Block Chain MATLAB Implementation

Key Words : Block Chain; MATLAB; Object-Oriented Programming

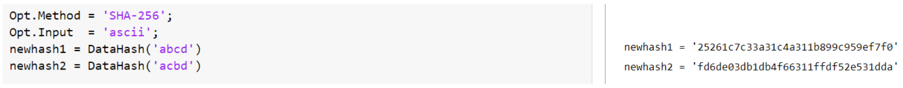
1. Block Chain Theory

1. A block is actually a data structure which can be used to store any type of data. In most practical applications of the blockchain, the stored data is usually **Transaction Data**. In MATLAB, we can use a class to represent a block.

2. In the definition of the Block class, the attributes include index (i.e., block number), data (i.e., transaction data), selfHash (i.e., the block's own hash value), previousHash (i.e., the hash value of the previous block), and nounce(i.e., mining times, it is a random value).

3. The block's function can accept 2 or 3 parameters. When two parameters are provided, the constructed Block object is called a Genesis Block, which is the first block on the entire blockchain; when it accepts three parameters, the third parameter is the hash value from the previous block. The implement details can be found in **2.1 block.m**.

4. Hash algorithm can be understood as a mapping algorithm, which **maps a string of characters to another fixed-length string**, as shown follows

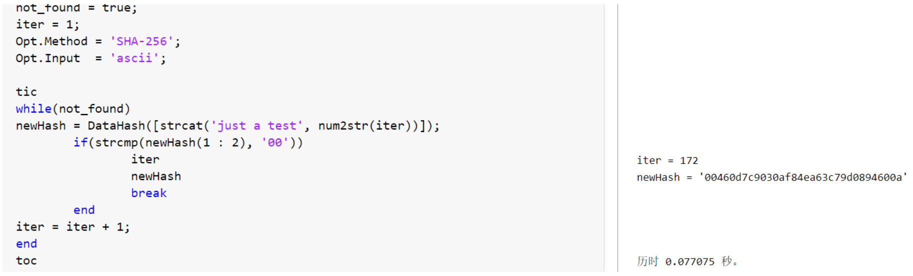


```
Opt.Method = 'SHA-256';
Opt.Input  = 'ascii';
newhash1 = DataHash('abcd')
newhash2 = DataHash('acbd')
```

```
newhash1 = '25261c7c33a31c4a311b899c959ef7f0'
newhash2 = 'fd6de03db1db4f66311ffdf52e531dda'
```

Figure 1: In this implementation we use SHA-256.

5. Let's consider a problem: If the first two digits of newHash begin with 00, we should find out what the input string can be. Note that we only specified the first two digits of newHash, not all digits. There may be many inputs that meet this requirement, but still because even if the output is known, it is difficult to find the input in reverse, so we can only use Brute-Force Exhaustion. For example,we calculate the hash value of 'abcd' . If it does not meet the requirements(i.e., the first two digits of newHash begin with 00), try the next integer until the first two digits of newHash are 00. Let's see two examples.



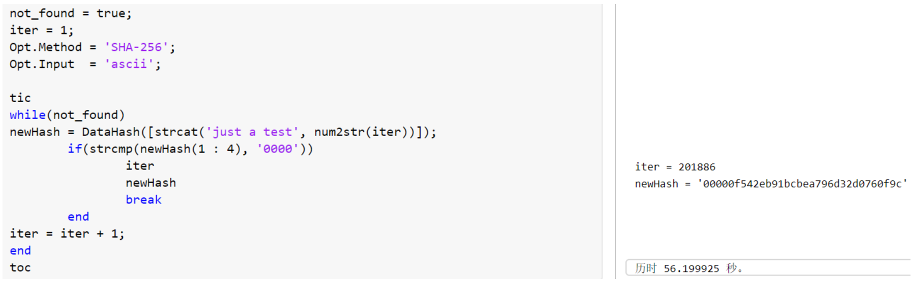
```
not_found = true;
iter = 1;
Opt.Method = 'SHA-256';
Opt.Input  = 'ascii';

tic
while(not_found)
newHash = DataHash([strcat('just a test', num2str(iter))]);
        if(strcmp(newHash(1 : 2), '00'))
                iter
                newHash
                break
        end
iter = iter + 1;
end
toc
```

```
iter = 172
newHash = '00460d7c9030af84ea63c79d0894600a'

历时 0.077075 秒。
```

Figure 2: the first two digits of newHash begin with 00.



```
not_found = true;
iter = 1;
Opt.Method = 'SHA-256';
Opt.Input  = 'ascii';

tic
while(not_found)
newHash = DataHash([strcat('just a test', num2str(iter))]);
        if(strcmp(newHash(1 : 4), '0000'))
                iter
                newHash
                break
        end
iter = iter + 1;
end
toc
```

```
iter = 201886
newHash = '00000f542eb91bcbea796d32d0760f9c'

历时 56.199925 秒。
```

Figure 3: the first two digits of newHash begin with 000.

6. This exhaustive method to find the hash that satisfies the conditions is the essence of mining. Since there is no connection between each loop, these operations can be **parallelized**. This is the nature of the mining machine. **The more strict the requirements for the initial characters of newHash, the more difficult that mining is, and the more time it takes**. We can compare Figure 2 to 3 to verify it.

2. MATLAB Implementation

2.1 Block.m

Program 2.1: Block

```
classdef Block < handle

  properties
  index % index of block
  data % transcation data
  previousHash % the previous hash
  selfHash % current hash
  nonce % random number
  end

  methods
  function obj = Block(index, data, previousHash)
    if nargin == 2 % genesis block!
      obj.index = index ;
      obj.data = data ;
    elseif nargin == 3
      obj.index = index ;
      obj.data = data ;
      obj.previousHash = previousHash;
    end
  end

  % The function below converts all data on the block except 'nonce' and
  % 'selfHash' into characters, which is then used to calculate selfHash.
  function str = getCombined(obj)
    str = strcat([num2str(obj.index), obj.previousHash, join(obj.data)]);
  end
  end
end
```

Listing 1: Block.m

Output 2.1:

```
>> Block(1, ’this is data’)

ans =
  Block - properties:

           index: 1
            data: 'this is data'
    previousHash: []
        selfHash: []
           nonce: []

>> Block(1, 'this is data', 'this is previous hash')

ans =

  Block - properties:

           index: 1
            data: 'this is data'
    previousHash: 'this is previous hash'
        selfHash: []
           nonce: []
```

2.2 BlockChain.m

Program 2.2: BlockChain

```
classdef BlockChain < handle

  properties
  totalCount % used to record the number of blocks
  blockArray % this is an object array that used to store the blockchain
  end

  methods
  function obj = BlockChain()
    obj.blockArray =[Block(0, 'Genesis Block')]; % genesis block
    obj.totalCount = 1 ;
    obj.calculateGensisBlockHash(); % calculate the hash of genesis block
  end

  function bc = getLatest(obj) % get the last block on the current blockchain
    bc = obj.blockArray(end);
  end

  function calculateGensisBlockHash(obj)
    gb = obj.blockArray(1);
    Opt.Method = 'SHA—256';
    Opt.Input  = 'ascii';
    str = strcat(num2str(gb.index), gb.data);
    disp(str);
    gb.selfHash = DataHash(str, Opt); % calculate current hash
  end

  function addBlock(obj, newBlock) % when Miner.m successfully 'digs out' a block that meets the requirements
    if  obj.validateNewBlock(newBlock) % call this function
      obj.blockArray(end+1) = newBlock; % and then add this block to this blockchain
    end
  end

  function tf = validateNewBlock(obj, newBlock) % verify that the newly added block meets the requirements or not.
    newHash = DataHash([strcat(newBlock.getCombined(), num2str(newBlock.nonce))]);
    if(strcmp(newHash(1:3), '000') && strcmp(newBlock.selfHash, newHash))
      tf=  true;
    else
      tf = false;
    end
  end
  end
end
```

Listing 2: BlockChain.m

Output 2.2:

```
>> BlockChain
0Genesis Block

ans =

  BlockChain - properties:

    totalCount: 1
    blockArray: [1×1 Block]
```

2.3 Miner.m

Program 2.3: Miner

```
classdef Miner < handle
  properties
  blockchain
  end

  methods
  function obj = Miner(blockchain)
    obj.blockchain = blockchain;
  end

  function mine(obj, newData)
    % get the last block on the current blockchain
    latestBlock = obj.blockchain.getLatest();
    % construct a new block
    newBlock = Block(latestBlock.index+1,...
    newData,...
    latestBlock.selfHash);% find appropriate selfhash
    not_found = true;
    iter = 1;
    Opt.Method = 'SHA—256';
    Opt.Input  = 'ascii';

    tic
    while(not_found)
    newHash = DataHash([strcat(newBlock.getCombined(), num2str(iter))]);
      if(strcmp(newHash(1 : 3), '000'))
        newBlock.nonce = iter; % solve violently
        newBlock.selfHash = newHash; % if the approproate selfhash is found
        disp(newHash)
        obj.blockchain.addBlock(newBlock); % add selfhash to blockchain
        break
      end
    iter = iter + 1;
    end
    toc
  end
  end
end
```

Listing 3: Miner.m

2.4 TradingTest.m

Program 2.4: TradingTest

```
clear; clc;
bc = BlockChain;
bc; bc.blockArray(1)
mining = Miner(bc);
disp('============================================');
transcation = ['A', 'B', 'MOP', '200'];
mining.mine(transcation)
bc; bc.blockArray(2)
disp('============================================');
transcation = ['B', 'C', 'USD', '300'];
mining.mine(transcation)
bc; bc.blockArray(3)
disp('============================================');
transcation = ['C', 'A', 'HKD', '700'];
mining.mine(transcation)
bc; bc.blockArray(4)
```

Listing 4: BlockChain.m

Output 2.4:

```
0Genesis Block

ans =

  Block - properties:

           index: 0
            data: 'Genesis Block'
    previousHash: []
        selfHash: '075c27741a3506846368fa6e5b3477f85b31ceee71a5
                   716e2f12b40fa21d23aa'
           nonce: []

============================================
000cfbb745e3d504306b8c435b639d1d
It took 0.562127 seconds.

ans =
  Block - properties:

           index: 1
            data: 'ABMOP200'
    previousHash: '075c27741a3506846368fa6e5b3477f85b31ceee71a5
                   716e2f12b40fa21d23aa'
        selfHash: '000cfbb745e3d504306b8c435b639d1d'
           nonce: 1209

============================================
0008fc36bf8a3fac06b898239c5f6ff5
It took 0.114654 seconds.

ans =

  Block - properties:

           index: 2
            data: 'BCUSD300'
    previousHash: '000cfbb745e3d504306b8c435b639d1d'
        selfHash: '0008fc36bf8a3fac06b898239c5f6ff5'
           nonce: 292

============================================
000b3d4205a9fcd798805f004e8d9a75
It took 0.946117 seconds.

ans =

  Block - properties:

           index: 3
            data: 'CAHKD700'
    previousHash: '0008fc36bf8a3fac06b898239c5f6ff5'
        selfHash: '000b3d4205a9fcd798805f004e8d9a75'
           nonce: 2940
```