Blockchain Demo - Mining in Centralized Network using Proof of Work (TO BE COMPLETED IN JAN 2019)

By Ng Yiu Wai, January 2019

Contents

1. Introduction

Bitcoins, the most well-known blockchain network, is described as

> *" A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network.* ***The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power****. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone."*[1]

In this project, I am going to demonstrate below two characteristics of a proof-of-work blockchain network.

i) building a chain of blocks using hash-based proof-of-work
ii) the miner who has highest CPU power could add new blocks to the blockchain.

Bitcoin	This project
◆ Transactions data, which are verified using digital signature, are packed into blocks.	◆ Arbitrary strings data are packed into blocks.
◆ Data are packed into blocks using merkle tree root hash.	◆ Data are packed into blocks using merkle tree root hash.
◆ Each block could be represented by a byte stream under a pre-defined protocol.	◆ Each block could be represented by a byte stream under a pre-defined protocol.
◆ Blocks are chained using hash-based proof-of-work. It is nearly impossible to change data that are already added into the blockchain.	◆ Blocks are chained using hash-based proof-of-work. It is nearly impossible to change data that are already added into the blockchain.
◆ New blocks are broadcast on a best effort basis in a decentralized network. Usually the miner who has highest CPU power could add new blocks to the blockchain.	◆ New blocks are broadcast in a centralized network. The miner who has highest CPU power could add new blocks to the blockchain.

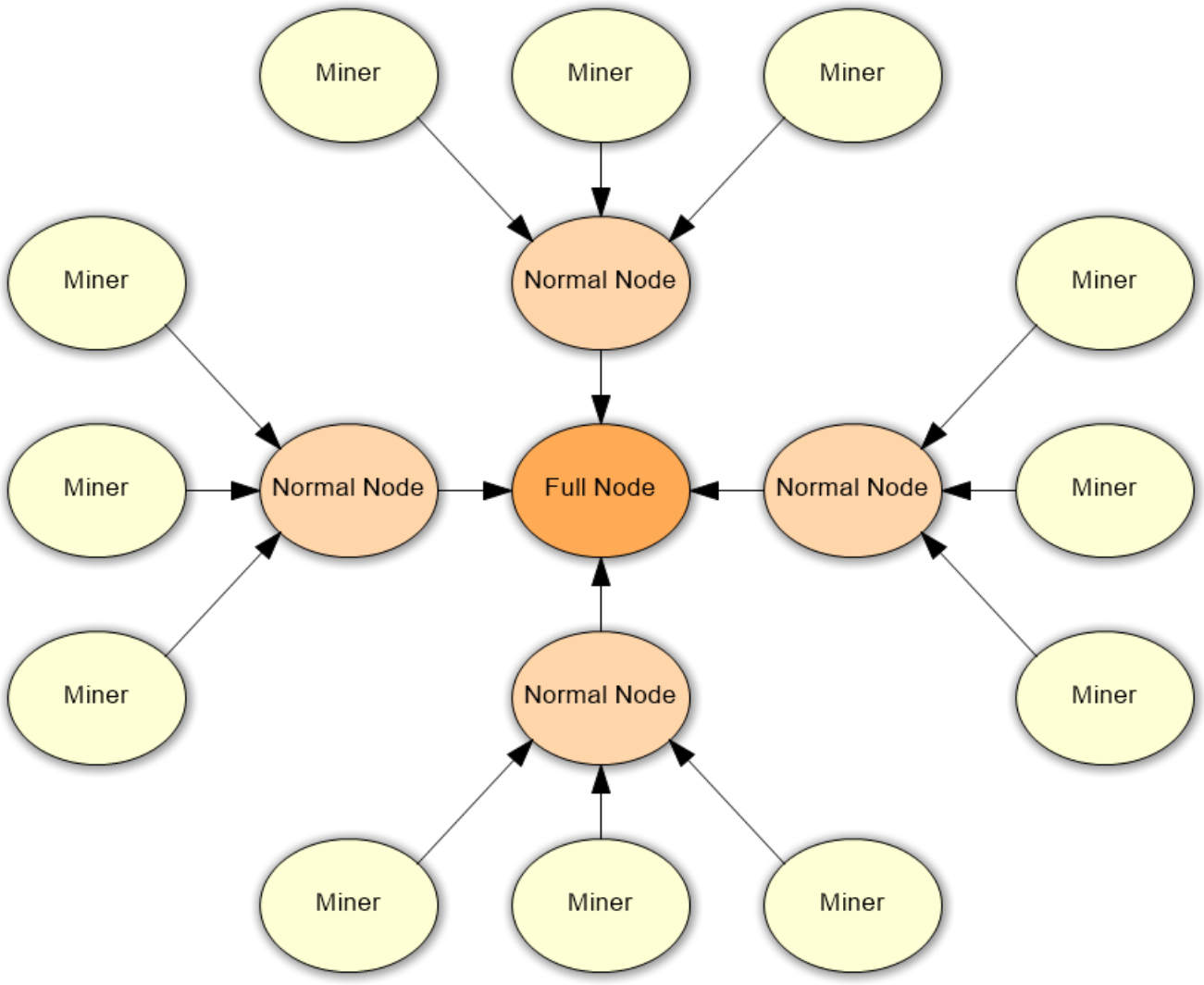
The project is written in GO language. Other characteristics of bitcoins, e.g. digital signatures, peer-to-peer network, broadcasting on a best effort basis, blockchain forks and etc, are ignored in this project.

2. Overview of System

The blockchain platform is a multi-layer server-client network. Three types of application are involved in this centralized network.

- Miner
- Normal Node (as a server)
- Full Node (as a server)

End user could input arbitrary strings in the user interface provided by Miner. The strings data will be saved in the blockchain if the Miner mined successfully.



Miner, who

- receive data from end user
- receive previous block hash from Normal Node
- perform hash-based proof-of-work to pack the data to a new block
- return the new block to Normal Node and try broadcasting

Normal Node, who

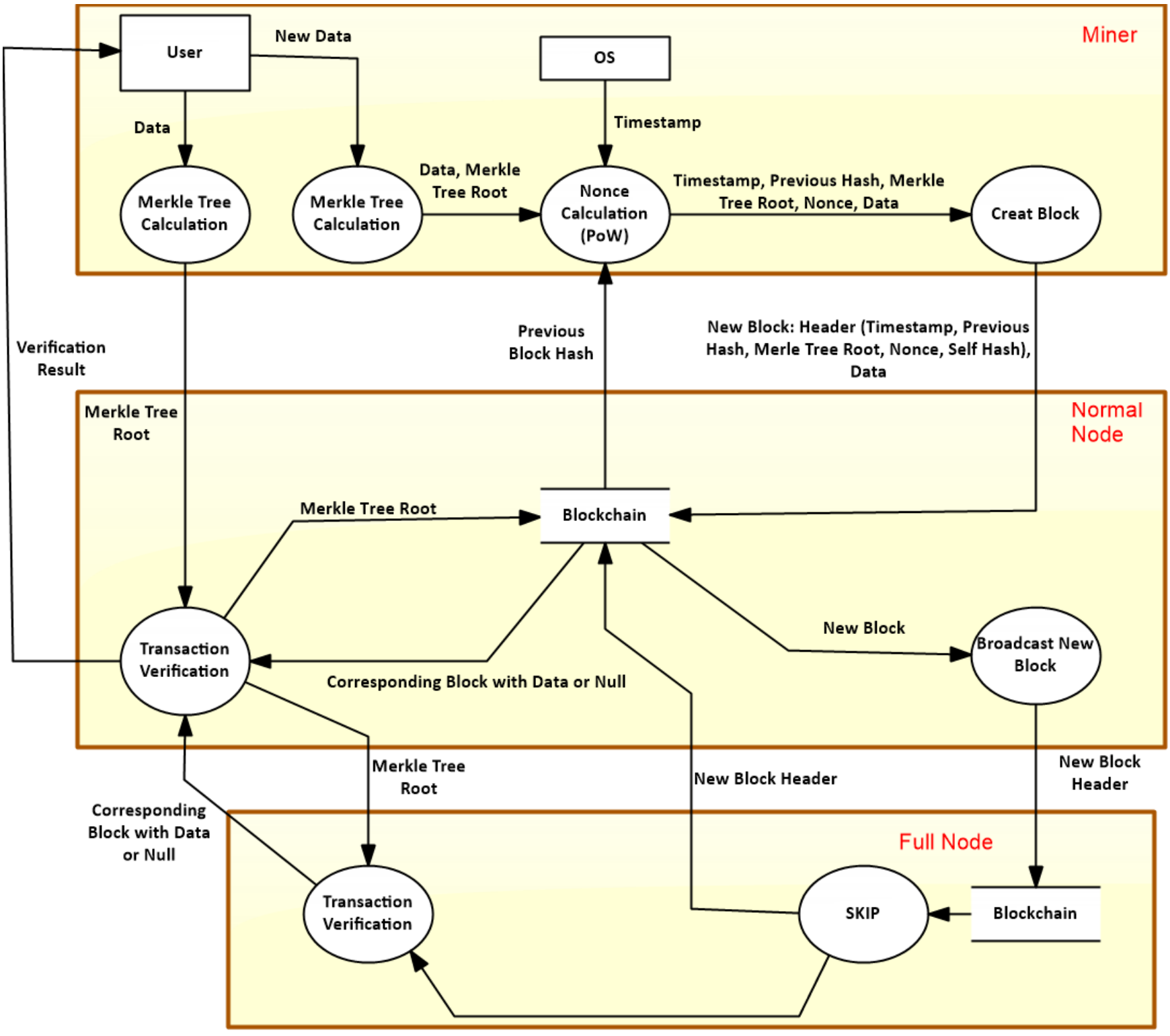
- download blockchain from Full Node.
- receive and verify block from Miner
- try broadcasting new blocks to Full Node
- update locally saved blockchain if Full Node admit the new blocks in blockchain

Full Node, who

- receive and verify blocks from Normal Nodes' broadcasting
- save new blocks to blockchain
- provide the latest blockchain to every Normal Node in the network

Please refer to **Section 7 Demonstration** for examples.

Below is a simplified data flow diagram the shows how the system works.



Miner provides a user interface to end user, and performs proof-of-work. It does not store any data on disk.

Normal Node is responsible for verify blocks from Miner, and performs some searching functions. It stores a partial blockchain on disk. The local blockchains contains (i) all blocks' header and (ii) some block data only.

Full Node is responsible for verify blocks from Normal Node, and performs searching functions if data is not found in Normal Node. It stores the full blockchain on disk. The full blockchains contains (i) all blocks' header and (ii) all block data.

3. System Design – Data & Merkle Tree Root

How Merkle Tree Root is used in Blockchain?

In Bitcoins, a block is composed of a 104-bytes header and variable-size data.[2]

Message Header	start string (magic number)	*4 bytes*	*char[4]*
	command name	*12 bytes*	*char[12]*
	payload size	*4 bytes*	*uint32*
	checksum	*4 bytes*	*char[4]*
Block Header	version	*4 bytes*	*uint32*
	previous block header hash	*32 bytes*	*char[32]*
	merkle root hash	*32 bytes*	*char[32]*
	time	*32 bytes*	*uint32*
	nBits	*4 bytes*	*uint32*
	nonce	*4 bytes*	*uint32*
Data	Transaction 1	*variable*	
	Transaction 2	*variable*	
	…		

Hash is generated from Transactions by using SHA256 and merkle tree. This hash, as known as **merkle tree root**, is a component of block header. Then the block header is chained into blockchain by another hash function. Hence, it is nearly impossible to change transactions that are already added into the blockchain. Further details of merkle tree root is discussed in next page.

In this project, a block is composed of an 80-bytes header and variable-size data.

Message Header	MagicNumber	*8 bytes*	*16-digit hex integer*
Block Header	Timestamp	*4 bytes*	*uint32*
	PrevBlockHash	*32 bytes*	*64-digit hex integer*
	MerkleTreeRoot	*32 bytes*	*64-digit hex integer*
	Nonce	*4 bytes*	*uint32*
Data	String 1	*variable*	
	String 2	*variable*	
	…		

which could be refer to source code *calBlock.go* line 11 to line 32.

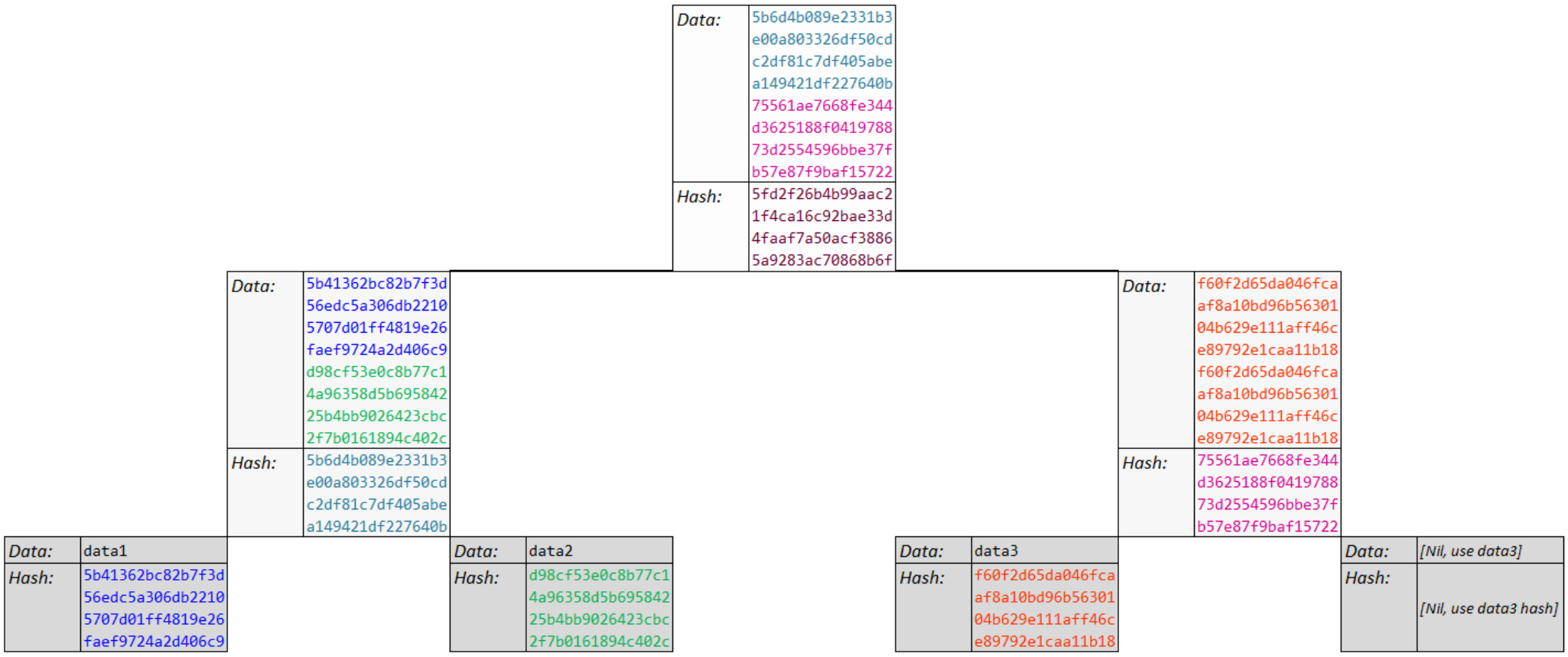
```
// Block : Define object Block
type Block struct {
    // Block Header
    Timestamp     uint32
    PrevBlockHash []byte
    Root          []byte
    Nonce         uint32
    // Block Data
    Data [][]byte
    // Block hash,  can be computed using header
    CurrBlockHash []byte
    // Byte Stream : Serialized Block Header
    //  Block is defines as
    //  8   bytes:  MagicNumber     (16-digit hexadecimal integer, 00004B61726C4E67)
    //  4   bytes:  Timestamp       (10-digit decimal positive integer)
    //  32  bytes:  PrevBlockHash   (64-digit hexadecimal integer)
    //  32  bytes:  MerkleTreeRoot  (64-digit hexadecimal integer)
    //  4   bytes:  Nonce           (10-digit decimal positive integer)
    //  Variable :  Data            (UTF-8)
    // Length of Header = 78 bytes
    ByteStream []byte
}
```

The two additional attributes, *CurrBlockHash* and *ByteStream,* could be calculated using block header. They are included in blocks for the purpose of easier computation only, which is not necessary attributes for blocks.

How to transform Data to Merkle Tree Root?

A hash function is any function that can be used to map data of arbitrary size to data of a fixed size. SHA256 is used in Bitcoins and this project.

Merkle tree is a method to generate a hash for an array of arbitrary size to data. Then the resulting hash, as know as merkle tree root, is packed into blockchain. Below is an illustrative example of merkle tree root generation.



Step 1: Prepare leaf nodes by calculating the SHA256 hash of each data.
Step 2: Use the combination of two leaf nodes hash as upper level node data. Calculate the hash of upper level node data.
Step 3: Repeat Step 2 until there is only one upper level node. It is the root node.

In this example, the merkle tree root is `5fd2f26b4b99aac21f4ca16c92bae33d4faaf7a50acf38865a9283ac70868b6f`.
And below is the execution result.

```
C:\Windows\System32\cmd.exe
Self Node port 9999 ; Server Node port 9999 ; (Full Node @ Port 9999 as a Server)
Enter 10 to become a Node
Enter 20 to become a Miner
Enter 30 to calculate a Merkle Tree Root
30
Tree:   Enter data to be used for Merkle Tree Calculation (seperated by ',')
data1,data2,data3

Leaf Nodes:
5b41362bc82b7f3d56edc5a306db22105707d01ff4819e26faef9724a2d406c9, [data1]
d98cf53e0c8b77c14a96358d5b69584225b4bb9026423cbc2f7b0161894c402c, [data2]
f60f2d65da046fcaaf8a10bd96b5630104b629e111aff46ce89792e1caa11b18, [data3]

Upper Nodes:
 Hash[0]        5b6d4b089e2331b3e00a803326df50cdc2df81c7df405abea149421df227640b
 Data[0]        5b41362bc82b7f3d56edc5a306db22105707d01ff4819e26faef9724a2d406c9d98cf53e0c8b77c14a96358d5b69584225b4bb9026423cbc2f7b0161894c402c
 Hash[1]        75561ae7668fe344d3625188f041978873d2554596bbe37fb57e87f9baf15722
 Data[1]        f60f2d65da046fcaaf8a10bd96b5630104b629e111aff46ce89792e1caa11b18f60f2d65da046fcaaf8a10bd96b5630104b629e111aff46ce89792e1caa11b18

Upper Nodes:
 Hash[0]        5fd2f26b4b99aac21f4ca16c92bae33d4faaf7a50acf38865a9283ac70868b6f
 Data[0]        5b6d4b089e2331b3e00a803326df50cdc2df81c7df405abea149421df227640b75561ae7668fe344d3625188f041978873d2554596bbe37fb57e87f9baf15722

Tree:   The Merkle Tree Root is 5fd2f26b4b99aac21f4ca16c92bae33d4faaf7a50acf38865a9283ac70868b6f

C:\Development - Go\Blockchain - Mining in Centralized Network using PoW\code>
```

Please refer to source code *calMerkleTree.go* for details. You may also refer to https://bitcoin.org/en/developer-reference#merkle-trees for more information about algorithm.

4. System Design – Block & Proof-of-Work

To update

5. System Design – Blockchain in Network

To update

6. System Design – Centralized Network & Data Storage

To update

7. Demonstration

To update

8. Extra Information – Blockchain should be in a decentralized network

To update

9. Reference

1. "Bitcoin: A Peer-to-Peer Electronic Cash System", Satoshi Nakamoto, https://bitcoin.org/bitcoin.pdf
2. “Bitcoin Developer Reference”, Bitcoin.org, https://bitcoin.org/en/developer-reference

The source codes are written using Visual Studio Code. https://code.visualstudio.com/
The diagrams are drawn using Software Ideas Modeler. https://www.softwareideas.net/

This project is a rebuild of student project I worked on for Hong Kong Polytechnic University COMP5311 (Semester 1 Year 2018-2019).
Special thanks to Dr Bin Xiao and Mr Haotian Wu, who taught me the fundamentals of Bitcoins and blockchain.