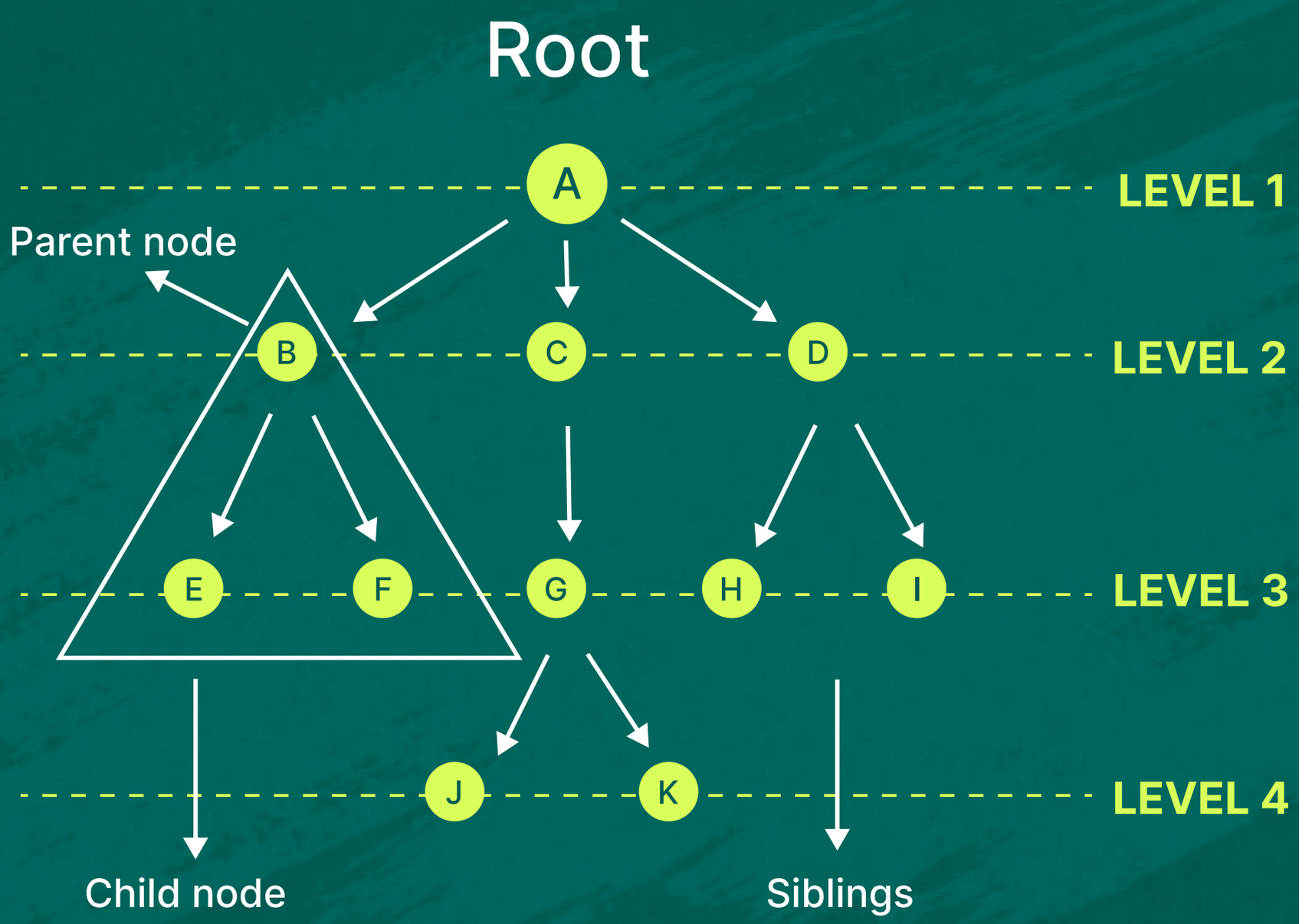
BOSSCODER ACADEMY

Complete Guide to

TREES



Most Commonly Asked Questions

What is TREES

- Trees are a fundamental data structure used in many applications such as compilers, parsers, file system organisation, decision logic
- Understanding trees is a crucial skill required for many coding interviews
- Here is a well-defined roadmap that can help you master trees from scratch to interview ready

Basics of trees

Trees are non linear, hierarchical structures where elements are arranged in multiple levels.

Terminologies

Root Node: Node which doesn't have a parent node and from which the entire tree originates.

Parent Node: An immediate predecessor of a given node is called its parent node.

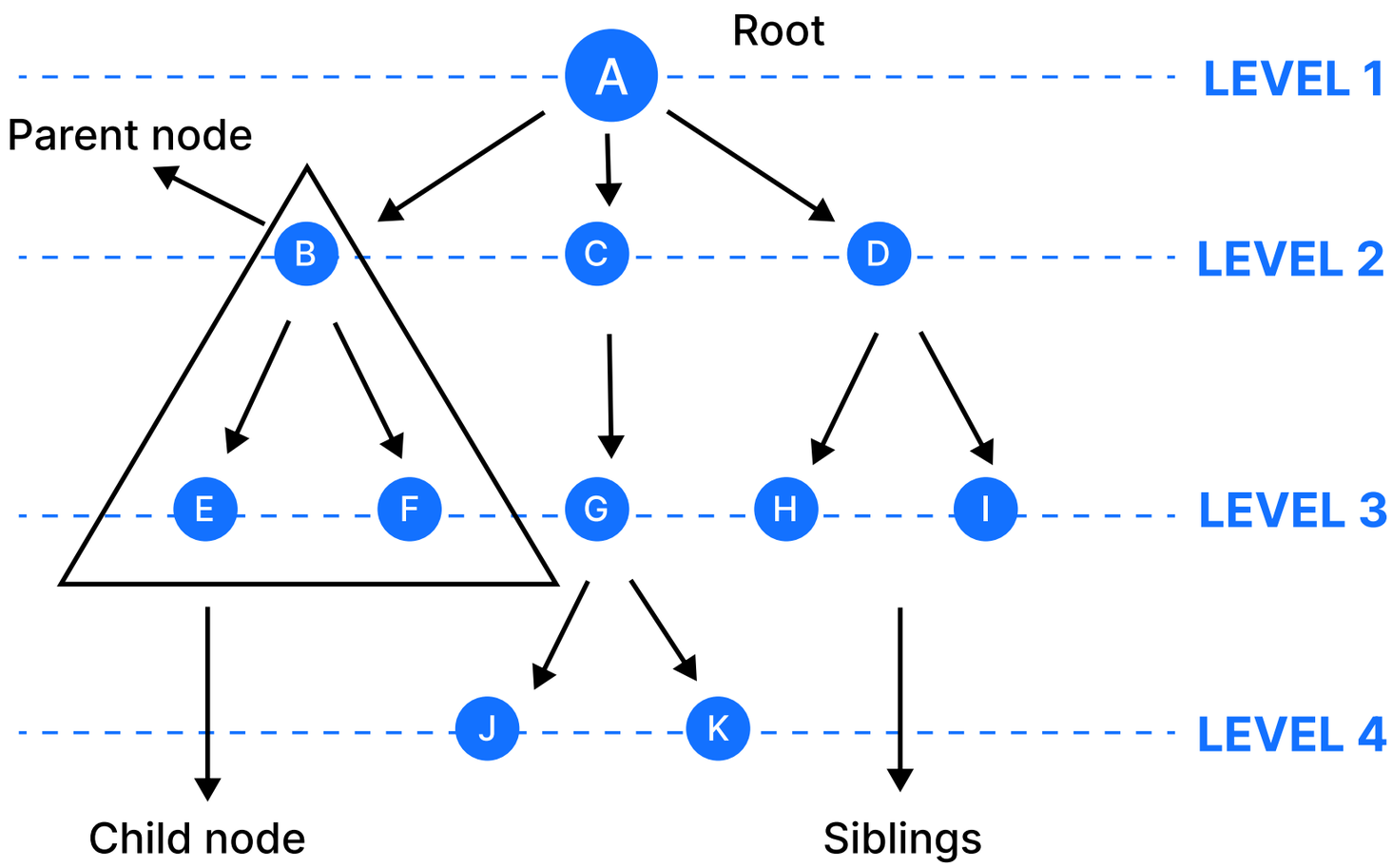
Child Node: All the immediate successors of a node are called its child node.

Leaf Node: A Node which does not have any child is a leaf or boundary node

Internal Node: Any parent node that has at least one child node

Siblings: A group of nodes that share the same parent node are known as siblings.

Degree of Node: The number of child nodes a given node has



Properties of Trees

- If a tree has n nodes, it will always have one less number of edges (n-1).
- Trees are recursive data structures because a tree is usually composed of smaller trees
- The depth of a node is the length between the root and that node or number of nodes from the root to node.
- If we consider level of root node to be zero, then depth of node is equal to its level+1

Tree Representation in Programming

Example Node structure in C++

```
struct Node {
 int value;
 Node* left;
 Node* right;
};
```

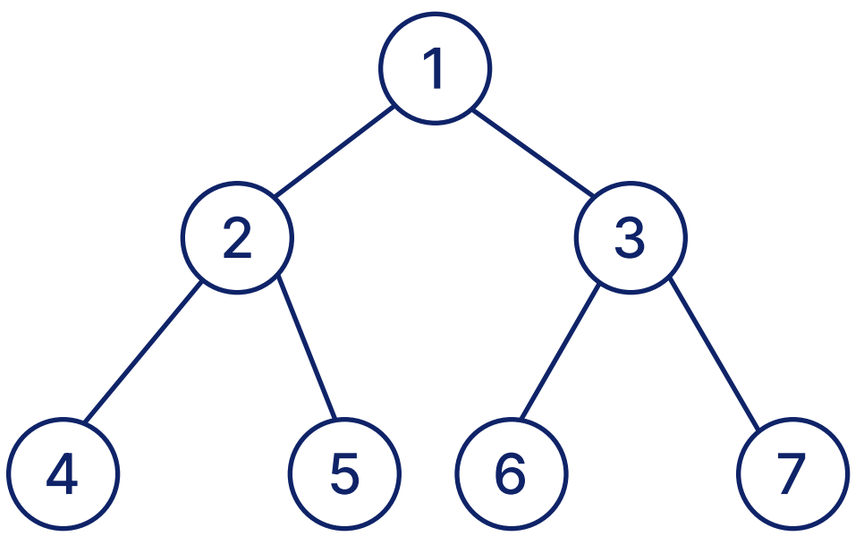
The left child pointer points to the left subtree, and the right child pointer points to the right subtree. If a node does not have any children, its corresponding child pointers are set to NULL.

Tree Traversals

Tree traversal is the process of visiting every node in a tree data structure exactly once.

The three most common types of tree traversals are:

1. **In-order traversal:** In this traversal, the left subtree is visited first, followed by the current node, and then the right subtree. This traversal is commonly used for binary search trees, where the values are sorted in ascending order.
2. **Pre-order traversal:** In this traversal, the current node is visited first, followed by the left subtree, and then the right subtree. This traversal is commonly used for creating a copy of the tree.
3. **Post-order traversal:** In this traversal, the left subtree is visited first, followed by the right subtree, and then the current node. This traversal is commonly used for deleting a tree.



Preorder	[1,2,4,5,3,6,7]
Inorder	[4,2,5,1,6,3,7]
Postorder	[4,5,2,6,7,3,1]

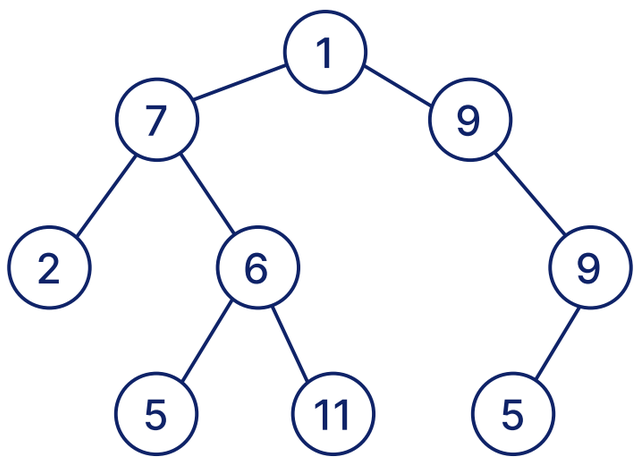
Code for Traversals

```
// In-order traversal
void inOrderTraversal(Node* node) {
    if (node == null)
return;
    inOrderTraversal(node→left);
    cout << node→value << " ";
    inOrderTraversal(node→right);
}
// Pre-order traversal
void preOrderTraversal(Node* node) {
    if (node == null)
return;
    cout << node→value << " ";
    preOrderTraversal(node→left);
    preOrderTraversal(node→right);
}
// Post-order traversal
void postOrderTraversal(Node* node) {
    if (node == null)
return;
    postOrderTraversal(node→left);
    postOrderTraversal(node→right);
    cout << node→value << " ";
}
```

Types of trees

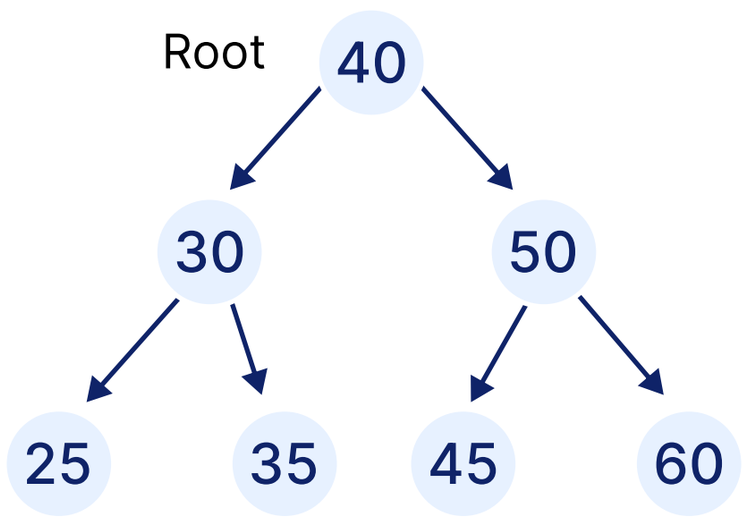
1. Binary Tree:

Tree in which each node has at most two children, known as the left child and the right child. This type of tree is commonly used for searching and sorting algorithms, such as binary search.



2. Binary Search Tree (BST):

Binary tree in which the left child of a node contains a value that is smaller than the value of the node, and the right child of a node contains a value that is greater than the value of the node. This property allows for efficient searching and sorting algorithms.



Types of trees

1. B-Tree:

A B-tree is a tree-based data structure that automatically maintains sorted data and provides efficient operations for searches, sequential access, insertions, and deletions in logarithmic time complexity.

2. AVL Tree:

A self-balancing binary search tree in which the heights of two child subtrees of any node differ by at most one. This balancing property ensures that the tree remains balanced, which allows for efficient searching and insertion operations.

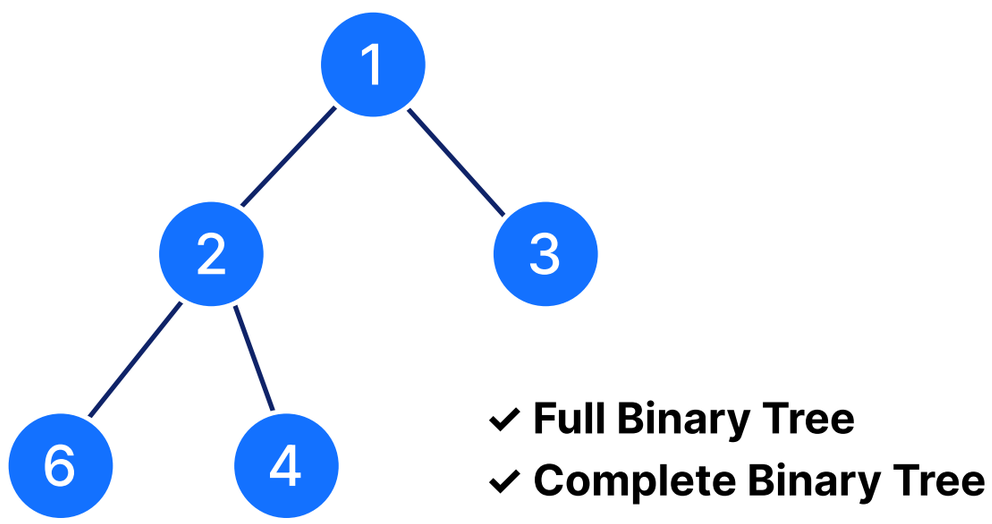
Types of Binary Trees

1. Full Binary Tree:

Each internal node has either zero or two children nodes but not only one child node.

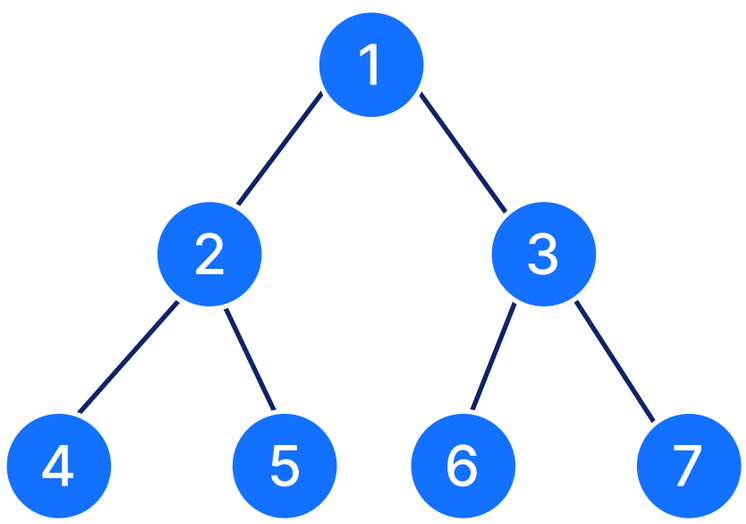
2. Complete Binary Tree:

If all the levels of a tree, except possibly the last level, have the maximum number of possible nodes, and all the nodes in the last level appear as far left as possible.



3. Perfect Binary Trees:

All the internal nodes have exactly two children nodes and all leaf nodes at same level.



Most Important Questions on Trees for Interviews

- https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/
- https://leetcode.com/problems/kth-smallest-element-in-a-bst/
- https://leetcode.com/problems/invert-binary-tree/
- https://leetcode.com/problems/maximum-depth-of-binary-tree/
- https://leetcode.com/problems/binary-tree-level-order-traversal/
- https://leetcode.com/problems/sum-root-to-leaf-numbers/
- https://leetcode.com/problems/subtree-of-another-tree/
- https://leetcode.com/problems/implement-trie-prefix-tree/
- https://leetcode.com/problems/same-tree/
- https://leetcode.com/problems/binary-tree-right-side-view/

BOSSCODER ACADEMY

WHY BOSSCODER?

1000+ Alumni placed at Top Product-based companies.

More than **136% hike** for every **2 out of 3** working professional.

Average package of **24LPA.**

The syllabus is most **up-to-date** and the list of problems provided covers **all important topics.**

Lavanya
Meta

Course is very well **structured** and **streamlined** to crack any **MAANG** company

Rahul .
Google

EXPLORE MORE