DMBLOCK Assignment 1 Documentation

Zaitsev Artem,
AIS ID: 121320,

Phase 1:

In phase 1 our task was to implement transaction validation and transaction handler which must validate as much transaction from the given list as it can.

My experience:
Transaction validation was ok,
Following things was taken into consideration:

- Valid signature on each Transaction's input.
- UTXO is in UTXO Pool
- No double spending in Transaction (For example 2 inputs with same hash and output index)
- Output values are non-negative
- Sum of output values =< sum of inputs

As I was coding in Python and not in Java, I couldn’t use RSA.jar, that is why for key-pair generation, signing and verification Pure Python RSA implementation was installed and imported.

To signing transaction additional method was added to Transaction class:

```
def sign_tx(self, privkey: rsa.PrivateKey, input: int):
    signature = rsa.sign(self.get_data_to_sign(input), privkey, 'SHA-256')
    self.add_signature(signature, input)
    self.finalize()
```

This method was inspired by the same method in Main.java added to phase 1 github

To verify signature in tx validator I used following code:

```
        rsa.verify(tx.get_data_to_sign(tx.inputs.index(input)), input.signature,
output.address )
```

It takes data which should have been signed, signature and public key of the wallet which has sent this transaction.

To check if UTXO is in UTXO pool I take inputs data and create object of UTXO class and check if get_tx_output return output, if it is not than there is no such UTXO in pool and transaction should not be validated.

UTXO check and signature verification is done to each input of transaction, in same cycle I sum up all values of UTXO's to check if output values < input values later.

Also there is set of UTXO's in which I add all existing UTXO's used in TX, if length of this set is lower than length of actual TX inputs it means that some of that have been used more than once and that means DoubleSpending.

Finally I iterate through all outputs to check that all values are non-negative and subtract those values from the sum of input values.
If this sum is lower than 0 this means that input values < output values -> return False / raise exception.

Handler

Handler is implemented as while cycle which will stop only if after 2 iterations possible_txs list won't change.
There is additional method handle_tx which is used to check if tx is valid, if so actualize utxo pool and return this tx,

I haven't implemented MaxFeeHandler that is why I haven't done any ordering in handler.

Tests:

As most of tests-cases are self-explanatory, I will only say few words about some of them.

Test 2:
To test valid signature on invalid data I signed tx and then changed one of it's outputs address.

Test 14:
Complex transactions are not very self-explanatory and unfortunately as I am doing it at last moment, it is pretty late to ask what was the meaning, so in my implementation of this test handler gets 10 each is dependant from previous, but handler gets them in reversed order so that 1st tx to check will be last created.

Handler must iterate through all of them while there is no more tx to validate. O(n^2) welcome.

Phase 2:

This phase was successfully failed and not done :()

Phase 3:

This one was interesting at first. Before I had to write tests for it…

The point of Phase 3 was to create a blockchain, In fact we had to implement blockchain initialization and new block addition.

Blockchain initialization

When we create new blockchain we should take genesis block and set it as root node of blockchain.
Also I had to create empty transaction pool and genesis block utxo_pool from txs in genesis_block (if any + coinbase_tx)
Max_height_block on initialization = genesis_block
And genesis_block is added to blocks dictionary

Block_add

Adding new block we had to check following things:

- If the block is young enough to be added to blockchain
- If all txs in block are valid
- If block's parent block exists.

If those conditions are True than we add block to blockchain and actualize following things:

- Delete txs from transaction pool (if they were there at all)
- Block's utxo pool should consist of previous block utxo pool + new UTXOs + coinbase TX utxo

For simplicity of implementation I had two useful things as blockchain attributes:

Max_height_block stores highest block in blockchain.

Blocks is a dictionary in which block_hash - BlockNode key-value pairs are stored.

Those attributes also were actualized when the new block was created, if it was necessary.
It is redundant to store too old blocks in dictionary that is why if new block updated max_height_block value then I iterate through dictionary blocks to find all blocks which must be deleted from it and delete it.

Coinbase TX

As creation of new blocks creates coinbase transaction we should create it in our code too.

```
    @staticmethod
    def create_coinbase(value, address):
        coinbase = Transaction([Transaction.Input(b'',0)],[Transaction.Output(value, address)],None, True)
        coinbase.finalize()
        return coinbase
```

Above you can see code which does it in my implementation. I used staticmethod not to have to create object of Transaction class on each block creation, I think it is valid to say that here is an example of Factory pattern (or I am mistaking).
Coinbase tx input previous hash is empty bytes string as we did with root_tx in Phase 1, output value and address are specified when method is called.
Value is 6.25 for now (23 days left at the moment)
Address is public key of miner which has created this block.
Finalize is called to set hash and coinbase tx is returned.

Tests

…. That was something ….

Maybe because I spent ~10 hours writing those tests, they seem pretty self-explanatory for me, if they are not, I am sorry and ready to take punishment for the absence of this particular part of documentation.

(Now I understand that writing tests has taken more time than implementing phases 1 and 3, but also without writing those tests those phases wouldn't be so clear for me now.)

User Guide

There are 2 folders:
Faza-1-python
Faza-3-python
Each of them has requirements.txt, to install rsa library for python.

It is recommended to create virtual environment in folder which has both those folders.

```
python -m venv /path/to/folder
```

Might help to create venv

```
Source ./.venv/bin/activate
```

Might help to start venv

```
pip install -r path/to/requirements.txt
```

Might help to install rsa.

```
python faza-1-python/tests.py
```

```
python faza-3-python/tests.py
```

Might help to run all tests of particular phase.

```
python -m unittest faza-1-python/tests.Phase1.test_#
```

```
python -m unittest faza-3-python/tests.Phase3.test_#
```

Might help to run one specific test, where # is test number (1 - 15 for 1st phase and 1 - 27 for 3rd phase)

Implementation Environment

Whole project was written and tested in python3.12.

Code was written in Neovim.

As mentioned additional library rsa was used: https://stuvel.eu/software/rsa/

Showing off

I don't know what to write there, I regret that I didn't managed time good to do GUI and MultiSig, as it seems interesting to create some GUI connected with Phase 1 and Phase 3 (maybe Phase 2 as well).

Conclusion

That was an interesting assignment, one of the best for me for now on FIIT as it was interesting to learn how works transaction signatures (Especially, when I wasn’t sure if additional libraries are ok and was rewriting COS 432 RSA and PRF in python :)), what should be taken into consideration while validating transactions, Blockchain was even more fun to code as it is inspiring to see that some systems which seemed impossible to understand become more familiar and clear.