Assignment #2: Programming a BlockChain

CSI 2110 – Data Structures and Algorithms

Fall 2018

School of Information Technology and Engineering

Course Coordinator: Professor Lucia Moura

Graydon Hope - 300045044
Assignment Due Date: October 22, 2018
Miner ID: ghope049

Class and Method Descriptions:

BlockChain: The BlockChain class is the database in which all transactions are recorded and stored. This class contains objects of the Transaction, Block, and Sha1 classes. The main role of the block chain is to store all the past bitCoin transactions in the "chain". This chain contains information on each individual block that was added. These blocks contain all required information of the transaction that occurred. One of the key pieces of information held in these blocks is the cryptographic hash. This is string of characters that uses Sha1 algorithm to produce a 160-bit long hash code. This is used so that malicious users cannot access or modify the block/transaction information.

In my BlockChain class, I generate the nonce and hash, then send all information into the block constructor. This means the nonce generation is occurring inside the blockchain class. The BlockChain class also accesses text files to read other predetermined block chain information and is then able to modify the chain (by adding transactions) and writing that information to an existing or new file.

To execute the program, I created a start up method which gets called from the main. This prompts the user which file to read the block chain values from. After the system reads the block chain information, it moves on to the validation. In the validation method, the system ensures the hashes are correct using the Sha1 algorithm, that the previous hash is correct, and that each index is valid.

Block: The Block class is used to create each individual block that gets added to the block chain. This requires Transaction information, a valid nonce value, and the hash values. To keep the chain secure, we use a timestamp, transaction, nonce, and previous hash all concatenated and then sent into the Sha1 algorithm to create the new hash code for that specific block. Each block contains a hash and previous hash value to link all the blocks together. This is how block chains are secure. To modify one block, you need the encrypted information of the previous block, which requires the information of the block before that in the chain, and so on.

The Block class is mostly made up of "setting" and "getting" methods. When constructing a block, you must provide all information that blocks require. The block class also contains a toString method which returns what will be sent to the Sha1 encryption algorithm.

Transaction: In our case (for simplicity), we only store one transaction per block. Each transaction object stores the name of the sender, receiver, and the amount of the transaction. These objects are required for each block.

In the transaction class, there is also mostly "setting" and "getting" methods. These give access to the sender, receiver, and amount of each transaction.

Sha1: Our blocks use Sha1 encryption. Currently, bitCoin uses Sha256-bit encryption. For the scope of our assignment, Sha1 is enough encryption to understand the concept of producing cryptographic hashes. However, this would not be an algorithm to use in any real application (lack of security).

Proof of Work Algorithm:

BitCoin requires that each valid hash code must start with a certain number of 0's. In our assignment, this number is 5. We concatenate the rest of our block information with a nonce. The value of the nonce changes until the Sha1 encryption algorithm produces a hash code beginning with 5 consecutive 0's.

My nonce producing algorithm (which can be found in the generateNonce method), uses an iterative approach. By nesting 4 for loops, and using ASCII characters in the range of 33-127, the algorithm produces a maximum of 73188024 permutations (permutation of 94 options in length 4). This will keep changing one value of the nonce (of length 4) until it finds a suitable hash code.

I chose this method as opposed to the "random selection" route to ensure I find a valid hash, and to avoid duplicates. I understand the probability of this occurring is very low, however when you randomly select nonce character values, you run the risk of never finding a valid code. Because the CSI 2110 class does not know how to "crack" the Sha1 algorithm, we essentially have to "guess and check" our generated nonce values. When you are going to go through all these possibilities, it is a safer approach to systematically increment the value of each index instead of a random character generation which may overlap previous "guesses". The approach that I used may not find a nonce which makes the hash valid as quickly as the random generation, but it has more of a guarantee to find one in general.

Hash Trial Statistics Table

Transaction Number	**Nonce Generation Attempts**
1	1122820
2	72109
3	636038
4	194617
5	2825056
6	15997
7	3060822
8	59760
9	222885
10	873495

The average number of nonce generation trials is 908359.9 (sum of 10 transactions / 10). The highest amount of trials the generation took was 3060822, leaving many permutations available before the total amount has been produced.