ClaraCoin: a rudimentary cryptocurrency

ClaraCoin is a rudimentary but functional cryptocurrency system, modelled upon Bitcoin. Any member of the public can both make transactions and mine blocks. The entire process is centered around a server hosted at `nasonfish.com`.

Note: if ClaraCoin was more popular, an adventurer could download the blockchain from any other adventurer and use custom built mining and transaction software. Because ClaraCoin is a weekend project developed by three undergrad students, we decided to become our own third parties. Miners connected to `nasonfish.com` provide the longest blockchain and files for both mining and transactions.

Initial setup

1. An adventurer goes to `claracoin.nasonfish.com` to downloads the client code that they will use to make transactions and mine blocks.
2. User connects to the `nasonfish.com` server and downloads a recent version of the blockchain.

Making transactions

1. An adventurer runs the script "`simulate_transaction.sh`" and provides the interface with their public key, private key, recipient of their transfer, and amount to be transfered.

Mining blocks

1. An Adventurer connects to the server to get new transactions. They run the script "`run_miner.sh`", which loads the transactions, verifies them (just like in the initial setup), and piles them together into a block, and generates a new magic number until the hash value of the new block is satisfactory. We define a satisfactory hash value as one that ends with the four hexadecimal characters "cccc".
    a. "`run_miner.sh`" iterates through the whole chain and makes sure that each and every transaction is valid. Each block has a valid hash value, and that the hash value of each block matches the "previous block" field of the following block in the chain.

Reclaiming disk space (iceboxed)

1. When the blockchain stored on an adventurer's computer becomes large, they have the option to reduce its size by pruning transactions from blocks that have been fully spent (that is, all of whose outflows have been spent).

See README.md in our repository for specific instructions on how to run and test the software.
https://github.com/nasonfish/ClaraCoin

Some discussion on an iceboxed interest of ours:

What problem do we solve by using a Merkle tree to store transactions within a block (rather than using a list)?

As the chain continues to grow, its size will become an inconvenience. To reduce the chain's size, it is important to remove spent transactions from their blocks. Complete removal of spent transactions also brings performance gain when verifying a new transaction; before, when checking for double spending, for a given inflow coming from a certain outflow, we had to iterate through a large number of transactions in the chain to make sure the outflow had not already been spent. Now, with spent transactions removed, we can tell right away if a new transaction was a double spend.

However, if the list of transactions are included in the computation of the block's hash value, removal of a transaction would obviously alter the hash and therefore introduce inconsistency in the chain. This is where a merkle tree becomes useful. In the Merkle tree, leafs are transactions and every parent node is the hash value of its children. The root, called the "merkle root" or "root hash", is the **only** part of the tree that is included in the *block header* and part of the input to the hash function as we compute the hash value of the block. With this setup, we are free to remove transactions (i.e. the leafs of the tree) as well as interior nodes whose children have been removed *without* altering *the hash value of the block*.

According to multiple sources, Bitcoin Core, the standard open-source blockchain validation software, does not yet implement pruning as described in the White Paper.

https://www.reddit.com/r/Bitcoin/comments/9103q5/blockchain_pruning/
https://bitcoin.stackexchange.com/questions/36100/pruning-the-branches-in-merkle-tree