CS764_Homework_6

February 26, 2020

1 Blockchains and Cryptocurrencies

1.1 Homework 6

1.2 Joseph S Cannella

Q1. [5 points] Ethereum employs KECCAK256 hash algorithm to compute hashes in the modified MerklePatricia trie. Determine the KECCAK256 hash of the following root note of a Merkel tree. The hexa string to be hashed is "0B8FC549A" (Note: Here, all arehexa characters.) It is the same as "0b8fc549a" if you want to correlate with the notation in the below figure.These are NOT ASCII characters.

[60]:
```python
# Note: SHA3 is the official name of KECCAK 256
import hashlib, binascii
import math
import pandas as pd
hexInput = 0x0B8FC549A# hex values

# Much of what we need reiles on converting this to character array
# NOTE: the bytes will remain the same, jut the representation changes
def hexToString(hexVal):
    numBytes = math.ceil(math.log(hexVal, 256))
    prevByte = 0
    charArray = ""
    for i in range(numBytes, -1, -1):
        currentByte = hexVal>>(8*i)
        prevByte = currentByte<<(8*i)
        hexVal = hexVal - prevByte
        print(hex(currentByte))
        charArray += chr(currentByte)
    return charArray
# Calculate Character Array Representation of Hex String
chrArray = hexToString(0x0B8FC549A)
# Verification
inputStr = chr(0x0) + chr(0xB8) +  chr(0xFC) + chr(0x54) +  chr(0x9A)
if(chrArray == inputStr):
    print("Calcualtion Verified!")

# Compute Hash using SHA3
```

```
s = hashlib.sha3_256()
s.update(chrArray.encode())
print(f'Computed Hash: {s.hexdigest()}')
```

```
0x0
0xb8
0xfc
0x54
0x9a
Calcualtion Verified!
Computed Hash: 4fe3f0f1badb26168c66bd23ab36206fd90abd30a762564db07ce733e4830588
```

Q2. [15 points] As shown in the below figure, modified Merkle Patricia tries in Ethereumare used to store the world state. Here, the tree represents 4 given accounts (shown in the Simplified World State). Give the following 6 accounts, with account# being the key expressed as a hexa character string. For simplicity, account# is represented as a 8-character string. In reality, it is 40 characters or 20 bytes in length.

1. Construct a Merkle tree with these 6 accounts. Employ SHA-256 for hashing within the Merkel tree.

2. Construct a Patricia tree with these 6 accounts. Consider the address as a string of hexa characters. (iii)Construct a modified Merkle-Patricia tree (similar to the one in the below figure).

Account# (in hexa)	Account balance (in Ether)	Number of transactions
b35023b1	250.256	108
b57d46e8	4500.4798	213
b57690a1	367.90	578
d9a545b2	70013.256	1023
d9a7d235	678.23	651
d9a7d456	78.00	25

3. Compare the three implementations and comment why Ethereum inventor proposed the modified Merkle-Patricia tree. First, create a transaction class for the individual transactions.

[17]:
```
# Account Class
class Account:
    accountNum = 0
    balance = 0
    numTrans = 0
    def __init__(self, accountNum, balance, numTrans):
        self.accountNum = accountNum
```

```
        self.balance = balance
        self.numTrans = numTrans
```

Now that we have created our basic classes we begin by constructing a list of transactions.

[18]:
```
accounts = []
accounts.append(Account(0xb35023b1, 250.256, 108))
accounts.append(Account(0xb57d46e8, 4500.4798, 213))
accounts.append(Account(0xb57690a1, 367.90, 578))
accounts.append(Account(0xd9a545b2, 70013.256, 1023))
accounts.append(Account(0xd9a7d235, 678.23, 651))
accounts.append(Account(0xd9a7d456, 78.00, 25))

for a in accounts:
    print(f'Account#: {a.accountNum}, Balance: {a.balance}, # Transactions {a.
 ↪numTrans}')
```

```
Account#: 3008373681, Balance: 250.256, # Transactions 108
Account#: 3044886248, Balance: 4500.4798, # Transactions 213
Account#: 3044446369, Balance: 367.9, # Transactions 578
Account#: 3651487154, Balance: 70013.256, # Transactions 1023
Account#: 3651654197, Balance: 678.23, # Transactions 651
Account#: 3651654742, Balance: 78.0, # Transactions 25
```

(i) Construct Merkle Tree implementing SHA-256

[71]:
```
# Refference: https://www.youtube.com/watch?v=GaFuBrkkI_w
import json
class MerkleTree:
    def __init__(self, accounts):
        self.accounts = accounts
        self.hashes = []
        self.getInitialHashes()

    def getInitialHashes(self):
        # loop over each account and determine its hash
        for a in self.accounts:
            accountStr = f'{a.accountNum}:{a.balance}:{a.numTrans}'
            self.hashes.append(hashlib.sha256(accountStr.encode()).hexdigest())

        # Recursively Build the Tree
        self.buildTree(self.hashes)


    def buildTree(self, previousHashes):
        # We need an even number of hashes each time
        if(len(previousHashes) %2 != 0):
            # If not even make even by adding appennding laast hash again
```

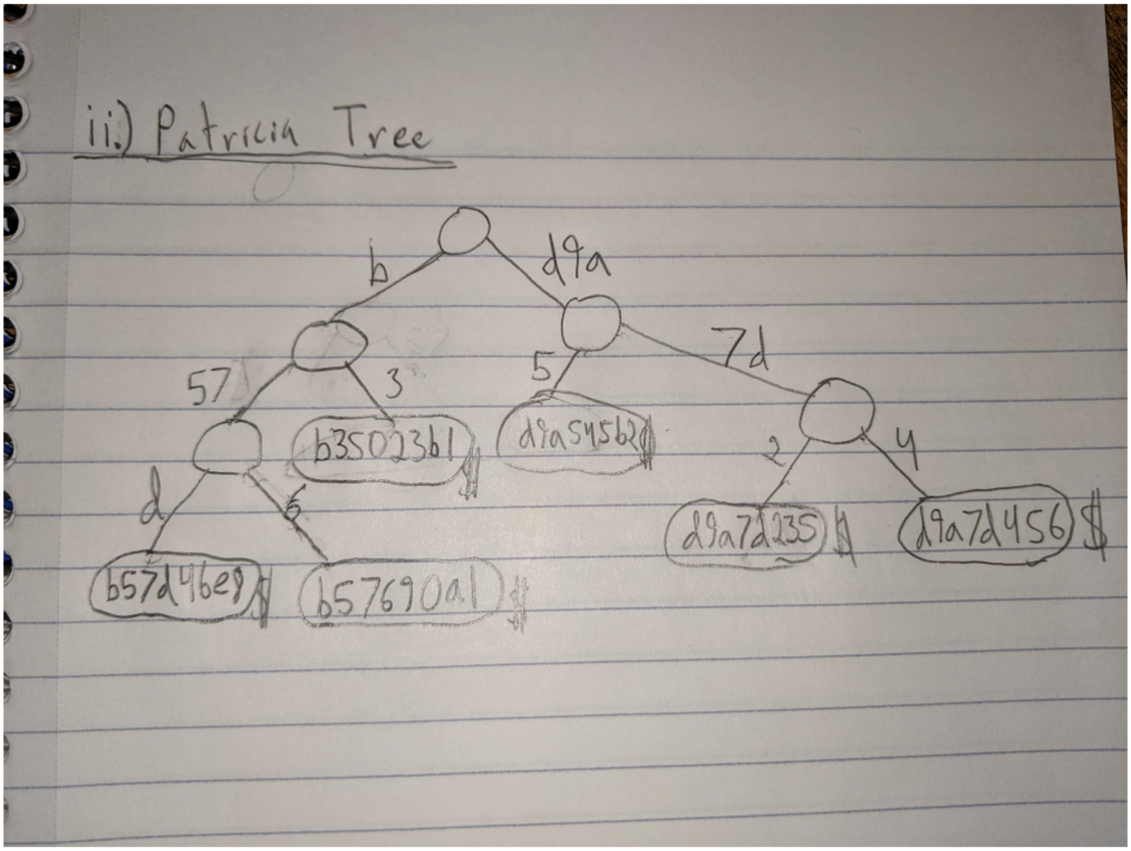
```python
            previousHashes.append(previousHashes[-1])

        # Keep a list or the new hases we will generate from the old ones
        newHashes = []

        # Loop over hashes in pairs of two hashing them together
        for i in range(0, len(previousHashes), 2):
            newStr = previousHashes[i] + previousHashes[i+1]
            newHash = hashlib.sha256(newStr.encode()).hexdigest()
            newHashes.append(newHash)
            self.hashes.append(newHash)

        # Check if we need to continue recursion (only one element)
        if(len(newHashes) > 1):
            self.buildTree(newHashes)
```

New that the tree class has been created. Lets populate, view it's entries, and asses its properties.

[83]:
```python
myMerkleTree = MerkleTree(accounts)
# Pandas would print nicer, but isnt supported by nbconvert
merkleDF = pd.DataFrame(myMerkleTree.hashes, columns=['Merkle Tree Hashes'])
myMerkleTree.hashes
```

[83]:
```
['836dc5a1e3dcfc064f62cff1da53500670d83e4a38d6e30c0cd1be3b5c83489b',
 'ac591ad1bbe41e02f38486b36128b294c3a8c56f908b40c39296f7989bb53568',
 'e6dd8393904cbb3d771a9db9d009959ccfaea2be159be7a6f7780a3901d0f2c0',
 '2413ad322568e89ee206bad47847d19d74802bf095a4089256266751399dec6c',
 '53e679861eaf673aafd1a5ad2df2354704c4631bbee76cfe2e4280df75532fca',
 '4e3041e733551c62581cf2df7a1aa81620bb8547906c4d0efd3a8500673bff4e',
 'd7778607e9f5b63c4b1789a6626eeb5401e983e1e4127a7bf50eb30c423092e3',
 'bcb0d4036dd808b0c6f985375004113b07da6269afd9132a3f0667ef2ed41b09',
 'aedbdbc17aaccfe2a03050678dffdb97f83d41c77e7d16a1420ef381ba466ad6',
 '6fc1b66014de9bf42e18f1ec91b09e4e2099fedbfd7553976dce3f05987d8cde',
 '836ec3ac8fc12a5e1bd08ef10369987e2fed7ac2d500f29c00ce17133a86aeba',
 '5a322ca30a21ecfd444701c434d445cfb3135ec9435122932c63dc76bd042a25']
```

(ii) Construct a Patricia Tree



Solution:

(iii) Construct a modified Merkle-Patricia Tree

Approach: We require three types of nodes to build modified Merkle Patricia Tree 1. Leaf Nodes - containing the actual value of a transaction 2. Extension Nodes - basically empty links that jsut point to the node 3. Branch Nodes - basically a 16 elemennt array or pointers to children nodes

[106]:
```python
# Leaf Node Class
class LeafNode:
    keyEnd = 0
    value = 0
    def __init__(self, keyEnd, value):
        self.keyEnd = keyEnd
        self.value = value

# Extension Node Class
class ExtensionNode:
    sharedNibbbles = 0
    nextNode = 0
    def __init__(self, sharedNibbles, nextNode):
        self.sharedNibbles = sharedNibbles
        self.nextNode = nextNode
```

```
# Branch Node Class
class BranchNode:
    address = [None for i in range(16)]
    def __init__(self, key):
        self.key = key
```

Notes: In this case because we have two distinct leading hex chars we will have two distinct roots and thus two distinct blocks. In this case both of these are extension nodes.

[107]:
```
# We First Construct out Leaf Nodes
A = accounts
accLen = 8# Number of hex chars in account number
leafs = []

# BLock 1
leaf1 = LeafNode(hexToString(A[0].accountNum)[-(7):], A[0].balance)# Keeps last␣
 ↪7 digits of account number
leaf2 = LeafNode(hexToString(A[1].accountNum)[-(3):], A[1].balance)# Keeps last␣
 ↪3 digitis of account number
leaf3 = LeafNode(hexToString(A[2].accountNum)[-(3):], A[2].balance)# Keeps last␣
 ↪3 digitis of account number
branch1 = BranchNode(hexToString(A[1].accountNum)[1:3])# Keeps digits [1:3] of␣
 ↪account number
branch1.address[0x4] = leaf2# Add leaf 2 to index 4 of extension nodes array
branch1.address[0x9] = leaf3# Add leaf 3 to index 9 of extension nodes array
branch2 = BranchNode(hexToString(A[1].accountNum)[0])# Root node only contains␣
 ↪the first n digist of account number (in this case oth)
branch2.address[0x3] = leaf2# Add leaf 2 to index 4 of extension nodes array
branch2.address[0x9] = leaf3# Add leaf 3 to index 9 of extension nodes array
```

```
0x0
0xb3
0x50
0x23
0xb1
0x0
0xb5
0x7d
0x46
0xe8
0x0
0xb5
0x76
0x90
0xa1
0x0
```

```
0xb5
0x7d
0x46
0xe8
0x0
0xb5
0x7d
0x46
0xe8
```

(iv) Compare the three implementations and comment why Etherium inventor proposed the modified Merkle-Patricia tree. The modified Merkle Patricia Tree has compression properties of the Patricia tree (it is optimized in size). This is due to the use of branch and extension nodes, making the tree quickly searchable. While the Merkle Patricia Tree allows for compace and secure storage of the world state it should be noted that only the root node's key is hashed. This differes from a Merkle tree where nodes are recursivley hased together in pairs untill a root is created. Due to it's compact nature and security the Merkle Patricia Tree makes a good choice for maintaining validating block within a cryptocurrency.