Experiment No.4
To implement the concept of block and blockchain using javascript
Date of Performance: 24/08/2023
Date of Submission: 24/08/2023

AIM: To implement the concept of block and blockchain using javascript

Objective: To develop a program, demonstrating the concept of block and blockchain

Theory:

Blocks are data structures within the blockchain database, where transaction data in a cryptocurrency blockchain are permanently recorded. A block records some or all of the most recent transactions not yet validated by the network. Once the data are validated, the block is closed. Then, a new block is created for new transactions to be entered into and validated.

A block is thus a permanent store of records that, once written, cannot be altered or removed.

A block stores information. There are many pieces of information included within a block, but it doesn't occupy a large amount of storage space. Blocks generally include these elements, but it might vary between different types:

- Magic number: A number containing specific values that identify that block as part of a particular cryptocurrency's network.
- Blocksize: Sets the size limit on the block so that only a specific amount of information can be written in it.
- Block header: Contains information about the block.
- Transaction counter: A number that represents how many transactions are stored in the block.
- Transactions: A list of all of the transactions within a block.

The transaction element is the largest because it contains the most information. It is followed in storage size by the block header, which includes these sub-elements:

- Version: The cryptocurrency version being used.
- Previous block hash: Contains a hash (encrypted number) of the previous block's header.
- Hash Merkle root: Hash of transactions in the Merkle tree of the current block.
- Time: A timestamp to place the block in the blockchain.
- Bits: The difficulty rating of the target hash, signifying the difficulty in solving the nonce.
- Nonce: The encrypted number that a miner must solve to verify the block and close it.
-

Genesis Block

The genesis block is the first block of the blockchain. The genesis block is generally hardcoded in the applications that utilize its blockchain. The Genesis Block is also known as Block Zero or Block 0. It is an ancestor that every Blockchain network's block that can be traced to its origin back.

Blockchain

A blockchain in simple word is a database that stores and encrypts information in a linked fashion, so that previous information cannot be altered, and a group verifies any entries before they are finalized through a consensus—an agreement that the data is correct.

Blockchains are used in cryptocurrency, decentralized finance applications, non-fungible tokens, with more uses constantly under development.

Process:

Step 1. Open the NetBeans IDE

Step 2. Create new project of categories HTML/javascript and select Node.js application in the projects tab and click next

Step 3. Give a suitable project name in the name and location tab and click next

Step 4. Tick the Create Package.json in the Tools tab and click Finish

Step 5. In the project directory under the source directory of the project create the required .js file

[block.js, blockchain.js, crypto-hash.js, genesis.js, server.js]

Step 6. Run the server.js file, if no error then the resulting blockchain is created

Code:

```
// block.js
const { GENESIS_DATA } = require('./genesis.js');
const cryptoHash = require('./crypto-hash');
class Block {
  constructor({timestamp, lastHash, hash, data}) {
    this.timestamp = timestamp;
    this.lastHash = lastHash;
    this.hash = hash;
    this.data = data;
  }
  static genesis() {
    return new this(GENESIS_DATA);
  }
```

```
    static mineBlock({lastBlock, data}) {
        const timestamp = Date.now();
        const lastHash = lastBlock.hash;
        return new this({
            timestamp,
lastHash,
            data,
            hash: cryptoHash(timestamp, lastHash, data)
        });
    }
}
module.exports = Block;
// blockchain.js
const Block = require('./block');
class Blockchain {
    constructor() {
        this.chain = [Block.genesis()];
    }
    addBlock({ data }) {
        const newBlock = Block.mineBlock({
            lastBlock: this.chain[this.chain.length-1],
            data
        });
        this.chain.push(newBlock);
    }
}
module.exports = Blockchain;/*
  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
  * Click nbfs://nbhost/SystemFileSystem/Templates/ClientSide/javascript.js to edit this template
 */
// crypto-hash.js
const crypto = require('crypto');
const cryptoHash =(...inputs) => {
    const hash = crypto.createHash('sha256');
    hash.update(inputs.sort().join(' '));
    return hash.digest('hex');
}
module.exports = cryptoHash;
```

```
// genesis.js
const GENESIS_DATA = {
   timestamp: Date.now(),
   lastHash: '64b7edc786326651e031a4d12d9838d279571946d8c9a5d448c70db94b0e143f',
   hash: 'c671c84681b9d682b9fd43b2a2ef01a343eab7cfa410df9835f8165007d38467',
   data: 'Dinesh'
};
module.exports = { GENESIS_DATA };
// server.js
const Blockchain = require('./blockchain');
const Block = require('./block');
const blockchain = new Blockchain();
for(let i=0; i<5; i++) {
   const newData = 'Dinesh'+i;
   blockchain.addBlock({data: newData});
}
console.log(blockchain);
```

Output:

```
Blockchain {
  chain: [
    Block {
      timestamp: 1692889990416,
      lastHash: '64b7edc786326651e031a4d12d9838d279571946d8c9a5d448c70db94b0e143f',
      hash: 'c671c84681b9d682b9fd43b2a2ef01a343eab7cfa410df9835f8165007d38467',
      data: 'Genesis Block'
    },
    Block {
      timestamp: 1692889990418,
      lastHash: 'c671c84681b9d682b9fd43b2a2ef01a343eab7cfa410df9835f8165007d38467',
      hash: 'cfa4f412aa535971e593aa3e7765bbb8c324ddd3ea865db6f0b5b6f205667fb3',
      data: 'Block 0'
    },
    Block {
      timestamp: 1692889990418,
      lastHash: 'cfa4f412aa535971e593aa3e7765bbb8c324ddd3ea865db6f0b5b6f205667fb3',
      hash: 'e42b4c19711eb28dcf0b3aa6f4e35bdaa7a5b03cb1c22ad2bd12536ea77dacaa',
      data: 'Block 1'
    },
```

```
  Block {
    timestamp: 1692889990418,
    lastHash: 'e42b4c19711eb28dcf0b3aa6f4e35bdaa7a5b03cb1c22ad2bd12536ea77dacaa',
    hash: '6708e2bc48c12b77d7f4eee8a8159eba0c9d5b4889b9ec6503480d39ecb89073',
    data: 'Block 2'
  },
  Block {
    timestamp: 1692889990418,
    lastHash: '6708e2bc48c12b77d7f4eee8a8159eba0c9d5b4889b9ec6503480d39ecb89073',
    hash: 'e8795a4575b4f46c41178dc078cda69d729ff6950c60c7d24a7acd2130dac4d3',
    data: 'Block 3'
  },
  Block {
    timestamp: 1692889990418,
    lastHash: 'e8795a4575b4f46c41178dc078cda69d729ff6950c60c7d24a7acd2130dac4d3',
    hash: 'c24bdbb01a8e8cee10aeb927487e2b01dc29d5a9826cbef877b0f6ab7d318add',
    data: 'Block 4'
  }
 ]
}
```

Conclusion: JavaScript's widespread familiarity and web integration make it an excellent choice for creating blocks and forming a blockchain. Its simplicity expedites development and supports iterative prototyping, especially when integrated with Node.js for server-side logic. JSON compatibility aligns well with blockchain data structures, while libraries like `web3.js` simplify interaction with networks. Moreover, JavaScript's role in writing smart contracts (e.g., Solidity) enables automated processes. Its educational value aids beginners, and its robust community offers abundant resources. However, for performance-intensive tasks, languages like C++ might be preferable. JavaScript's accessibility, versatility, and community support contribute to efficient blockchain development, especially for web-driven applications.