☰ Introduction to Computer Science for Engineers

Blockchain - Blocks and Proof of Work ✓✓✓ ☺ 🎓

This assignment **closed** January 12, 2025 at 23:15.

> The due date for the assignment got extendet. It is to be done for the 01.01.2024, 23:15.

During the tasks of this assignment sheet we will implement a very simple version of a block chain - from scratch.

For a block chain to make sense we need several things:

1. Blocks
2. A chain (a.k.a. a linked list)
3. A way to bring the two together

Task 1.A - Blocks

In this first assignment you will have to implement a basic block for a block chain.

A block contains a message (represented as a String), a proof of work (see Task 1.B), and the hashcode of a previous block.

1. Create a `class Block` that has the following attributes: `message: str`, `proofOfWork: int`, `previousHashCode: bytes`. Initially, `proofOfWork` should be zero.

   Info: The `proofOfWork` attribute in litrature is also called `Nonce`.

2. Implement the function `hash(self) -> bytes` that calculates the hash value of the block. Take all three attributes (`message`, `proofOfWork` and `previousHashCode` into account). Use the *SHA-256*[1] algorithm.

   Hint: Take a look into https://docs.python.org/3/library/hashlib.html

3. Implement the function `__str__` to nicely visualize each block.

Task 1.B - Proof of Work

The second part of this assignment requires you to implement some way to proof that your computer has performed a certain amount of work.

For that you will try to find a number that fulfills a certain condition, i.e. the bit string representation of its *SHA-256*[1] hash starts with a certain number of zeros.

1. Implement the function `number_of_leading_zeros(block: Block) -> int` that checks the hash of the Block `block` from left to right for the number of zeros it starts with, ie. number of leading zeros of the hash `b'\x01'` is 15, since `b'\x01'` in bits is `0000000000000001`

   Hint: Use the `bytes_to_bits` function to convert the `bytes` from the hash value to a string containing the bit representation.

2. Implement the function `proof_of_work(block: Block, x: int) -> None` that tries to find a number `block.proofOfWork` such that the bit representation of `blocks`'s hash starts with at least `x` zeros.

   Hint: You have to use the function `number_of_leading_zeros` to calculate if the hash is a valid proof of work.

3. Implement the function `verify(block: Block, x: int) -> bool` which should return `True` if and only if the bit string representation of `block`'s hash start with at least `x` zeros.

   Hint: Use `number_of_leading_zeros` here too.

- Try guessing random numbers. It may actually be faster than just counting up.
- The `main` method already contains some useful ways of calling the functions you need to implement.

---

1. SHA-256 is a special kind of hashing function: it is a cryptographic hash. The hashing functions from the lecture were simple mappings from $\mathcal{X} \rightarrow \mathbb{Z}$. Cryptographic hash functions have to fulfill some more requirements. ↩ ↩2

Template files

Get all files in an archive templates.zip or templates.tgz .

`block_chain.py`