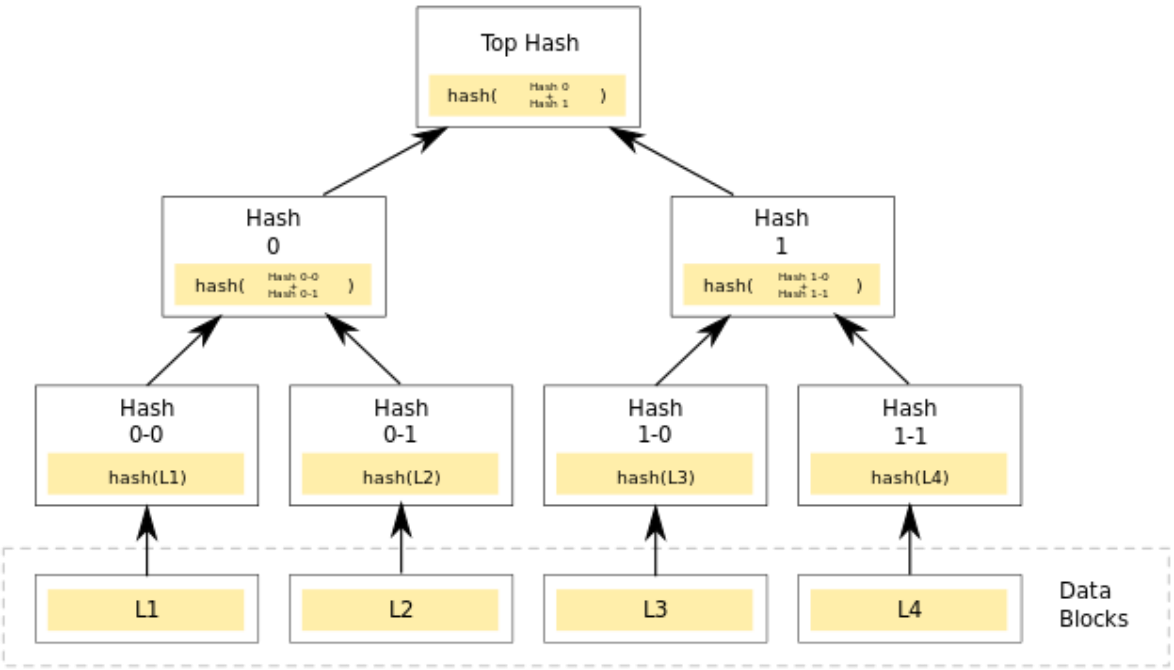
merkle_tree

The Merkle tree, commonly known as the hash tree, is a data structure for synchronising and verifying data. Each non-leaf node is a hash of its children in this data structure. The leaf nodes all have the same depth and are as far to the left as possible. It makes use of hash functions to keep data safe.



The top hash of this binary merkel tree is a hash of the entire tree.

- **Because of the tree's structure, enormous amounts of data can be mapped quickly, and little changes in the data can be easily detected.**
- **We can verify if the data is consistent with the root hash to see where it has changed, which means we won't have to traverse the full structure but only a small section of it.**
- **The root hash serves as the data's fingerprint.**

Applications:

- **Merkle trees are useful in distributed systems where the same data must be stored in multiple places.**
- **Merkle trees can be used to check for inconsistencies.**
- **Apache Cassandra uses Merkle trees to find inconsistencies across database replicas.**
- **It's used by both Bitcoin and blockchain.**

Algorithm of remove function.

- **It will take tree and key as parameters.**
- **If the tree is null then return null.**
- **If the key is smaller than the tree->key then tree->left is equal to remove(tree->left, key) and return tree.**
- **If the key is greater than the tree->key then tree->right is equal to remove(tree->right, key) and return tree.**
- **else if the tree->left is equal to null and the tree->right is equal to null then decrement the size and return tree->left.**
- **else if the tree->left is not equal to null and the tree->right is equal to null then decrement the size and return tree->left.**
- **else if tree->left is equal to null and tree->right is not equal to null then decrement the size and return tree->right.**
- **else assign tree->left to a pointer called left of data type node.**
- **While left->right is not equal to null, left is equal to left->right.**
- **tree->key is equal to left->key.**
- **tree->value is equal to left->value.**
- **tree->left is equal to remove(tree->left, tree->key).**
- **Return tree.**