FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING
Department of Computer Engineering

1. Course , Subject & Experiment Details

Academic Year	**2022-23**	**Estimated Time**	**02 - Hours**
Course & Semester	**B.E. (CMPN)- Sem VII**	**Subject Name & Code**	**BCT - (CSDC7022)**
Chapter No.	**01**	**Chapter Title**	**Introduction to Blockchain**

Practical No:	**2**
Title:	Construction of Merkle Tree
Date of Performance:	**08/08/2022**
Date of Submission:	**15/08/2022**
Roll No:	**8953**
Name of the Student:	**Brendan Lucas**

Evaluation:

Sr. No	Rubric	Grade
1	On time submission Or completion (2)	
2	Preparedness(2)	
3	Skill (4)	
4	Output (2)	

Signature of the Teacher:

Date:

Code:

Transaction.js:

```
const sha256 = require("./helper");

class Transaction {
    constructor(to, from, amount) {
        this.to = to;
        this.from = from;
        this.amount = amount;
        this.id = Transaction.getCount();
        this.hash = sha256(this.to + this.from + this.amount + this.id);
        Transaction.incrementCount();
    }

    static getCount() {

        return Transaction.count;
    }

    static incrementCount() {
        Transaction.count++;
    }

    getHash() {
        return sha256(this.to + this.from + this.amount + this.id);
    }

    toString() {
        return `
        to:${this.to}
        from:${this.from}
        amount:${this.amount}
        hash:${this.hash}
        id:${this.id}`;
    }
}

Transaction.count = 0;

module.exports = Transaction;
```

TransactionList.js:

```
class TransactionList {
    constructor() {
        this.list = [];
    }

    add(transaction) {
        this.list.push(transaction);
    }
}

module.exports = TransactionList;
```

test.js:

```
const MerkelTree = require("./MerkelTree");
const TransactionList = require("./TransactionList");
const Transaction = require("./Transaction");
const util = require("util");

let transactionList = new TransactionList();

for (let index = 0; index < 5; index++) {
    transactionList.add(new Transaction(Math.random(), Math.random(),
Math.random()));
}

const tree = new MerkelTree();

tree.createTree(transactionList.list);
console.log(util.inspect(tree, false, null, true /* enable colors */));
tree.verify(transactionList.list[2]);

// Lets tamper the data

transactionList.list[2].to = "malicious address";
transactionList.list[2].amount = 1000;
console.log(util.inspect(transactionList, false, null, true /* enable colors
*/));
tree.verify(transactionList.list[2]);
```

MerkelTree.js:

```
const sha256 = require("./helper");
const util = require("util");

//TODO: Add comments

class MerkelTree {
    constructor() {
        this.root = [];
    }
    /**
     * Takes a list of transaction as input and
     * @param {TransactionList} transactionList
     */
    createTree(transactionList) {
        this.root.unshift(transactionList);
        this.root.unshift(transactionList.map(t => t.hash));

        while (this.root[0].length > 1) {
            let temp = [];

            for (let index = 0; index < this.root[0].length; index += 2) {
                if (index < this.root[0].length - 1 && index % 2 == 0)
                    temp.push(sha256(this.root[0][index] + this.root[0][index +
1]));
                else temp.push(this.root[0][index]);
            }
            this.root.unshift(temp);
        }
    }

    verify(transaction) {
        let position = this.root.slice(-1)[0].findIndex(t => t.hash ==
transaction.hash);
        console.log(position);
        if (position) {

            let verifyHash = transaction.getHash();
```

```
            for (let index = this.root.length - 2; index > 0; index--) {

                let neighbour = null;
                if (position % 2 == 0) {
                    neighbour = this.root[index][position + 1];
                    position = Math.floor((position) / 2)
                    verifyHash = sha256(verifyHash + neighbour);
                }
                else {
                    neighbour = this.root[index][position - 1];
                    position = Math.floor((position - 1) / 2)
                    verifyHash = sha256(neighbour + verifyHash);
                }

            }
            console.log(verifyHash == this.root[0][0] ? "Valid" : "Not Valid");
        }
        else {
            console.log("Data not found with the id");

        }
    }
}

module.exports = MerkelTree;
```

helper.js:

```
const crypto = require("crypto");

// Hashes data and returns a hex string
function hash(data) {
    return data != null
        ? crypto
            .createHash("sha256")
            .update(data.toString())
            .digest("hex")
        : "";
}

// Export the function
```

```
module.exports = hash;
```

file1.js:

```
const { MerkleTree } = require('merkletreejs')
const SHA256 = require('crypto-js/sha256')

const leaves = ['a', 'b', 'c'].map(x => SHA256(x))
console.log("-------------------------------------")
console.log("Leaves Hashes")
console.log(leaves)
console.log("-------------------------------------")
const tree = new MerkleTree(leaves, SHA256)
const root = tree.getRoot().toString('hex')
console.log("-------------------------------------")
console.log("Merkle Root")
console.log(root)
console.log("-------------------------------------")
const leaf = SHA256('a')
console.log("-------------------------------------")
console.log("Hash of Leave 'a'")
console.log(leaf)
console.log("-------------------------------------")
const proof = tree.getProof(leaf)
console.log("-------------------------------------")
console.log("Is 'a' a valid leaf of the tree")
console.log(tree.verify(proof, leaf, root)) // true
console.log("-------------------------------------")


const badLeaves = ['a', 'b', 'c','e'].map(x => SHA256(x))
console.log("\n\n\n\n\-------------------------------------")
console.log("Tree 2 Leaves Hashes")
console.log(badLeaves)
console.log("-------------------------------------")
const badTree = new MerkleTree(badLeaves, SHA256)
const badRoot = badTree.getRoot().toString('hex')
console.log("-------------------------------------")
console.log("Merkle Root")
console.log(badRoot)
console.log("-------------------------------------")
```

```
const badLeaf = SHA256('d')
console.log("------------------------------------")
console.log("Hash of Leave 'd'")
console.log(badLeaf)
console.log("------------------------------------")
const badProof = badTree.getProof(badLeaf)
console.log("------------------------------------")
console.log("Is 'd' a valid leaf of the tree")
console.log(badTree.verify(badProof, leaf, root)) // false
console.log("------------------------------------")
```

Output:

```
PS C:\React_WS\blockchain_pracs\exp_1> node file1.js
---------------------------------------
Leaves Hashes
[
  {
    words: [
      -896040686,
      -904151606,
      -87936589,
      -1708925875,
      -1484328968,
      343690866,
      -1182763131,
      -1343338309
    ],
    sigBytes: 32
  },
  {
    words: [
       1042540566,     3758410,
        864636773,  1692512564,
      -1950516736, -1999360950,
       -881594706,  -711196515
    ],
    sigBytes: 32
  },
  {
    words: [
      779955203,
      -1454343454,
      1710028213,
      896042405,
      865313282,
      -1658580076,
      -1720556127,
      -1571098682
    ],
    sigBytes: 32
  }
]
---------------------------------------
---------------------------------------
Merkle Root
7075152d03a5cd92104887b476862778ec0c87be5c2fa1c0a90f87c49fad6eff
---------------------------------------
---------------------------------------
Hash of Leave 'a'
{
  words: [
    -896040686,
    -904151606,
```

```
    -87936589,
    -1708925875,
    -1484328968,
    343690866,
    -1182763131,
    -1343338309
  ],
  sigBytes: 32
}
------------------------------------
------------------------------------
Is 'a' a valid leaf of the tree
true
------------------------------------




------------------------------------
Tree 2 Leaves Hashes
[
  {
    words: [
      -896040686,
      -904151606,
      -87936589,
      -1708925875,
      -1484328968,
      343690866,
      -1182763131,
      -1343338309
    ],
    sigBytes: 32
  },
  {
    words: [
       1042540566,     3758410,
        864636773,  1692512564,
      -1950516736, -1999360950,
       -881594706,  -711196515
    ],
    sigBytes: 32
  },
  {
    words: [
      779955203,
      -1454343454,
      1710028213,
      896042405,
      865313282,
      -1658580076,
      -1720556127,
```

```
      -1571098682
    ],
    sigBytes: 32
  },
  {
    words: [
      1064942459,
      1130038578,
      374463215,
      -747319866,
      -2116286726,
      -1503774898,
      942898056,
      -905679382
    ],
    sigBytes: 32
  }
]
---------------------------------------
---------------------------------------
Merkle Root
d9d99d8e1dd54bf3af669795a54f4bce6e6008093c074ca1d1b7bb4ffb7c0de1
---------------------------------------
---------------------------------------
Hash of Leave 'd'
{
  words: [
     413941363, 1139807881,
     206638739, -113957359,
    1775887349, 1698915369,
   -2096124151,  888469732
  ],
  sigBytes: 32
}
---------------------------------------
---------------------------------------
Is 'd' a valid leaf of the tree
false
---------------------------------------
```