EE374 Blockchain Foundations **Stanford, Spring 2022**

Homework #3 *

Soft Deadline: Wed, Apr 20, 3 pm
Final Deadline: Fri, Apr 22, 3 pm

1 Block Validation

In this exercise, you will implement block validation for your Marabu node.

1. Create the logic to represent a block. See [1] for the structure of a block.
2. Check that the block contains all required fields and that they are of the format specified in [1].
3. Ensure the target is the one required, i.e.

   `"00000002af000000000000000000000000000000000000000000000000000000"`
4. Check the proof-of-work.
5. Check that for all the txids in the block, you have the corresponding transaction in your local object database. If not, then send a `"getobject"` message to your peers in order to get the transaction.
6. For each transaction in the block, check that the transaction is valid, and update your UTXO set based on the transaction. More details on this in Section 2. If any transaction is invalid, the whole block will be considered invalid.
7. Check for coinbase transactions. There can be at most one coinbase transaction in a block. If present, then the txid of the coinbase transaction must be at index 0 in `txids`. The coinbase transaction cannot be spent in another transaction in the same block (this is in order to make the law of conservation for the coinbase transaction easier to verify).
8. Validate the coinbase transaction if there is one.
   a) Check that the coinbase transaction has no inputs, exactly one output and a height. Check that the height and the public key are of the valid format. (We will check if the height is correct in the next homework when we validate chains, not now.)

*Version: 1 – Last update: Apr 13

b) Verify the law of conservation for the coinbase transaction. The output of the coinbase transaction can be at most the sum of transaction fees in the block plus the block reward. In our protocol, the block reward is a constant 50×10^{12} picabu. The fee of a transaction is the sum of its input values minus the sum of its output values.

9. When you receive a block object from the network, validate it. If valid, then store the block in your local database and gossip the block. Here, "gossip" means that you send an `ihaveobject` message with the corresponding blockid.

2 Maintaining UTXO Sets

In this exercise, you will implement a UTXO set and update it by executing the transactions of each block that you receive. This homework will not cover all the features of the UTXO set. We will revisit this part in future homeworks.

1. For each block in your database, store a UTXO set that will be computed by executing the transactions in that block. You should not update these UTXO sets when you receive transactions objects. Right now, you don't have to maintain the mempool UTXO set (this will be done in Homework 5).

2. When you receive a new block, you will compute the UTXO set after that block in the following way. To begin with, initialize the UTXO set to the UTXO set after the parent block (the block corresponding to the `previd`). Note that the UTXO set after the genesis block is empty. For each transaction in the block:
   a) Validate the transaction as per your validation logic implemented in Homework 2. Additionally, check that each input of the transaction corresponds to an output that is present in the UTXO set. This means that the output exists and also that the output has not been spent yet.
   b) Apply the transaction by removing UTXOs that are spent and adding UTXOs that are created. Update the UTXO set accordingly.
   c) Repeat steps a-b for the next transaction using the updated UTXO set.

For testing your node, you can use the genesis block which is a valid block. Below is another block mined on the genesis block that is valid. You can also mine more blocks to test your validation.

```
{
  nonce: "c5ee71be4ca85b160d352923a84f86f44b7fc4fe60002214bc1236ceedc5c615",
  T: "00000002af0000000000000000000000000000000000000000000000000000",
  created: 1649827795114,
  miner: "svatsan",
  note: "First block. Yayy, I have 50 bu now!!",
```

```
  previd: "00000000a420b7cefa2b7730243316921ed59ffe836e111ca3801f82a4f5360e",
  txids: [
    "1bb37b637d07100cd26fc063dfd4c39a7931cc88dae3417871219715a5e374af"
  ],
  type: "block"
}
```

The transaction in this block is

```
{
  type: 'transaction',
  height: 0,
  outputs: [
    {
      pubkey: '8dbcd2401c89c04d6e53c81c90aa0b551cc8fc47c0469217c8f5cfbae1e911f9',
      value: 50000000000
    }
  ]
}
```

3 Sample Test Cases

IMPORTANT: Make sure that your node is running at all times! Therefore, make sure that there are no bugs that crash your node. If our automatic grading script can not connect to your node, you will not receive any credit. Taking enough time to test your node will help you ensure this.
Below is a (non-exhaustive) list of test cases that your node will be required to pass. We will also use these test cases to grade your submission. Consider two nodes Grader 1 and Grader 2.

1. On receiving an `object` message from Grader 1 containing any invalid block, Grader 1 must receive an error message and the transaction must not be gossiped to Grader 2. Beware: invalid blocks may come in many different forms! Some examples are as follows:
   a) The block has an incorrect target.
   b) The block's proof-of-work is not valid.
   c) There is an invalid transaction in the block.
   d) There are two transactions in the block that spend the same output.
   e) A transaction attempts to spend an output
   f) The coinbase transaction has an output that exceeds the block rewards and the fees.
2. On receiving an `object` message from Grader 1 containing a valid block, the block must be gossiped to Grader 2 by sending an `ihaveobject` message with the correct blockid.

References

[1] EE 374: Blockchain Foundations. https://web.stanford.edu/class/ee374/protocol.