COP5615
Distributed Operating Systems
Project #4.1

Kaustubh Bapat (3385-9975)
Indrajit Shanbhag (6569-8772)

Testing the Code:

The code requires two dependencies – Poison and Crypto to run, hence the dependencies must be installed prior to testing. Run

deps.get

to install the dependencies.
To test the code run

mix test

This runs all the unit tests in a randomized manner and gives success or failure message.

Implementation and tests:

The application is composed of various modules (classes) to implement the functionality of bitcoin. These modules are as follows:

- Block
  - This class defines the various methods that are relevant for a single block, for example creating a block, checking if the hash for a block is valid, etc.
- Blockchain
  - This class defines methods for blockchain like creating blockchain, validating if blocks are in order(as per hashes), if blocks are valid, etc.
- FullNode
  - This is the main module of the application. It is responsible for maintaining and extending a block chain, keeping track of connected wallets and confirming pending transactions.
  - When it extends the blockchain it sends broadcast to all the connected wallets with the newly formed blockchain
  - This class also creates the network of peers, can mine genesis block, add peers, notify peers, confirm transactions etc.
- Miner
  - This module can validate new blocks for the blockchain and can mine bitcoins.
- TransactionUnit
  - This class defines the basic structure of a transaction unit and methods associated with it like creating a transaction unit consisting of unique id, amount and a key.

- Transaction
  - This defines methods for creating transactions (consisting of one or more transaction units), validate transactions, etc.
- Utils
  - A utility class
- Wallet
  - This module can create wallets, update its copy of blockchain, validate the amount to be sent (based on bitcoins it owns) and send bitcoins to peers.

For each of these modules unit tests are written and the tests can be found in ***test/*** directory with file name as ***<module>_test.exs*** for example wallet_test.exs

Two types of tests have been written in the code,

1) Tests that simply check if the output for a called function is correct or not using ***assert*** statements of elixir.
2) Tests that wait for a reply from another entity and then asserts if the received message is as expected or not. This is done by using ***assert_receive*** statements in elixir.

Following tests cases are implemented for each modules:

Block_test

- Creates blocks and verifies parent of different blocks
- Tests if blocks are valid
- Tests if blocks are equal

Blockchain_test

- Creates a block chain
- Tests whether blocks are added to the chain properly and if the resultant block chain is valid
- Adds random block to the chain and checks whether chain is invalid (as expected)
- Retrieves unspent transactions from the block chain and if the amounts in transaction units add up to correct value (essentially what will be owned by a user in wallet).

FullNode_test

- Before mining a new block verifies that the transaction does not contain a transaction input that has been spent already
- Starting a node with empty block chain: Validates that when started with an empty blockchain, it can mine the first block of the system (genesis block)
- Started a node with existing blockchain: Makes sure that when started with valid block chain the node verifies that the blockchain is valid and notifies connected wallets about new block chain
- Verifies if transactions between different wallets is valid
  - Wallets can send half bitcoins

- Wallets cannot send negative amounts
- Overspending is not allowed
- Sending a valid amount updates the current balance of both wallets.

Mine-test

- test if miner can mine a new block containing a valid transaction
- difficulty has been set to 1, which means the block will be valid if the hash contains only one 0 at the beginning
- Starting from an empty blockchain, miner creates the genesis block
  Given an array of transactions, verifies if miner can mine a new block

Transaction_test

- Create a transaction and verify its inputs and outputs
- Verifies validity of the transaction

TransactionUnit_test

- Creates a transaction unit and checks by querying if the ID, amount and key are valid

Wallet_test

- Creates multiple wallets with unique public-private keys
- Check whether balance reported by the wallet matches with transactions in blockchain
- Check if amount being sent to another wallet / public key is valid
  - Checks for negative transfers
  - Checks for overdraft
  - If amount is valid, checks if the returned array of transactions are suitable as “inputs”(in bitcoin jargon) for the transaction
- checks if generated “outputs”(in bitcoin jargon) is correct for a transaction