Atelier 1

Contents

Exercise 1: Merkle Tree

I implemented a basic Merkle Tree from scratch in Node.js using the sha.js library for cryptographic hashing. The goal was to create a Merkle Tree structure that computes a root hash from a given set of leaves, allowing for efficient verification of data integrity.

Implementation Details

1. ***Hash Function:***
   - A hash function combines two node values (leftNode and rightNode) to create a SHA-256 hash. This function is crucial for linking nodes in the Merkle Tree.
   - It uses the update method to concatenate the node values and then computes the hash using digest.
2. ***MerkleTree Class:***
   - The MerkleTree class constructs the tree based on the provided height and leaves.
   - The N(level, index) method is a recursive function that computes the hash of nodes at different levels.
     - If the current level is equal to the tree height, it returns the leaf node.
     - If not, it computes the hash of the left and right children.

Testing the Implementation

Two test functions were implemented to demonstrate the functionality of the Merkle Tree:

1) ***example1:***
   - This function creates a Merkle Tree with a height of 3 and a set of leaves.
   - It computes and prints the root hash, showcasing the initial state of the tree.
2) ***example2:***
   - Similar to example1, but this function modifies one of the leaves (changing 2 to 9) to demonstrate the impact of leaf changes on the root hash.
   - It prints the root hash before and after the modification.
3) ***Remarques:***

   The Merkle Tree implementation successfully computes root hashes from a set of leaves and demonstrates the impact of changes to leaves on the tree structure. This exercise highlighted the utility of Merkle Trees in ensuring data integrity and efficient verification.

Exercise 2: Proof of Work

Implemented the Proof of Work (PoW) consensus mechanism in Python, creating a simple blockchain structure that mines blocks while varying the difficulty levels of hashing.

Implementation Details

1. ***Block Class:***
    - Represents individual blocks in the blockchain, containing essential attributes such as index, previous hash, data, timestamp, nonce, and hash.
    - The compute_hash method combines these attributes into a string and generates a SHA-256 hash.
    - The proof_of_work method increments the nonce until it finds a hash that starts with a specified number of leading zeros (the difficulty).
2. ***Blockchain Class:***
    - Manages the sequence of blocks in the blockchain.
    - Includes methods for creating the genesis block, retrieving the last block, adding new blocks, and validating the integrity of the chain.

Testing the Implementation

Main Function

The main function initializes the blockchain and creates multiple blocks at increasing difficulty levels. It measures the time taken to mine each block and prints the results.

Remarques:

The Proof of Work implementation demonstrated how blocks are mined based on varying difficulty levels, showcasing the time complexity involved in finding valid hashes. The blockchain's integrity was maintained, and the implementation provided valuable insights into the PoW mechanism.

Exercise 3: Proof of Stake

In this exercise, I added a Proof of Stake (PoS) implementation in Python. The goal was to illustrate how PoS differs from PoW and to measure the execution time for each consensus mechanism.

Implementation Details

1. ***ProofOfStake Class:***
    - Manages the staking process, where validators are selected based on their stake in the network.

- Validators are incentivized to act honestly, as they risk losing their stake if they attempt to validate fraudulent transactions.

2. Main Function:

- Initializes the PoS system and adds stakes for several validators.
- Selects a validator based on the stakes and measures the time taken for this selection process.

3. Remarques

The Proof of Stake implementation illustrated the process of selecting a validator based on their stake, demonstrating the efficiency of PoS compared to PoW in terms of execution time. This exercise provided a comparative analysis of both consensus mechanisms.

Additional info:

for more details on the implementation, please check the github repo