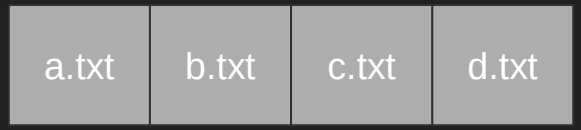
Merkle Trees

Background

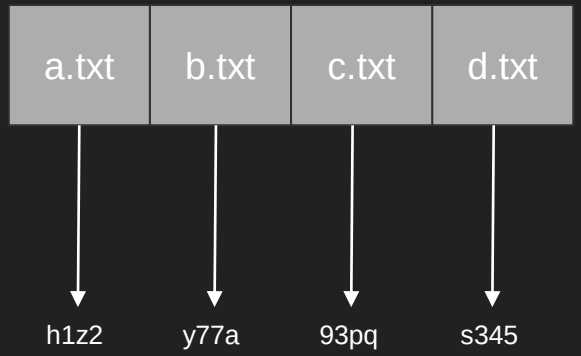
Merkle trees are an algorithm that we have brushed over a few times now - it is extremely useful for efficiently calculated the differences between sets of data or files, and determining where they differ in logarithmic time (as opposed to linear time).

Like Bloom Filters, they heavily rely on hashing for their good performance, and we can see them being used in the Git revision system, anti entropy processes in leaderless databases, and also in blockchain/decentralized currencies.

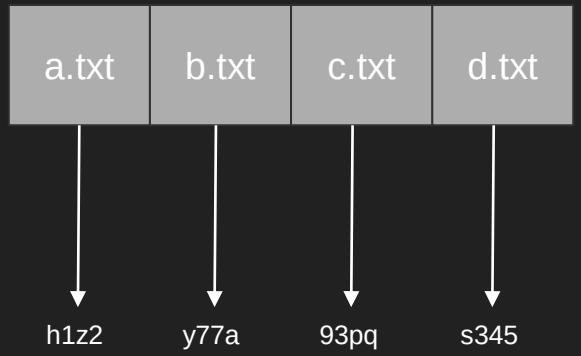
Merkle Tree Basic Algorithm



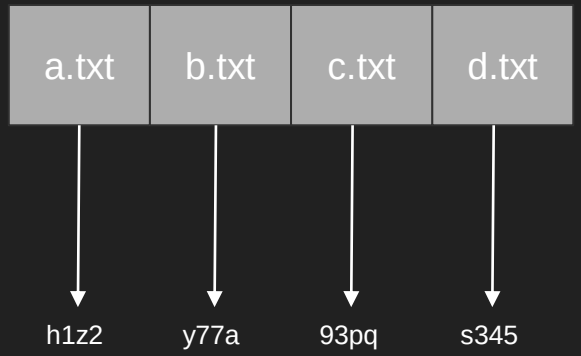
Merkle Tree Basic Algorithm

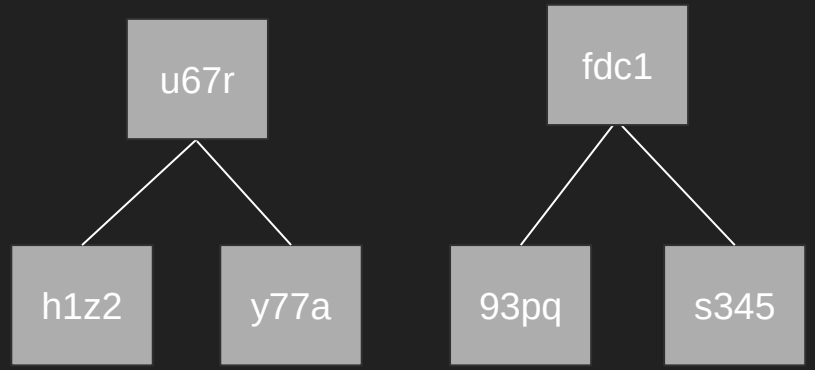


Merkle Tree Basic Algorithm

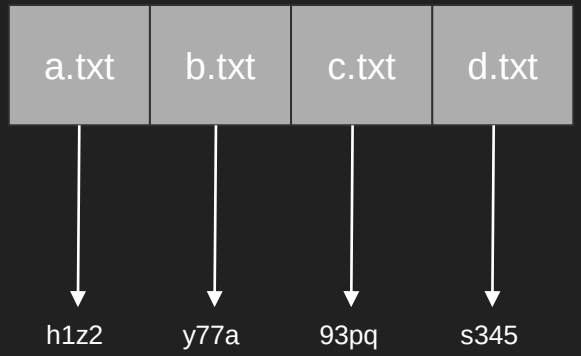


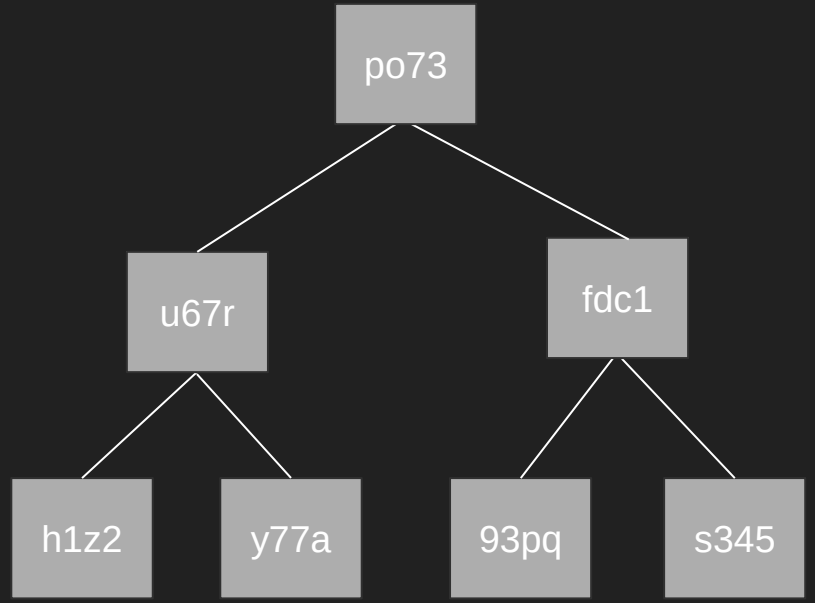
Merkle Tree Basic Algorithm



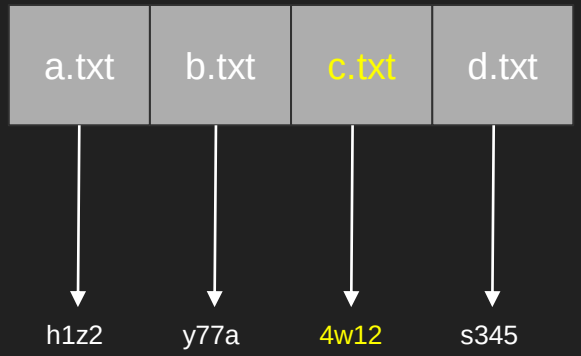


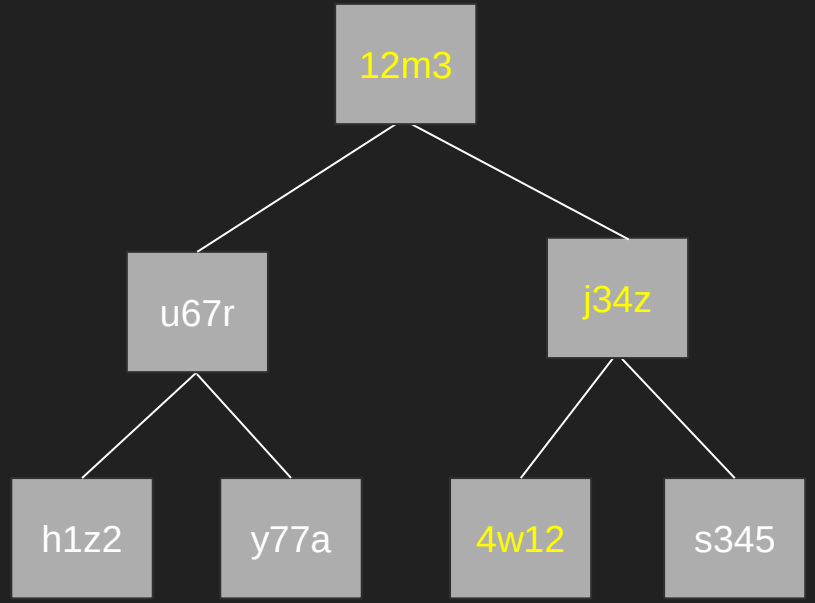
Merkle Tree Basic Algorithm





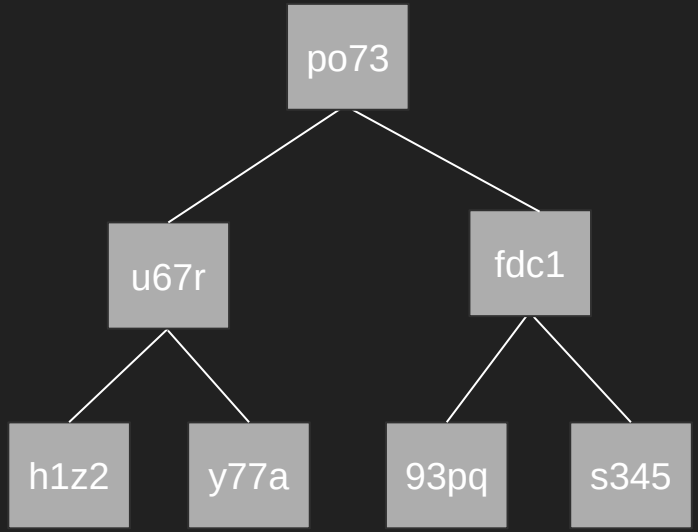
Merkle Tree Basic Algorithm

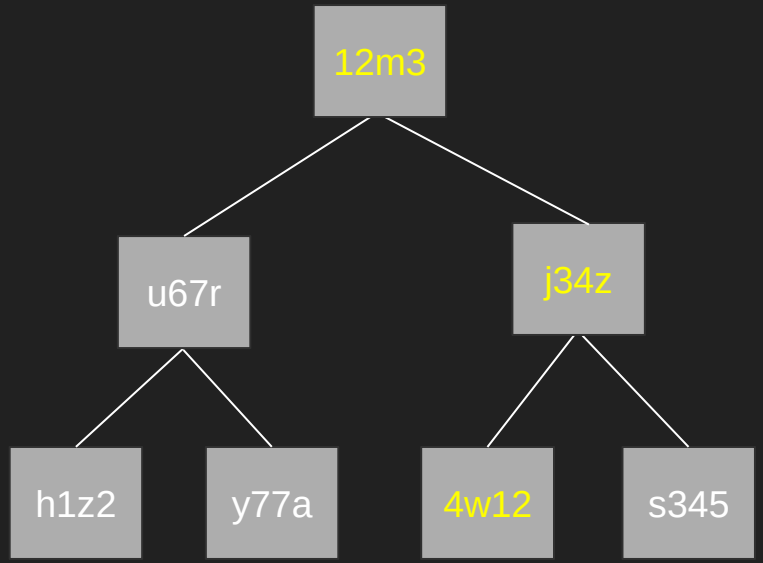




Now, make a change to c.txt!

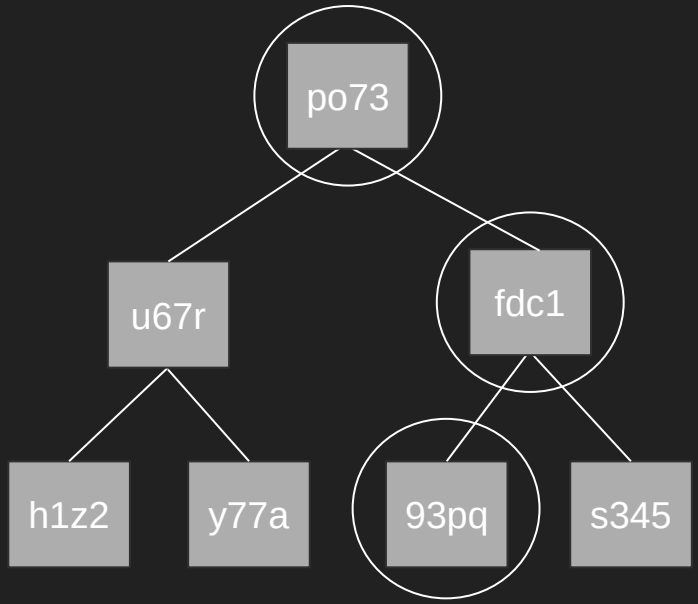
Merkle Tree Comparison

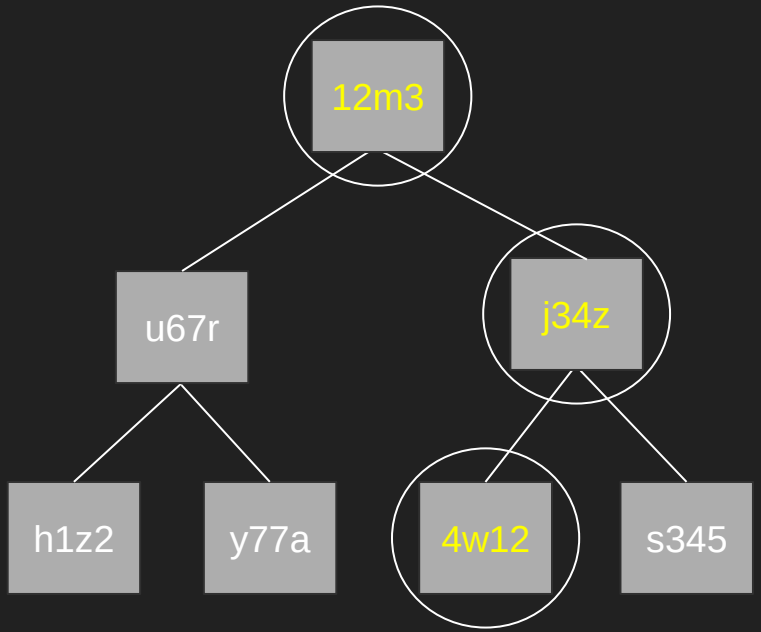




Start from the root, and traverse depth first finding differences in the hashes!

Merkle Tree Comparison





Start from the root, and traverse depth first finding differences in the hashes!

Merkle Trees in Git

The way Merkle Trees are presented above means that they throw out all of the old hashes of files/chunks, but in Git we want persistence so that we can inspect old commits:

- To fix this, each change to a file is treated as a separate file and hashed
  - If the hash is already in the merkle tree, nothing changes (good for file metadata changes)
  - If it is a new hash, add it as a leaf to the merkle tree and create a new branch/copy of the path from root to leaf
  - To go back in time to old commit, just start from the root hash of that commit

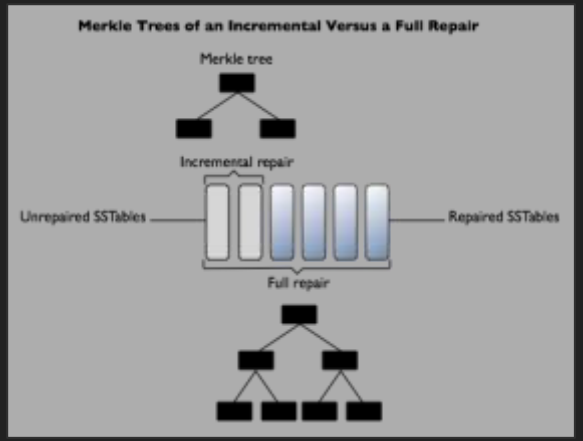
Merkle Trees in Cassandra

Recall: Anti-entropy is the background process of making sure that all replicas are consistent in a multi-master replication schema.

Cassandra uses Merkle trees for this! But wait, isn’t the data always changing? How can we be sure that the changes that one replica sends to another will still be valid by the time they travel over the network?

Incremental Read Repair

Recall: Cassandra uses immutable SSTable files. Hence, two replicas can easily perform comparison and anti-entropy in order to do this! In order to decrease duplicate data from being sent over the network, repaired SSTable files are marked as repaired, whereas ones that have not been marked as repaired are marked as unrepaired.



Incremental read repair allows us to send less data over the network, because the Merkle Trees are smaller!

In order to avoid having to repair SSTables that have already gone through the anti-entropy process, we separately compact repaired tables and unrepaired tables! (Size tiered compaction)

Conclusion

Merkle trees are a very clever way of detecting the differences between large sets of records by using a logarithmic tree of hashes. They are extremely applicable in multiple areas of systems design, such as detecting the differences in files, as well as finding the smallest units of difference in order to propagate them over the network, such as in anti-entropy.