Problem Set #4 [*]

Soft Deadline: TBD
Final Deadline: TBD

1 Recursive Block/Chain Validation

Here, you will build upon your block validation logic from the previous homework to recursively validate the parent block, if you have not done so already.

1. When you receive a new block, check if the parent block is available in your database (after the proof-of-work check). If not, request your peers for the parent block using a `getobject` message.
2. Recursively validate the parent block before validating the new block. This would include downloading all the blocks in the prefix of the new block (until you hit genesis or a block that you already knew was valid), validating all the blocks in the prefix, and updating the UTXO set after all these blocks. Validating the parent block also ensures that you have the UTXO set after the parent block, and hence you can validate the transactions in the new block and update the UTXO set. If a parent block is not found or invalid, send back an `UNFINDABLE_OBJECT` error.
3. The height of a block is the number of blocks it is away from the genesis. The genesis has height 0 and heights increase along the chain. When you receive a new block, determine its height and check that if the block has a coinbase transaction, its `height` field matches the height of the block. If not, send back an `INVALID_BLOCK_COINBASE` error.
4. Check that the timestamp of each block (the `created` field) is later than that of its parent, and earlier than the current time. If not, send back an `INVALID_BLOCK_TIMESTAMP` error.

2 Longest Chain Rule

In this exercise, you will implement the longest chain rule, and sync the chain tip with your peers.

1. Implement the longest chain rule, i.e. keep track of the longest chain of valid blocks that you have.

[*]Version: 2 – Last update: July 1

2. When your node boots up and after connecting to some peers, send a `getchaintip` message to your peers to sync their longest chain.

3. On receiving a `getchaintip` message from a peer, respond with a `chaintip` message with the blockid of the tip of your longest chain.

4. On receiving a `chaintip` message or on receiving a new block in an `object` message, validate the chain by recursively downloading and validating each block in the chain. Update your longest chain if required.

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.

0. Reset your block/object database before submitting for grading. This is so that blocks that your node might have earlier considered valid but are actually invalid (because, for example, you had not implemented coinbase height check earlier) are removed from the database. This is to prevent your node from adopting a longer chain which is actually invalid.

1. Grader 1 sends one of the following invalid blockchains by advertizing a new block with the `object` message. Grader 1 must receive the appropriate error message, and Grader 2 must not receive an `ihaveobject` message with the corresponding blockid. Note that if a parent block is invalid, send back an `UNFINDABLE_OBJECT` error, but if the block itself is invalid, send back the appropriate error (`INVALID_FORMAT`, `INVALID_BLOCK_COINBASE`, `INVALID_BLOCK_TIMESTAMP`, etc.).
   a) A blockchain that points to an unavailable block
   b) A blockchain with non-increasing timestamps
   c) A blockchain with a block in the year 2077
   d) A blockchain with an invalid proof-of-work in one of the blocks
   e) A blokchain that does not go back to the real genesis but stops at a different genesis (with valid PoW but a null previd)
   f) A blockchain with an incorrect height in the coinbase transaction in a block

2. Grader 1 sends a number of valid blockchains. When Grader 1 subsequently sends a `getchaintip` message, it must receive a `chaintip` message with the blockid of the tip of the longest chain.