Merkle Tree

Part 1 - Merkle Tree Implementation

Check out my Github page for the Java code:
https://github.com/KevinKhau/merkle-tree-java/tree/master/src

This is an interesting exercise, so I'm working with it on a personal React project too.

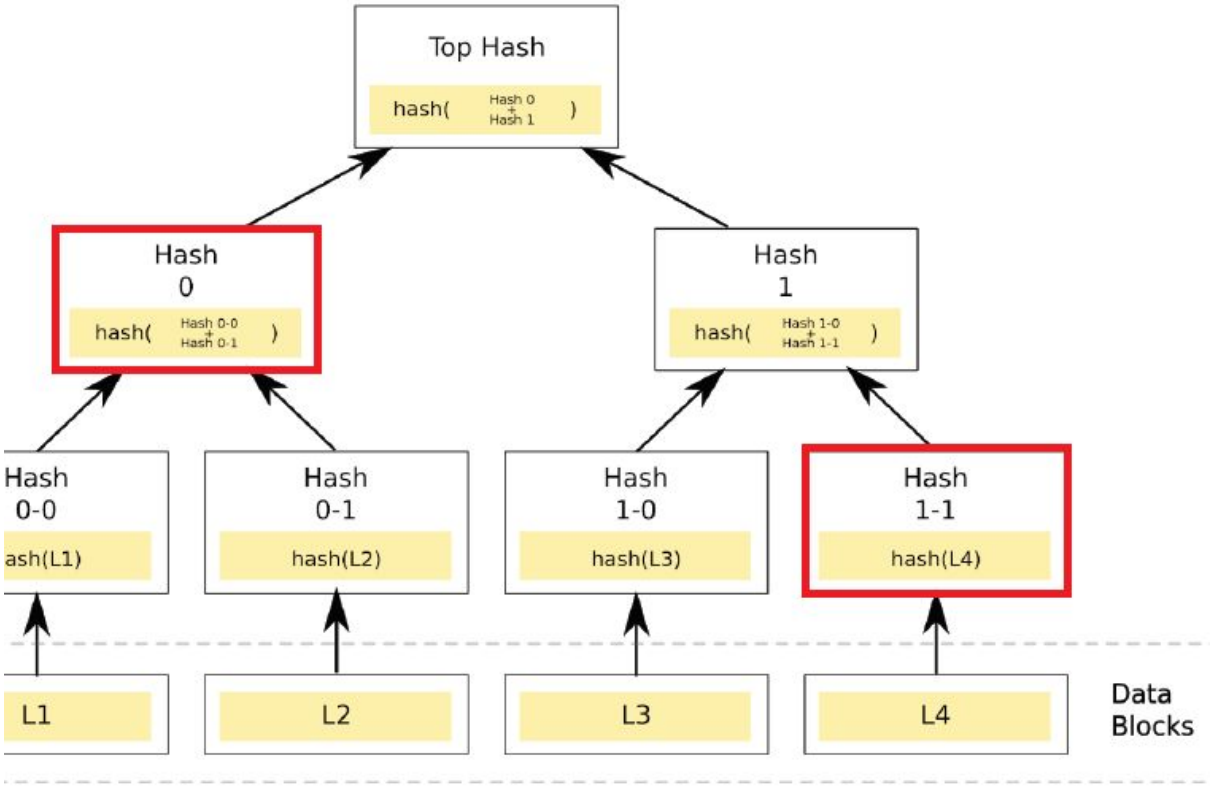
Part 2 - Questions

1) In the illustration, let's imagine I know the whole Merkle tree. Someone gives me L2 data block but I don't trust him. How can I check if L2 data is valid?

The cryptographic function SHA256 is a one-way hash. If someone gives me a block of data, I can first hash it using the same SHA256 function, let the result be **expectedL2Hash**. Given that the node hashes are public, I need to compare the latter with the hash of L2, let it be **actualL2Hash**.

If *expectedL2Hash* and *actualL2Hash* are equal, then the provided L2 data is valid. Thus I can verify provided data without knowing it, by using its hash.

2) I know only the L3 block and the Merkle root. What minimum information do I need to check that the L3 block and the Merkle root belong to the same Merkle tree?



To know whether the L3 block and the Merkle root belong to the same Merkle tree, we have to rebuild the nodes on the path from L3 to the root. *Hash1* is the hash of the

concatenation of **Hash1-0** and Hash-1-1, thus we need ***Hash1-1*** or its child.ren to build *Hash1*. Using *Hash1*, we need ***Hash0*** or its child.ren to build *Top Hash*. If the resulting ***Top Hash*** and a provided Merkle root are equal, then this latter and L3 block belong to the same Merkle tree.

Ensuring the integrity of the non mentioned data blocks is not necessary, which gives way to optimisation possibilities.

3) What are some Merkle tree use cases?

As highlighted by the previous questions, a Merkle tree can be used to **verify the integrity** of data, which may have been damaged or modified during a transfer, a storage or another data operation.

Due to the nature of a hash, comparing hashes is **faster than** comparing raw data. A more interesting property would be to check the integrity of a data block while requiring a minimal amount of known hashes. Due to the binary aspect of the Merkle tree, it deals effectively with issues of **scalability**, requiring logarithmic operations.

While applied, this would prove useful to check the integrity of a portion of file, or of a blockchain in a node network.