**File to be modified**:

BlockChain.java

The BlockChain class is responsible for maintaining a block chain. Since the entire block chain could be huge in size, you should only keep around the most recent blocks. The exact number to store is up to your design, as long as you’re able to implement all the API functions.

**Since there can be (multiple) forks, blocks form a tree rather than a list. Your design should take this into account. You have to maintain a UTXO pool corresponding to every block on top of which a new block might be created.**

**Assumptions and Guidelines**:

* A new genesis block would not be mined. If you receive a block which claims to be a genesis block (parent is a null hash) in the addBlock(Block b) function, you can return false.
* If there are multiple blocks at the same height, return the oldest block in getMaxHeightBlock() function.
* Assume for simplicity, a coinbase transaction of a block is available to be used in the next block mined on top of it. (This is contrary to the actual Bitcoin protocol when there is a gap of 100 blocks only after which the coinbase transaction can be used).
* Maintain only one global Transaction Pool for the block chain and keep adding transactions to it on receiving transactions and keep removing transactions from it if a new block is received or created. This might cause some transactions to be lost. For example, a block is received on Chain A including transaction Tx1. We remove Tx1 from the transaction pool. Now suppose chain B offshoots chain A. Do not put Tx1 back in the pool, although ideally it should be put back in. This is to simplify your work as well as our work in testing. (Miners are not responsible for including transactions in the blocks. If a transaction is lost, it is the responsibility of the transaction owner to re-broadcast it in the network).
* The coinbase value is kept as constant in our entire block chain (= 25 bitcoins) whereas we know it changes every four years.
* When checking for validity of a newly received block, just checking if the transactions form a valid set is enough. The set need not be a maximum possible set of transactions. Also, you should not check for hash of the block to contain specific zeros (no proof of work here).

**An autograder is provided to you to self check the correctness of your program. Follow the following steps:**

**Step 1.**Copy **BlockChain.java** into **grading** folder.

**Step 2**. While holding down **Windows key** press **R** (or click **Start**) and then type **cmd**. Press **Enter**.

**Step 3**. Move into the folder that contains the **project**-related classes by typing this command:

cd C:\grading

**Step 4**. Compile the classes with this command (**Replace ; with : for Linux or Mac OS)**:

javac -cp blockChainGrader.jar;algs4.jar;rsa.jar;. DropboxTestBlockChain.java

**Step 5**. Finally, run the tests with this command:

java -cp blockChainGrader.jar;rsa.jar;. DropboxTestBlockChain