PROJECT REPORT

BITCOIN

SUBMITTED BY-

SHREYA SINGH (7915-4462)

AMRUTA BASRUR (4463-4819)

1) Implementation steps and functional description

Features implemented are –

1) **Transactions** –
   A transaction is a part of the transaction chain which is of the specific format. A transaction consists of the following input and output attributes-
   Format-
   Input transaction ID, outpoints
   Signature script which is signed with the private key of the node
   Output public keys and amounts

   The above attributes have been hashed and signed and stored as the signature for the next Transaction. Transactions with multiple inputs and outputs can also be created. The block validates the ownership of the transaction using the public key and private key pair. Key pairs have been generated using Elliptic curve cryptography.
   - A transaction can take multiple inputs and outputs
   - A coin base transaction has been implemented for transferring the mining rewards to the miners.

2) **BlockChain-**
   A block chain is a chain of blocks which consists of below details:
   After a miner mines a block, the block first mined is validated by all miners and after validation it is added to the block chain.
   Along with this no of transactions within the block and the entire transaction list is stored inside.
   There are several validations performed on each element within the block before it is selected.

   Header
   - Hash of block header
   - Hash of previous block
   - Merkle root - double hash of all transactions within block to form a merkle tree)
   - Block number (32-byte SHA256(SHA256())
   - Nonce - a number incremented from 1 until a hash is obtained which is lesser than difficulty target (32-byte)

- Version - 32-byte integer
- Time – time when miner started mining current block (32-byte)

It also includes the below transaction details:

- No of transactions within the block
- Entire transaction list

3) **Wallets** –

Each node has a wallet of its own which stores the amount of transactions which have been signed by it's public key. Each time a transaction is confirmed the wallet gets updated by deducting the amount from the sender's wallet to adding the amount in the receiver's wallet.

The wallets are also used to check the combination of transactions to be sent to the output and to identify if a transaction requires another output with the self-address.

Transaction fee-

A transaction fee of 0.1% of the output amount has been added to the transaction and is present only if the sender has enough balance in the wallet.

4) **Mining** –

Miner nodes have been created which parallelly mine the pending transactions and the miner who mines first has his block added to the block chain after block reaches consensus by other miners.

- Miners pick up the transactions with maximum reward first.
- Also, the wallet gets updated on receiving the mining reward
- The miners are also responsible for creating a Coinbase transaction addressed to themselves to get the mining reward.

Difficulty target –

A difficulty target with 2 leading zeros was chosen for the block to get confirmed. The following difficulty target has been used in the program.

@difficulty_target
"00805F511E5157EB90B7754ACC85055B19EA74B43BD0F1D7A066946F973F87E8"

5) Peer to Peer network

The nodes in the network are connected in a peer-to-peer network, such that each node has the public key of the other one while retains private key with itself. All the

transactions which are made in a network are visible to everybody and can be seen by all the nodes in the network. But the node can only own a transaction for which it has the private key.
Since block chain is a distributed ledger the right to view a transaction reserves with everybody. Whenever a block is mined and added to a block chain, it is accessible to all. Even the list of pending transactions is visible to all the nodes in the network.
The block which has been added is circulated to the entire network.

6) **Block validation**
The block before getting added is validated by two of its peer nodes. This is the part of promises in block chain where the validating nodes are chosen at random and verifies the miner's outputs before a block gets added to the block chain.

Basic functioning

- The test cases provide input arguments to the program and validate the results
- The Coinbase transaction starts the network and initially each node is given a fixed amount of 25
- The genesis block is created first and the inputs have been fixed for the block.
- As soon as a node gets a request to transfer the amount to the other node, the node uses the receiver's Public key and the transaction IDs of the transactions which are available with it. It also checks the amount available in the wallet. And then the transaction is created by hashing and signing using public keys. ECC has been used to generate the key-pair.
- The transaction is then moved to the list of pending transactions and the miners begin working on it to mine the transaction.
- After a block is mined, the first block mined or the block with maximum transactions is selected and sent to all other nodes for validation
- After reaching consensus, mined block is added to the block chain and sent to all nodes via P2P network
- The largest number of nodes being tested for block chain is 100 nodes and maximum number of requests tested is 100

2) Procedure for Running the code

1) Unzip the file Amruta_Shreya.zip
2) Navigate to the folder directory.
3) Run the command –
   mix test
4) The test cases would execute, and the outputs will be displayed.

Unit Test Cases (10 cases)

Test Case Number	Test Case Name
Test Case 11	Only 1st Transaction is coin base per block
Test case 9	Validate null transactions
Test case 8	Check for spending > money in wallet
Test case 10	Validate invalid money range transactions (amount less than 0)
Test case 4	Only first Block is Genesis Block
Test case 6	Block Hash less than difficulty target
Test case 10	Validate hash > difficulty target
Test case 5	Proof of work
Test case 8	Validate Merkle Root
Test case 7	All mined transactions had inputs from unspent transactions

Functional Test Cases (5 cases)

Test Case Number	Test Case Name
Test Case 1	Create genesis block with initial amount
Test case 9	Validate all mined transactions to be in proper bitcoin format
Test case 2	Mine next Block and send it to all peers after validation
Test case 7	Wallet successfully updated after a transaction
Test case 3	Validate entire Block Chain

3) Outputs

Blocks:


```
{"Blocks", 20,
[
  "005911B9BFA6899764571C302C9FA6F2A21FD5F61CC7F9E4FFD1BFADAB411966",
  %{
    merkleRoot: "DF0CBAF241C50138C356346DB460A7DE03C018E3D7E696D055351FA2C1B32EBD",
    nBits: "1F805F51",
    nonce: 92,
    previousBlockHash: "003C981FAC205DDD966F3EA17BC601062E364E3E13DD043B5E8D10C9191CA036",
    time: <<91, 252, 208, 103>>,
    version: <<0, 0, 0, 1>>
  },
  3,
  [
    {"3AA6E14EC5D0B28369278179F890559C17DD7DD9C0F158E29DA665BCFB3722B2", 0,
     %{
       inputTxIds: [],
       inputs: {"0", 0},
       outputs: [["BE8746607C9F2FDBCEC765B74F95C0D45ED8B603", 28.0]],
       sig: ""
     }},
    {"4BDF33372C427D13490CD828655BD16188B409A20848F6C7D433A89F31ACE7B1", 1.5,
     %{
       inputTxIds: ["3909E08F5AB635614F8E0584EBD3220BA5813BC28B5847224456548BC25607EAE"],
       inputs: {"019C96FBBD508433EF6A1FB39E8A1A3F109C5926", 25.0},
       outputs: [
         ["E76039B0D9B395D712983BAC7FA10AC6989E71D7", 15],
         ["019C96FBBD508433EF6A1FB39E8A1A3F109C5926", 8.5]
       ],
       sig: "65249AEAECFECB4C9E70F0D165D9072E6E7A2DD9E65DA3EFCDE6E7D4CB9CA723180104DB899A5F9B07C1F2DFACE5BEBE59746608CE803A7A35F738E2D66
```


Transactions


```
{"unspentTxns",
 "D70C82FE55D8404D50D26FCF5AE890940F8E26947292AF2B52271699C4801D93", 0,
 %{
   inputTxIds: [],
   inputs: {"0", 0},
   outputs: [["140C3EBE1A5C0002059C2C54699F25B17B0AA029", 26.2]],
   sig: ""
 }},
{"unspentTxns",
 "3A9421F35D300A968E461F1BD48F598372B96160FF5C0F539C18F09F532CC800", 0.6,
 %{
   inputTxIds: ["B66364AE180A1888F3D2BD57788C588081CB1B08ACC6B91CF4939365DB894680"],
   inputs: {"F2A4B3BE1AE1D583ACF487699C718CA3E769BF1F", 25.0},
   outputs: [
     ["0F75DBFD652F1FC6B294A2EF43C5359B127148BF", 6],
     ["F2A4B3BE1AE1D583ACF487699C718CA3E769BF1F", 18.4]
   ],
   sig: "65025EA8B9B4752CEF539554397C7C8E303A3EE546A4854D952FF40881923F6AEB0104C9411965E43B8E330174FA41A24ACF307E3A0ED1E3604553970349832E210F7915A49997E3947C58025ED
7ACF61AEC1C932BC3F56C27322C7853002EE"
 }},
{"unspentTxns",
 "0E29419BA4738797806652A6E58617B3D6CD9E70BC2634DD1B2B9ABE2E03660A", 0.6,
 %{
   inputTxIds: ["68FBCAA9B0E93110EADCF9BEA0CD066800E5DCAE2AA82E8422BBA7290DB65AE9"],
   inputs: {"38B5C4453969F5FD11FB2BE24110215ECE6CA9EB", 25.0},
   outputs: [
     ["F2A4B3BE1AE1D583ACF487699C718CA3E769BF1F", 6],
     ["38B5C4453969F5FD11FB2BE24110215ECE6CA9EB", 18.4]
   ],
   sig: "6534876A28EE74A7A0295F999E631DC302663525D4D38256730213CD188EF242260104A0F9120B6DA58A026FE3DCB52CF7D8084F5810FEE4B5C0E2F7965ACD67E7992363B12DCC0E6D5E2319CC6
7E4610719C2D9879A5F2A51FA6514FF6753B"
```

Test cases output:

```
Starting test cases
Test case 7
.Test case 3
.Test case 8
.Test case 10
.Test case 9
.Test case 5
.Test case 11
.Test case 4
.Test case 2
.Test case 6
.Test case 7
.Starting test cases
Test case 9
.Test case 8
.Test case 10
.Starting test cases
Test case 1
.

Finished in 5.4 seconds
15 tests, 0 failures

Randomized with seed 135000
```

Test case outputs with test names:

```
E:\DOSProject1\bitcoin>mix test
Starting Blockchain Test Cases
Mine next Block
.Proof of work
.Validate all mined transactions to be in proper bitcoin format
.All mined transactions had inputs from unspent transactions
.Validate Block Chain
.Only 1st Transaction is coinbase per block
.Block Hash less than difficulty target
.Validate hash > difficulty target
.Validate Merkle Root
.Wallet succesfully updated after a transaction
.Only first Block is Genesis Block
.
Check for spending > money in wallet
.Validate invalid money range transactions
.Validate null transactions
.Starting Genesis block test cases
Create genesis block with initial amount
.

Finished in 5.3 seconds
15 tests, 0 failures

Randomized with seed 674000

E:\DOSProject1\bitcoin>
```

4) Test cases executed

The following is the detailed structure of the test cases which have been executed –

1) Create Genesis Block with initial amount-

The genesis Block is the first block of the blockchain. It is a special block without any input transaction reference IDs and hence "0…... 32 bytes" have been used as an initial input.

Functionalities tested –

- The test has been created with respect to the updating wallet amounts for coin base transactions and genesis block.
- Coinbase transactions is created with 0 inputs and the desired value is reflected within the wallet of each node.
- The program terminates after the execution of one block and no further transactions between nodes are initiated.

Passing Criteria –

- Test cases is marked as pass when the wallets of all the nodes have been updated with the initial amount and the genesis block is created with the corresponding transactions.

2) Mine next Block-

The test case checks whether the miners which run parallelly pick up next block and do not update the same block over again. Only the miner who mines the block first is able to add it into the block and take the mining rewards.

Functionalities tested –

- Miner who mines first gets the reward
- Miner who mines first has his block added to the blockchain

Passing Criteria –

- Updating of miner's wallet with the mining rewards.

3) Validate Block Chain-

This test validates the hashes of the previous block and if it is valid for all the blocks. The block id is generated by hashing the previous block.

- This test case checks whether the hash of previous block produces the hash of the current block.

- This also ensures no two blocks are added concurrently to the chain, in case each miner mines the two blocks simultaneously.

4) Only first block is Genesis block-

The first block has no input Ids so the input needs to be provided to the first block. This test case validates the hash of the first block

- The hash of the input of the first block i.e. 0's is taken and validation is performed so that no other block in the chain produces the same hash value.
- Also, the transactions in the block are the coin base transactions in our code

5) Proof of Work-

This test case validates the below mentioned conditions.

- Validation towards the entire blockchain – This includes the validation for the previous and the current block and whether the block has been generated using the transaction reference of the previous block. The hash of the previous block is calculated.
- Validate Merkle Root – The validation of the merkle root is done. This ensures that the program calculates the merkle hash in the specific calculation format by ensuring that only two hashes are taken at a time and then the subsequent two hashes are taken next.
- Validate All Hashes – This function computes hashes for all the blocks by increasing nonce by one and then computing the hash values of the remaining bits.

6) Block hash less than difficulty target-

A difficulty target of a number with 2 leading zeros has been chosen for the program. The test case validates whether the difficulty target has been achieved by the block before getting added to the blockchain.

- The difficulty target pre set for the program > the transaction reference of the block.
- A block which does not meet the difficulty target level is rejected and the miner mines the nodes again to achieve the target level
- The test case passes if all the blocks have achieved the difficulty target.

7) Only first transaction is coin base per block

Every mined block has an initial coin base transaction and all the other transactions are fetched from 'Pending transactions'.

This test case performs the following validations;

- First transaction of every block is a coin base transaction with input as 0 and output lesser than miner fee + transaction fee.
- All the other transactions have a non-zero input and a non-zero output with a valid transaction fee.

8) Wallet updated after every transaction successfully

We have included the code for updating of wallet after every block is mined as well as when the user wants to call wallet to see current wallet amount.

Here after all transactions are mined into the block, we are verifying whether the amount in wallet of every node matches with the amount in unspent transactions list.

9) All mined transactions present in unspent transactions

Here we validate whether every block consists of input transactions present in unspent transaction global list.

Since after mining a block, input transactions are removed from unspent transaction list and current transaction included in the block is removed from pending list. We will have a valid state in both the lists.

10) Validate merkle root

Here we take every transaction from a block and calculate its merkle root. It is then matched with the merkle root present in the block.

11) All mined transactions in bitcoin format

Here we check the following validations

- Input is non-zero for all transactions apart from coin base
- Output is non-zero for all transactions
- Input <= outputs + transaction Fee
- No transaction input with wrong public key
- Inputs and outputs exist in legal money range
- Transaction list is non- zero

12) Hash greater than difficulty target-

This function validates that the hash of all the blocks in the block chain is greater than the difficulty target. The difficulty target with leading 2 zeros has been set

- The test case checks for all the transaction reference of the blocks and checks if all of them are greater than the difficulty target.

13) Money in invalid range transactions-

This test case checks if the money is in invalid range. It takes in the input amount to be transferred to the receiver and validates that it should not be zero.

14) Null transaction validation –

This test case validates that a transaction cannot be null. A null transaction is the one in which outputs are zero or the amount transferred is zero.

- The test case rejects the null value transactions in the amount or in the receiver node.
- It passes with the correct values in the sender's amount, receiver's amount and the correct public key of the receiver.

15) Check for spending greater than the money in wallet-

This test case checks for over spending and fails when the amount to be transferred is greater than that present in the wallet.

- This test case fails if the amount to be transferred is greater than that to be sent.
- This prevents over spending by the sender node.