## **Backend Track (Backend / Infrastructure / SRE / Data platform)** **Pre-Interview Assignment**

In this assessment, you will build a simplified blockchain. It is a simplified version of the original bitcoin chain. Please feel free to use any mainstream programming language. The assignment has five exercises. *Please try to complete as many as you can*. This assignment evaluates CS foundation, the ability to learn and master new concepts and the ability to deal with complexity and edge cases.

**Exercise 1. Finding the nonce**. Given a string, pad it on the right with random alphanumeric characters such that the padded string is 100 in length and its SHA256 hash starts with “0000”. Please search online for how to compute the SHA265 hash in your language.

**Input**:

* A string with no more than 70 alphanumeric characters

**Output**:

* The alphanumeric padding that satisfies the above requirements. This padding is also called a ***nonce***.

**Example**:

* Input: ‘3’
* Output: ‘RAXEFAMPMCCKZKCELFYEWBMRPILETDPLQXOTAFMRQLTBFCISBMEBETZUHYZHGWFQIRVUXNDJHKHUUGSVVBQFHEZCGRMUSNOCWJA’
* Because: SHA256(‘**3**RAXEFAMPMCCKZKCELFYEWBMRPILETDPLQXOTAFMRQLTBFCISBMEBETZUHYZHGWFQIRVUXNDJHKHUUGSVVBQFHEZCGRMUSNOCWJA’) = ‘**0000**1fb13c4c92dd6b3c462a971a23e659848738a64eb0f14f0605ffd9cde155’

**Questions**:

* On average, how long does it take to find the nonce?
* What if we require the hash to start with five zeros instead of four?

**Take away**:

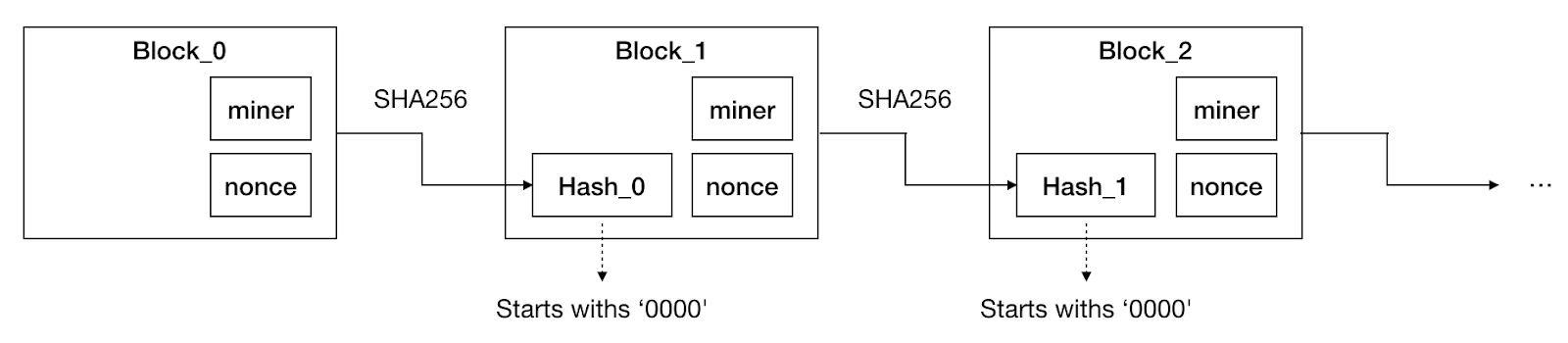
* You know how to compute the SHA256 hash in your language.
* You know how to find a nonce satisfying the above requirements. The next exercise builds on this.

**Exercise 2. Constructing and verifying a blockchain**. You will construct a blockchain with ten blocks and verify it.

Here is the definition of the blockchain data structure. A blockchain is an ordered list of blocks where each block has two fields:

* **nonce**: An alphanumeric string.
* **miner**: An integer number which is always 0 in this exercise.

The following figure might help you understand the concepts:



We will use the following notations:

* **Block\_i**: The i-th block
* **Block\_i.nonce** and **Block\_i.miner**: The two fields in **Block\_i**
* **ToString(n)**: Convert an integer number **n** to string. If **n** = 0, then **ToString(n)** gives ‘0’.
* **ToString(Block\_i.miner)**: Because **Block\_i.miner** is always 0 in this exercise, **ToString(Block\_i.miner)** is always ‘0’.
* **s1 + s2**: the concatenation of the two strings **s1** and **s2**
* **SHA256(s)**: the SHA256 hash of the string **s**
* **Genesis block**: the first block in a blockchain, i.e., **Block\_0**

We define **Hash\_i** -- the hash of the block **Block\_i** -- as follows:

* For the genesis block, **Hash\_0 = SHA256( ToString(Block\_0.miner) + Block\_0.nonce)**. To mine the genesis block, we need to find a value for **Block\_0.nonce** such that:
  + **ToString(Block\_0.miner) + Block\_0.nonce** is 100 long.
  + **Hash\_0** starts with ‘0000’.
* For i > 0, **Hash\_i = SHA256( Hash\_{i-1} + ToString(Block\_i.miner) + Block\_i.nonce)**. To mine the i-th block, we need to find a value for **Block\_i.nonce** such that:
  + **Hash\_{i-1} + ToString(Block\_i.miner) + Block\_i.nonce** is 100 long. (Please note that **Hash\_{i-1}** is a SHA256 hash and is 64 in length. Therefore **Block\_i.nonce** is 100 - 1 - 64 = 35 in length.)
  + **Hash\_i** starts with ‘0000’.

One should be able to build a blockchain given the above definition. To mine ten blocks we need to find ten nonces. Mining each block requires the hash of the previous block. The genesis block’s nonce is 99 in length and all other blocks’ nonces are 35 in length.

Please encapsulate your mining code into a function called **mineTheNextBlock** and call it multiple times to mine the entire chain. Please create another function called **verifyChain** to verify a given blockchain. Verifying a blockchain means computing the hash of each block and verifying that they all start with ‘0000’.

**Input:**

* None

**Output:**

* A verified blockchain with ten blocks

**Example**:

[{'nonce': 'HKDPEAJZJLILXPBEHAJNNQTFAMHHZWLJILHNQHYFQQZOIYQCZUYVKBLNYTEKMVDTDWAAFHMACIEFASYSXUJWLXTGRVBWQYRXTPZ', 'miner': 0},

{'nonce': 'MZPLFFYHZRLYGTUZDRYRRACCGWBUMEERFTR', 'miner': 0}, {'nonce': 'PXOCCFVAYDDEEJPNHLOGCRRQVICCLEQDCJX', 'miner': 0}, {'nonce': 'XOEIMJXOJXQCZXHHBHRDMHZTVHJXKENGLMS', 'miner': 0}, {'nonce': 'UNXWCHEUTPLVWEZWDAHSRMPQJRFUYBCPKAX', 'miner': 0}, {'nonce': 'XJBRWQOUROBXDAHQZTWUEWIROKEXLVMEKMR', 'miner': 0}, {'nonce': 'HHZDLXBMJMGJOFBRAOECCOXAMNBUVHRCDME', 'miner': 0}, {'nonce': 'BZCRONSKMWAIOEUTTJUIHSVCNHLXXXCKUTL', 'miner': 0}, {'nonce': 'FOZGXSJCCKHGPJIMYQNBFLCURVWQTFPFZRQ', 'miner': 0}, {'nonce': 'YIEWUPHLCCHLWIOEYSFOTUWXSMSCPTVIUNH', 'miner': 0}]

**Take away**:

* You can construct a blockchain with an arbitrary number of blocks.
* You have two reusable functions -- **mineTheNextBlock** and **verifyChain** -- for building blockchains.

*In the next three exercises, you will decentralize the blockchain. This is done by mining from multiple threads. Exercise 3 and exercise 4 familiarize you with multi-threaded programming and in exercise 5 you will decentralize the blockchain. If you are familiar with multi-threaded programming, please feel free to skip exercise 3 and exercise 4.*

**Exercise 3. Multithreading**: Please start K threads (2 <= K < 10) and let each thread store a string value into a local variable and then print the local variable. Please search online on how to use multiple threads.

Please name each thread with a numerical ID. If there are K threads, the thread IDs will be 0, 1, 2, …, K-1. Printing the thread IDs helps debugging. Here is an example output:

Thread-0: [0] -- time: Thu Jul 1 02:29:26 2021

Thread-2: [2] -- time: Thu Jul 1 02:29:26 2021

Thread-1: [1] -- time: Thu Jul 1 02:29:26 2021

Thread-3: [3] -- time: Thu Jul 1 02:29:26 2021

Please also find a way to shut down the threads before the program exists.

**Take away**:

* You know how to start and stop multiple threads.
* You know how to store information in thread-local variables.

**Exercise 4. Inter-thread communication**. Please start two threads and let one thread send a string message to the other and let the other thread print the received message. Please search online on how to send messages between two threads. Please increase the number of threads to K (2 <= k < 6) and let every thread send messages to every other thread and let every thread print all of the received messages. Please verify that the printed messages are correct.

Here is an example:

Thread-0 sending message WSDSFVFQUC to all other threads.

Thread-1 sending message YMGQJLAEFL to all other threads.

Thread-1 received message WSDSFVFQUC.

Thread-2 sending message NGBABUSZOC to all other threads.

Thread-2 received message WSDSFVFQUC.

Thread-2 received message YMGQJLAEFL.

Thread-0 received message YMGQJLAEFL.

Thread-0 received message NGBABUSZOC.

Thread-1 received message NGBABUSZOC.

As a general rule of thumb in multi-threaded programming, if your program doesn’t run as expected, letting each thread print out their internal states might help.

**Take away**:

* You know how to communicate between threads.

**Exercise 5. Decentralizing the blockchain.** Please use K threads (2 <= k <= 10) to collaboratively mine a blockchain of length ten.

Here is a high-level description of the algorithm. A sample pseudocode will be provided later:

* All threads start by mining the genesis block.
* Let’s assume that thread-i is the first thread to find the genesis block. Now thread-i has a blockchain of length one. It would immediately broadcast this chain to all other threads. All other threads, upon receiving this message, will stop what they are doing and start mining the second block on this new chain.
* Same way, the first thread to find the second block will broadcast the new chain to all other threads. All threads will then jump onto this new chain to mine the third block.
* In other words, any thread’s progress will be quickly synced to all threads.
* This continues until all ten blocks have been mined.

Please note that, in contrast to exercise 2, we will use a thread’s ID as **Block\_i.miner.** For example, thread-2 will use **Block\_i.miner = 2** when trying to mine a new block and thread-5 will use **Block\_i.miner = 5.** When a block is successfully mined, the miner’s thread ID will be persisted into the chain and be broadcasted to all threads. We can figure out which block was mined by which thread by looking at the **miners** fields in the chain.

There are three caveats worth paying attention to:

* **Mine and receive messages at the same time**. You can let each thread alternate between mining and reading messages. This allows a thread to make progress on mining without missing any messages.
* **When should an incoming chain be accepted?** When a thread receives a blockchain in a message, it should only start mining on it if it satisfies the following two conditions: First, it is longer than the thread’s own chain. Second, it is valid.
* **Sending blockchain in messages**. The messages in exercise 4 are strings. Serialization and deserialization might be needed to send a blockchain in a message.

For your reference, here is the pseudocode of a possible solution:

|  |
| --- |
| # Thread start  **chain** = empty-chain  while (chain has less than ten blocks):  # If a new block can’t be mined within a short period of time,  # the following function exits with **new-block** being NONE.  **new-block** = *mineTheNextBlock* (**chain**)  if **new-block** is not NONE:  **chain** = **chain** + **new-block**  send **chain** to all other threads  **incomingChains** = read the chains from all of the incoming messages  for **incomingChain** in **incomingChains**:  if *verifyChain*(**incomingChain**) and **incomingChain** is longer than **chain**:  **chain** = **incomingChain**  print **chain** |

**Input**:

* None

**Output**:

* A blockchain with ten blocks

**Example**:

[{'nonce': 'LHYXDEWDUBAXKQCIJQKWDMAGEZNBXDICAQAIEEROMCEEAFKCWPTYITDDJOWKQMUYWTONVWDDCBUZBLQMBHRRTGVYBOMXRBLQRIB', 'miner': 4},

{'nonce': 'TWNMEYSSICBLTNUHCJWRJBQVSEWVRNQMCXO', 'miner': 4}, {'nonce': 'CUFORMUYFRBJKTWDLNBHEDKNFVVUMIADEBF', 'miner': 4}, {'nonce': 'QQPVJDOWIBIXNFAPHOJZETTWNYDVHVZFWZX', 'miner': 8}, {'nonce': 'XWCBRHWIWVQPNUCYXTODYRFKGSIAXHJTVOY', 'miner': 0}, {'nonce': 'YCYXOUQXBFLSBTMPGRARNAUVHAFENCABAYY', 'miner': 8}, {'nonce': 'ICFXLWAIUHDVFMEUJWZLJXMFTCMRQILFSDK', 'miner': 7}, {'nonce': 'PYSQDKZWWBLBJXLZKMWJZSNAHCSHHJREJDM', 'miner': 6}, {'nonce': 'ASAYROFPRTZZQLIYGLZPQLFBCTNGFLCVNBW', 'miner': 5}, {'nonce': 'SIGPCXMAVHBVEQFAZHMZQJUYYKZDMZSKJSK', 'miner': 2}]

**Take away**:

* You know how to mine a blockchain in a decentralized fashion.

**Submission**:

* Please document your thought process and explain how your code works.
* Please include commented source code.
* Please include the output of each exercise.
* Please try to answer the following questions (feel free to simulate if it helps):
  + Does this blockchain scale? What happens when the number of threads go up?
  + If one thread stopped working, does it affect the blockchain? What if the thread comes back on later?
  + **Bonus points**:
    - What if we require the SHA256 hashes to start with more zeros?
    - Is it easy for a “malicious thread” to modify the **miner** fields in the chain?

**End of assignment survey**

We want to thank you for working on this assignment. Please let us know what do you think:

* How long did it take?
* What’s the most difficult part?
* Any other feedback for the problem or the process?