000fcbecf4db0e6a34522dbcd0bf63bb90359001e4bc72957c5f4
--------------
Block Hash: bd251a61ad61406b3c7e0e5fb692ca97f7847d39a83c80459a84a6ae79ed17ff
Previous Hash: 0158d06a4716b621484396bb8fffb758b23e3203deb625a15cb8236209fe6f45
BlockNo: 5
Block Data: txn 3
Nonce: 104427
Proof: 00004177279b57c47f154ee57679ee700cc94ba1e9c86b68a128d86c64a0e15d
--------------
```

Part-3B: Implement Blockchain using Python

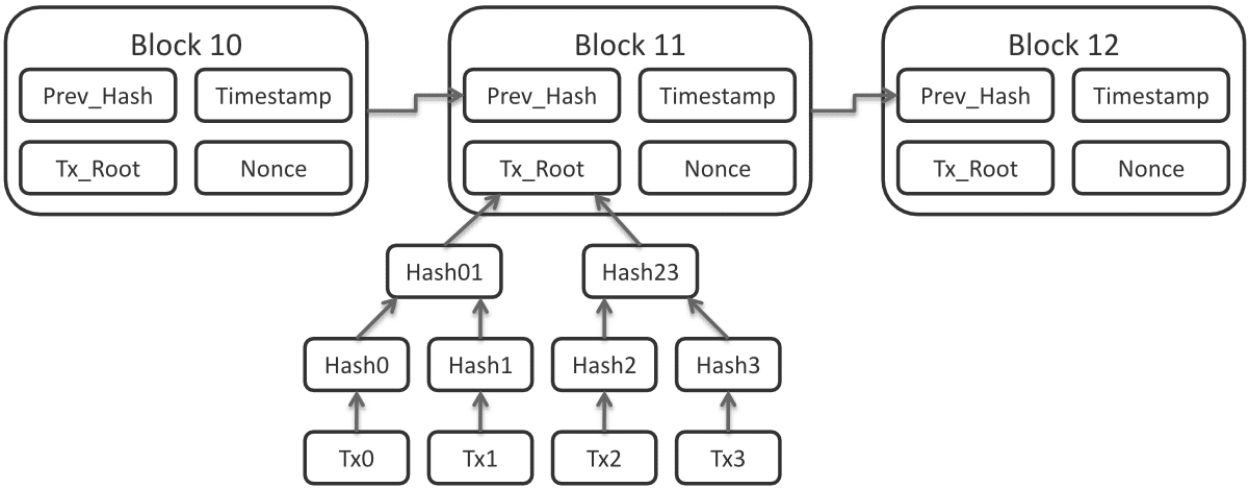

Figure-2: Blockchain Structure

Build a simple Blockchain with Block A, Block B, and Block C.

class Block:
id = None
history = None
parent_id = None

Sign data in Blockchain
Apply the Linked list to blockchain
Perform a mining process (proof-of-work) for the hash output to have at least five zeros at the front.

Code:

```
from typing import List

import typing

import hashlib



class Node:

    def __init__(self, left, right, value: str,content)-> None:

        self.left: Node = left

        self.right: Node = right
```

```
        self.value = value
        self.content = content

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

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:][0])
        self.root: Node = self.__buildTreeRec(leaves)

    def __buildTreeRec(self, nodes: List[Node])-> Node:
        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 =
self.__buildTreeRec(nodes[:half]).content+"+"+self.__buildTreeRec(nodes[half:]).content

        return Node(left, right, value,content)


    def printTree(self)-> None:

        self.__printTreeRec(self.root)


    def __printTreeRec(self, node)-> None:

        if node != None:

            if node.left != None:

             print("Left: "+str(node.left))

             print("Right: "+str(node.right))

            else:

             print("Input")

            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(elements)-> None:
    elems = elements
    mtree = MerkleTree(elems)
    return mtree



import datetime
import hashlib

class Block:
    blockNo = 0
    next = None
    hash = None
    nonce = 0
    root=Node(None, None, Node.hash(''),'')
    previous_hash = 0x0
    proof=0
    timestamp = datetime.datetime.now()
```

```
    def __init__(self, elements):
        self.root=mixmerkletree(elements)

    def hash(self):
        h = hashlib.sha256()
        h.update(
        str(self.nonce).encode('utf-8') +
        str(self.previous_hash).encode('utf-8') +
        str(self.timestamp).encode('utf-8') +
        str(self.blockNo).encode('utf-8')+
        str(self.root).encode('utf-8')
        )
        return h.hexdigest()

    def __str__(self):
        return "Block Hash: " + str(self.hash()) +"\nPrevious Hash: "+
str(self.previous_hash)+"\nBlockNo: " + str(self.blockNo) +  "\nNonce: " + str(self.nonce)
+"\nProof: "+str(self.proof)


class Blockchain:


    block = Block([''])
    dummy = head = block
```

```
    def add(self, block):

        block.previous_hash = self.block.hash()
        block.blockNo = self.block.blockNo + 1

        self.block.next = block
        self.block = self.block.next

    def proof_of_work(self, block, difficulty):
        block.nonce = 0
        computed_hash = block.hash()
        while not computed_hash.startswith('0' * difficulty):
            block.nonce += 1
            computed_hash = block.hash()
        return computed_hash

    def mine(self,new_block):
        new_block.proof = self.proof_of_work(new_block,5)
        self.add(new_block)



blockchain = Blockchain()
```

```
for n in range(3):
    elements=list(input('Enter the elements to be stored in block '+str(n+1)+' ').split(' '))
    blockchain.mine(Block(elements))

while blockchain.head != None:
    print(blockchain.head)
    print("Root Hash: "+blockchain.head.root.getRootHash()+"\n")
    print(blockchain.head.root.printTree())
    print('\n----------------------------------------')
    blockchain.head = blockchain.head.next
```

Output:

```
PS C:\Users\Nupur Gupte> python -u "c:\Users\Nupur Gupte\Desktop\files\Blockchain\merkeltree.py"
Enter the elements to be stored in block 1 tx1
Enter the elements to be stored in block 2 tx1 tx2 tx3 tx4
Enter the elements to be stored in block 3 tx1 tx2
Block Hash: d463734d5ef710dc462782a00898a6af16a90b297f2055a941b9b118349b6bfc
Previous Hash: 0
BlockNo: 0
Nonce: 0
Proof: 0
Root Hash: 3b7546ed79e3e5a7907381b093c5a182cbf364c5dd0443dfa956c8cca271cc33
```

```
----------------------------------------
Block Hash: 0623493bd51d420cef6b517ea702ca21c1ec59c6eb580cea7738aad5ac99937b
Previous Hash: d463734d5ef710dc462782a00898a6af16a90b297f2055a941b9b118349b6bfc
BlockNo: 1
Nonce: 2708037
Proof: 000005570826ccb30823594ec691aaced0650b90bd6bdf36548f1643b93b6015
Root Hash: 204dc5d3270c37eec0976d42c8e473e33a4984a5ce3dead8e89f91b4c20b3303

Left: 709b55bd3da0f5a838125bd0ee20c5bfdd7caba173912d4281cae816b79a201b
Right: 709b55bd3da0f5a838125bd0ee20c5bfdd7caba173912d4281cae816b79a201b
Value: 204dc5d3270c37eec0976d42c8e473e33a4984a5ce3dead8e89f91b4c20b3303
Content: tx1+tx1

Input
Value: 709b55bd3da0f5a838125bd0ee20c5bfdd7caba173912d4281cae816b79a201b
Content: tx1

Input
Value: 709b55bd3da0f5a838125bd0ee20c5bfdd7caba173912d4281cae816b79a201b
Content: tx1
```

```
----------------------------------------
Block Hash: f170c34fe5db5ba2c833d84bf33b376c471fc30d5e491b01f9aaa68b28777bef
Previous Hash: 0623493bd51d420cef6b517ea702ca21c1ec59c6eb580cea7738aad5ac99937b
BlockNo: 2
Nonce: 944471
Proof: 00000fa0f7179690d01f9b51e609ed3647df15e9c26d9ae8b2b70306d31a3a84
Root Hash: 773bc304a3b0a626a520a8d6eacc36809ac18c0b174f3ff3cdaf0a4e9c64433d

Left: f8f28ede979567036d801ad6cf58b551c7d8530bba005c48e46d39c73ab52664
Right: 850cf301915d09ebcfa84e2ee4087025e17a6fca7e4149ce02cff94cd3db55de
Value: 773bc304a3b0a626a520a8d6eacc36809ac18c0b174f3ff3cdaf0a4e9c64433d
Content: tx1+tx2+tx3+tx4

Left: 709b55bd3da0f5a838125bd0ee20c5bfdd7caba173912d4281cae816b79a201b
Right: 27ca64c092a959c7edc525ed45e845b1de6a7590d173fd2fad9133c8a779a1e3
Value: f8f28ede979567036d801ad6cf58b551c7d8530bba005c48e46d39c73ab52664
Content: tx1+tx2

Input
Value: 709b55bd3da0f5a838125bd0ee20c5bfdd7caba173912d4281cae816b79a201b
Content: tx1

Input
Value: 27ca64c092a959c7edc525ed45e845b1de6a7590d173fd2fad9133c8a779a1e3
Content: tx2

Left: 1f3cb18e896256d7d6bb8c11a6ec71f005c75de05e39beae5d93bbd1e2c8b7a9
Right: 41b637cfd9eb3e2f60f734f9ca44e5c1559c6f481d49d6ed6891f3e9a086ac78
Value: 850cf301915d09ebcfa84e2ee4087025e17a6fca7e4149ce02cff94cd3db55de
```

```
--------------------------------------
Block Hash: aa407ca7a7cc8cc4813ef21806cdef8af2641f4c175fd42072292e7b85ea57e1
Previous Hash: f170c34fe5db5ba2c833d84bf33b376c471fc30d5e491b01f9aaa68b28777bef
BlockNo: 3
Nonce: 36196
Proof: 00000e465430936a1741793ff91d2fd9a85b514ad910ef3878b609c0745b321e
Root Hash: f8f28ede979567036d801ad6cf58b551c7d8530bba005c48e46d39c73ab52664

Left: 709b55bd3da0f5a838125bd0ee20c5bfdd7caba173912d4281cae816b79a201b
Right: 27ca64c092a959c7edc525ed45e845b1de6a7590d173fd2fad9133c8a779a1e3
Value: f8f28ede979567036d801ad6cf58b551c7d8530bba005c48e46d39c73ab52664
Content: tx1+tx2

Input
Value: 709b55bd3da0f5a838125bd0ee20c5bfdd7caba173912d4281cae816b79a201b
Content: tx1

Input
Value: 27ca64c092a959c7edc525ed45e845b1de6a7590d173fd2fad9133c8a779a1e3
Content: tx2

None

--------------------------------------
```

Conclusion:

Thus implemented a simple blockchain in python language. Understood the structure of a block in blockchain. Also, I performed the proof of work where it is checked for each nonce value whether we get the required number of zeros at the beginning of the hash output or not. Implemented the merkle tree in python. The merkle tree is fast, efficient, requires less storage space and is verifiable. It can be verified by checking whether root hash is valid or not. If invalid then we traverse down the tree.

References:

[1] https://www.softwaretestinghelp.com/blockchain-tutorial/

[2] https://coincentral.com/what-is-a-nonce-proof-of-work/

[3] http://blockchain-basics.com/HashPuzzle.html