Binary Trees

A binary tree is a hierarchical, non-linear data structure used in computer science.

**Non-linear** indicates that data is not stored sequentially, like it is in arrays and lists.

A binary tree consists of **nodes**, where each node contains its own data, and pointers to its child nodes. A single node in a binary tree can have up to 2 children, hence the term **binary** tree.

The following is an example of how a node of a binary tree could be represented in C++ code:

struct TreeNode {

int value; // The value stored in the node

TreeNode\* left; // Pointer to the left child

TreeNode\* right; // Pointer to the right child

// Constructor to initialize the node with a value and set children to null

TreeNode(int val) : value(val), left(nullptr), right(nullptr) {}

};

A binary tree is typically represented visually with the root node at the top of the tree, and child nodes below their parents. If `i` represents each row in the tree and the root node is said to be at row 0, then the maximum number of nodes that can be in a row is 2^i.

|  |  |
| --- | --- |
| **Row number** | **Maximum nodes** |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |

## Key Terms

The following are key terms/concepts relating to binary trees:

* **Node** - each element of a binary tree is called a node. A node contains a value and pointers to its children.
* **Root** - the topmost node of the tree.
* **Leaf** - a node with no children.
* **Internal node** - a node with at least one child.
* **Subtree** - any node of a tree, along with its descendants, forms a subtree.
* **Height** - the height of a tree is the length of the path from the root to the deepest node.
* **Depth** - the depth of a node is the length of the path from the root to that node.

**Minimum and Maximum Number of Nodes in a Binary Tree**

The minimum number of nodes that must exist in a binary tree with `i` rows is `i +1`. For example, if there are two rows in a tree, and the row containing the root is row 0, then there would be a minimum of 2 nodes in the tree.

The maximum number of nodes that a binary tree with `i` rows can contain is (2^0 + 2^1 + 2^2…2^i), which is also equal to 2^h+1 - 1.

If a tree has the minimum number of nodes for its number of rows, it will be of a maximum height. If a tree has the maximum number of nodes for its number of rows, it will be of a minimum height.

## Types of Binary Trees

* **Full Binary Tree** - every node has either 0 or 2 children.
* **Complete Binary Tree** - a binary tree is complete if all its levels (except possibly the last one) are completely filled, and all nodes are as far left as possible.

1

/ \

2 3

/ \ /

4 5 6

* **Perfect Binary Tree** - a binary tree is perfect if all internal nodes have exactly two children and all levels (including the leaf level) are completely filled.

1

/ \

2 3

/ \ / \

4 5 6 7

*\*All perfect binary trees are complete, but not all complete binary trees are perfect.*

* **Balanced Binary Tree** - the difference between the heights of the left and right subtrees of any node is at most one.
* **Binary Search Tree** - each node has a key (value), and in a binary search tree, the left subtree of a node contains only nodes with keys less than the node's key, and the right subtree of a node only contains nodes with keys greater than the node's key.

Both the left and right subtrees must also be binary search trees.

10

/ \

5 20

/ \ / \

3 7 15 25

## Use Cases of Binary Trees

* File System Hierarchies
* Network Data Routing
* Searching (using binary search trees)
* Database indexing

## Merkle Trees

A **Merkle tree**, also known as a **hash tree**, is a fundamental data structure in computer science used to verify data integrity and consistency. Merkle trees are particularly important in blockchain technologies like Bitcoin for efficient and secure verification of transactions.

A Merkle tree is a binary tree in which each **leaf** node represents a hash of a block of data, and each **non-leaf** node represents the hash of its two child nodes. The structure ensures that the data integrity can be checked efficiently.

**Structure of a Merkle Tree**

1. **Leaves**: the leaf nodes contain the hash of individual data blocks.
2. **Non-leaf nodes**: each non-leaf node is a hash of its two child nodes.
3. **Root**: the topmost node, known as the Merkle root, is the hash of all the nodes beneath it, effectively summarising all the data.

In the below diagram, `D`, `E`, `F` and `G` are the hash values of data blocks. `A` is the hash of `D` and `E`, and `B` is the hash of `F` and `G`. Finally, the root is the hash of `A` and `B`.

