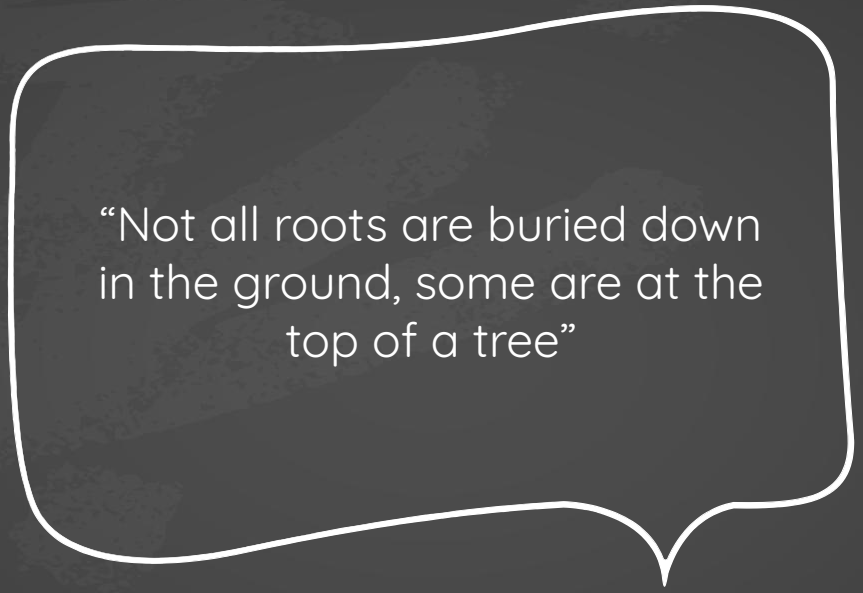
# Trees

A data structure with root & leaves....




–Someone Famous

# Harini Rajendran

Software Engineer, Confluent

https://www.linkedin.com/in/harinirajendrancmu

## 01
## Intro

What, where & why ?

## 02
## Types

There are so many of them...

## 03
## Traversals

Let's walk the trees...

## 04
## Code examples

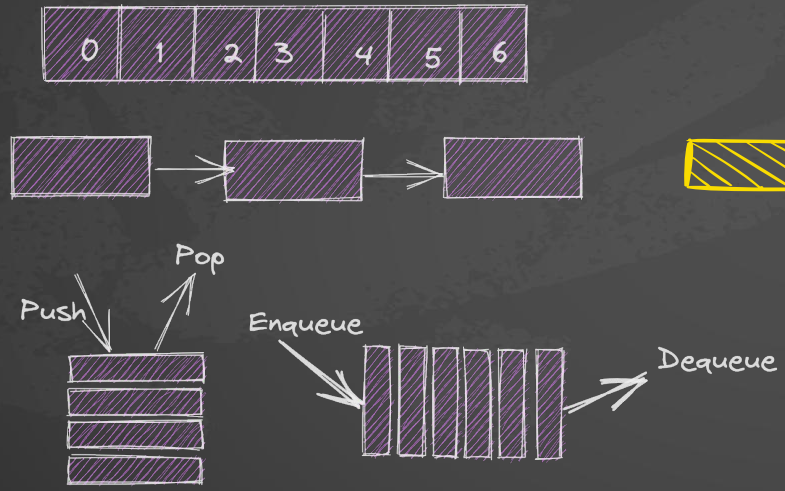
for interviews...

# 01 Intro

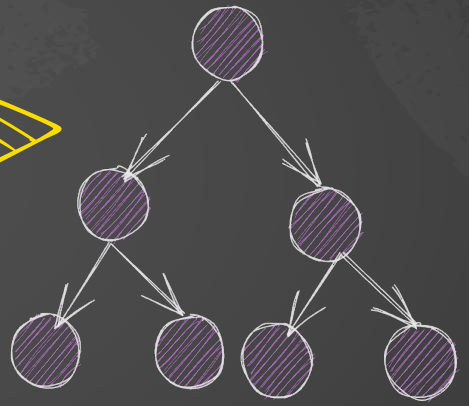
What, where & Why?

# What's a Tree?

## Linear




## Non-linear

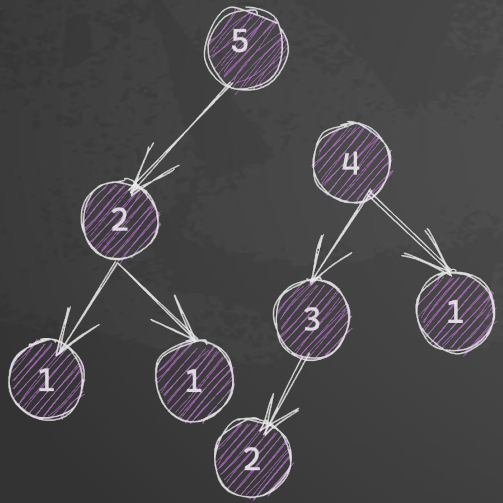


Arrays - Random access, fixed size
Linked List - Dynamic, O(1) add & delete, O(n) search
Stacks & Queues - Special Arrays/LL

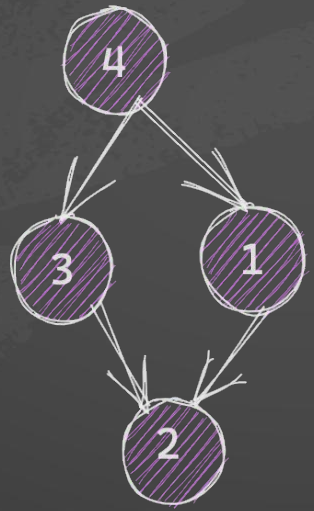
Trees - Dynamic, Nodes connected by directed edges, hierarchical relation, recursive data structure

# What's NOT a Tree?

Disjoint roots

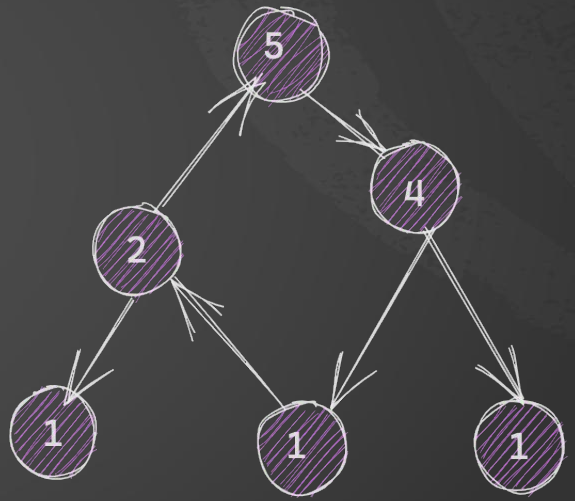



Node has 2 parents




Has Loop


Has Cycle



# Example 1

Linux File System

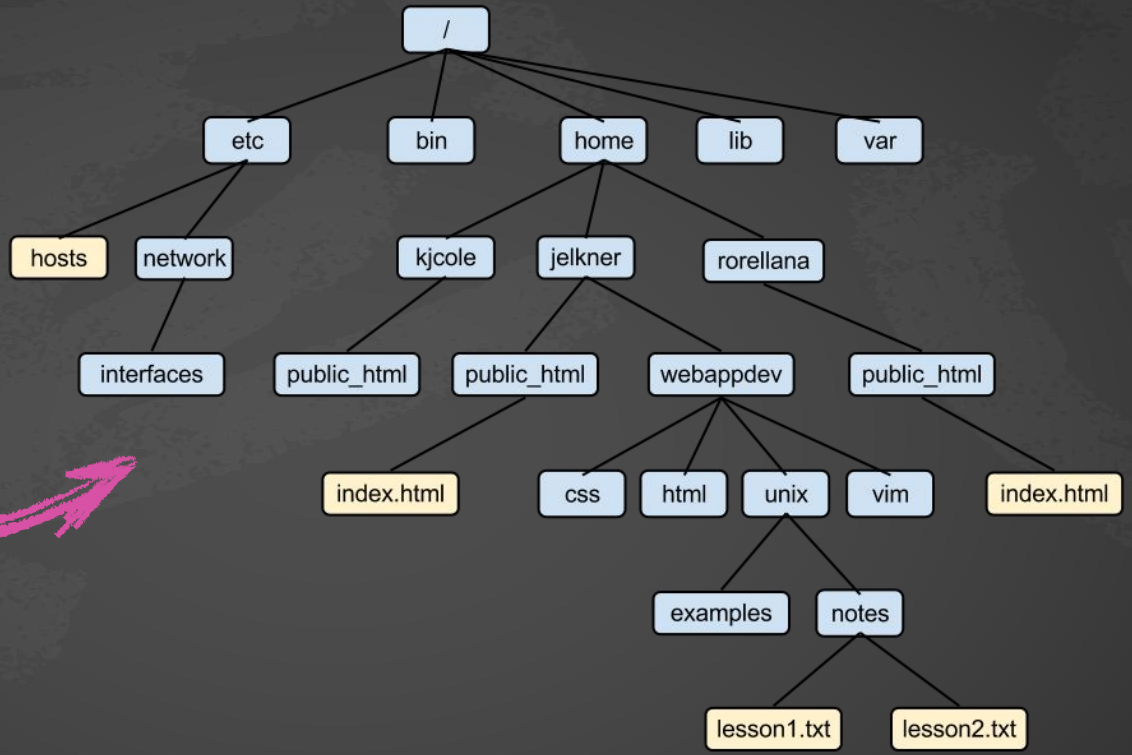





Image Credit

# Example 2

Office Org Chart

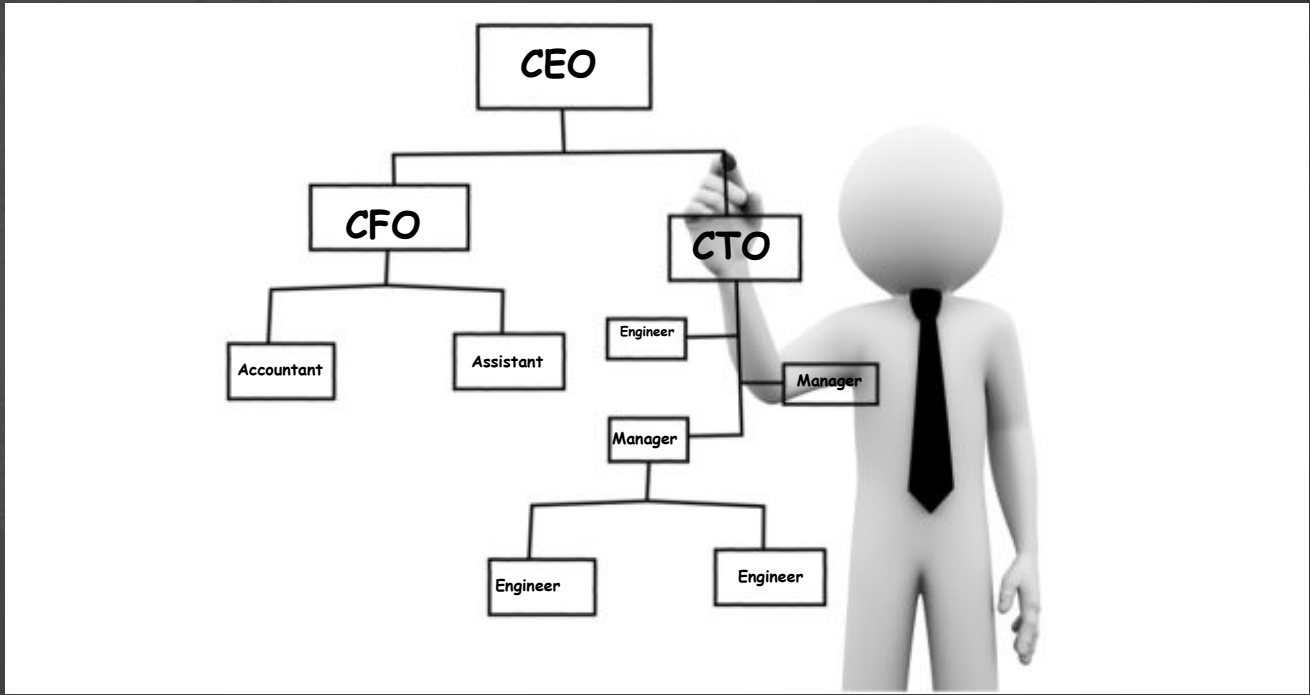





Image Credit

# The Lingo

Node - Fundamental unit of a tree

Edge - Links connecting nodes

Root - First node

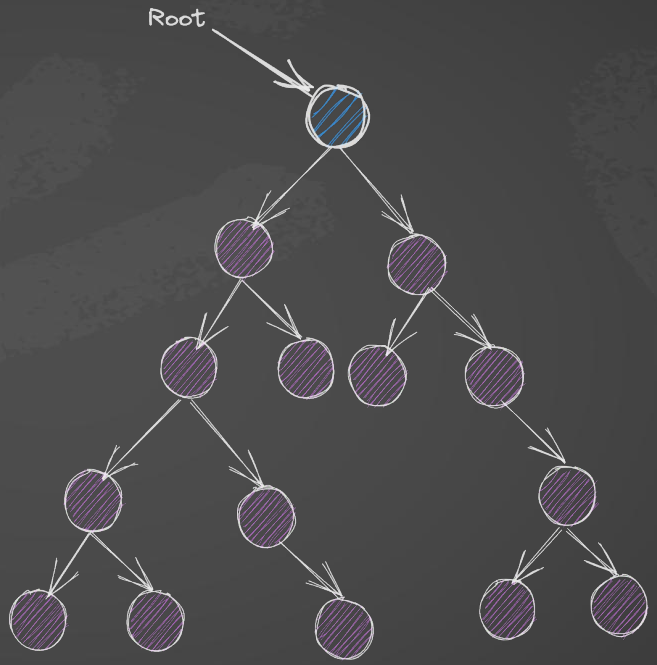
Leaf - Node with NO children

Internal nodes - All non-leaf nodes

Parent - Predecessor of a node

Children - Immediate descendent of a node

Siblings - Nodes with the same parent



# The Lingo

Node - Fundamental unit of a tree

Edge - Links connecting nodes

Root - First node

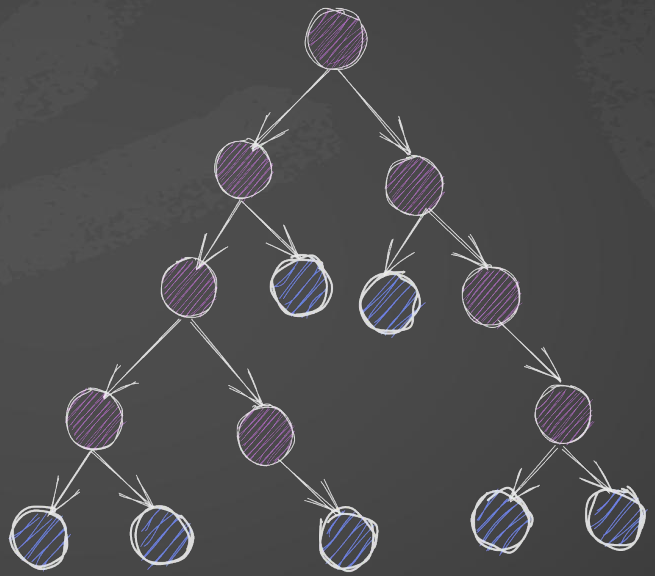
Leaf - Node with NO children

Internal nodes - All non-leaf nodes

Parent - Predecessor of a node

Children - Immediate descendent of a node

Siblings - Nodes with the same parent



# The Lingo

Node - Fundamental unit of a tree

Edge - Links connecting nodes

Root - First node

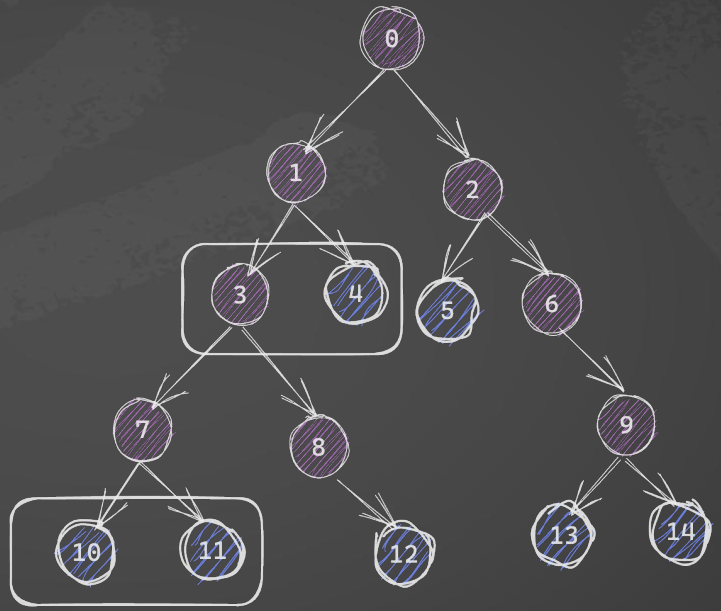
Leaf - Node with NO children

Internal nodes - All non-leaf nodes

Parent - Predecessor of a node

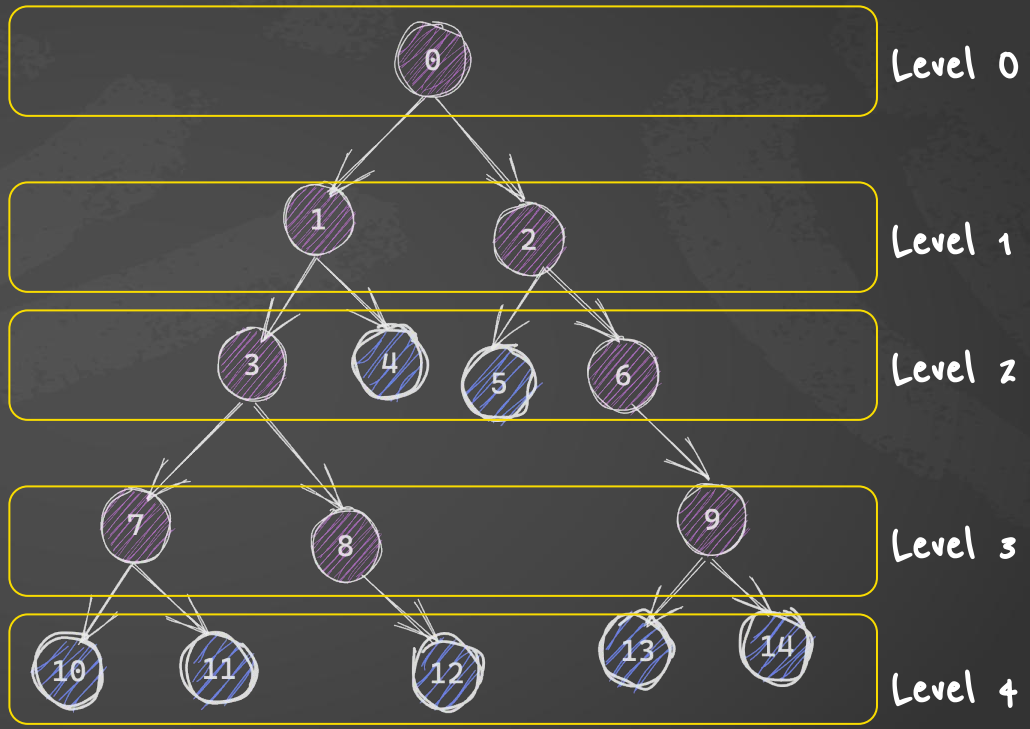
Children - Immediate descendent of a node

Siblings - Nodes with the same parent



# The Lingo

- **Level** - A generation of nodes are in same level
- **Depth** - # edges from root to a node
- **Path** - Sequence of nodes & edges b/w 2 nodes
- **Height** - Length of longest path in the tree
- **Ancestor** - Any predecessor node in the path from root
- **Subtree** - Each node & its descendants form a tree



# The Lingo

**Level** - A generation of nodes are in same level

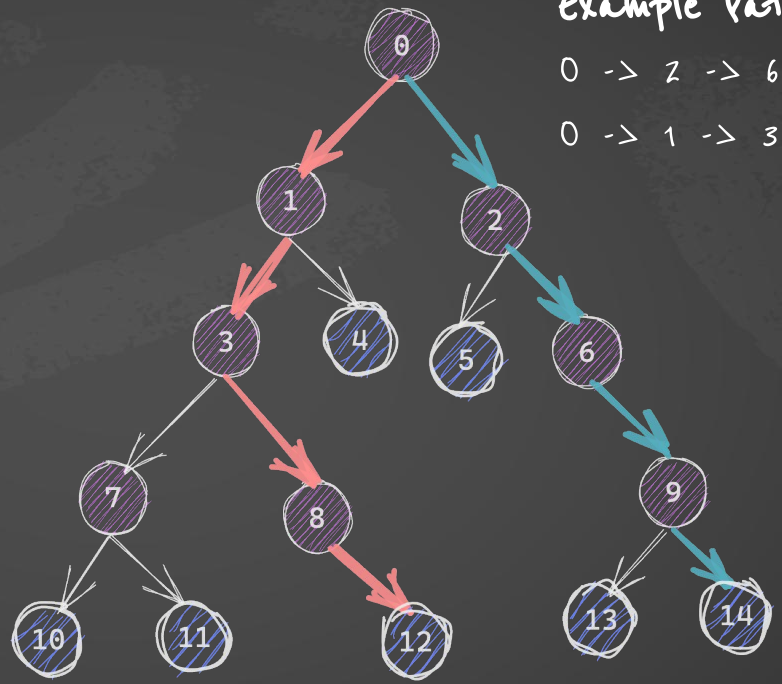
**Depth** - # edges from root to a node

**Path** - Sequence of nodes & edges b/w 2 nodes

**Height** - Length of longest path in the tree

**Ancestor** - Any predecessor node in the path from root

**Subtree** - Each node & its descendants form a tree




**Example Paths**

0 -> 2 -> 6 -> 9 -> 14

0 -> 1 -> 3 -> 8 -> 12

# The Lingo

**Level** - A generation of nodes are in same level

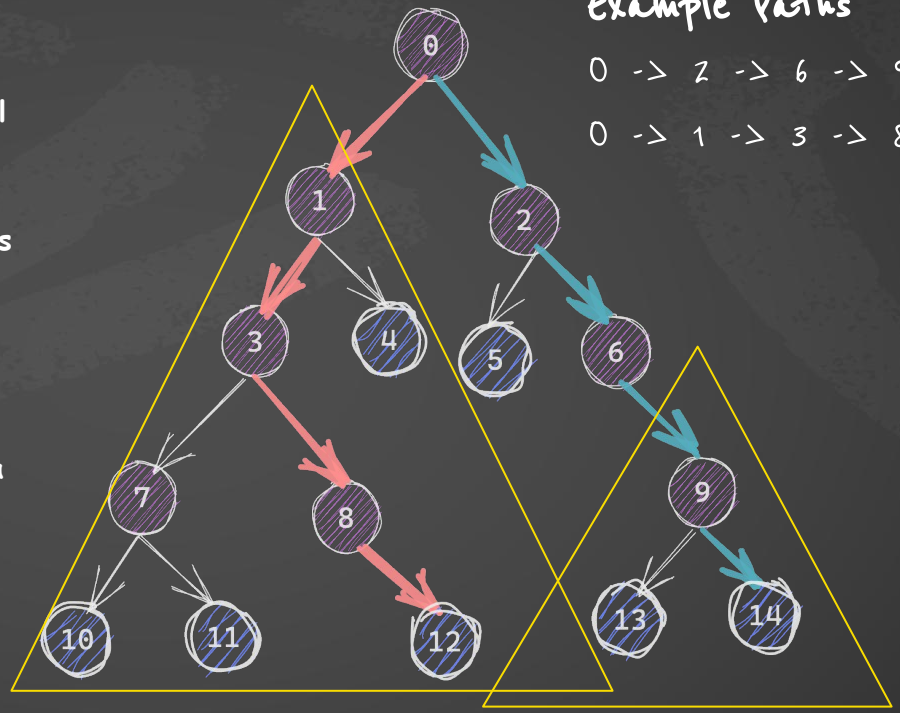
**Depth** - # edges from root to a node

**Path** - Sequence of nodes & edges b/w 2 nodes

**Height** - Length of longest path in the tree

**Ancestor** - Any predecessor node in the path from root

**Subtree** - Each node & its descendants form a tree



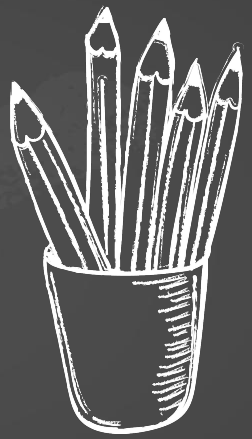

**Example Paths**

0 -> 2 -> 6 -> 9 -> 14

0 -> 1 -> 3 -> 8 -> 12

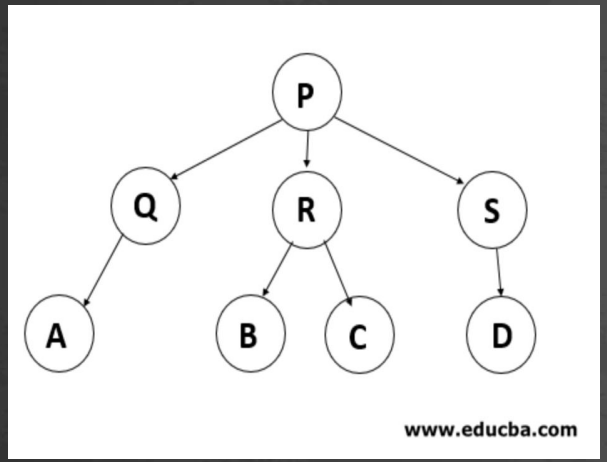
# 02 Types

There are so many of them...



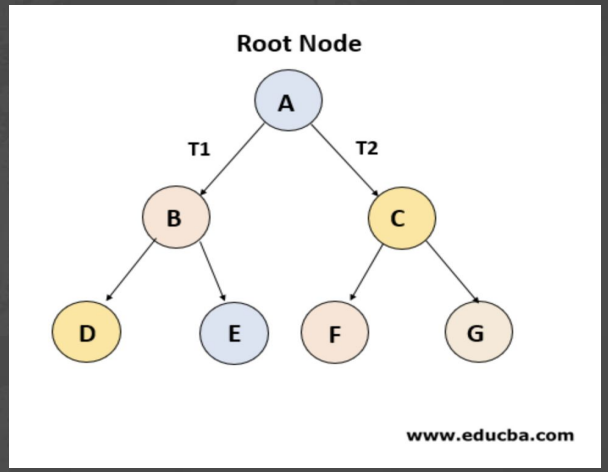
# Types of Trees

## General Tree



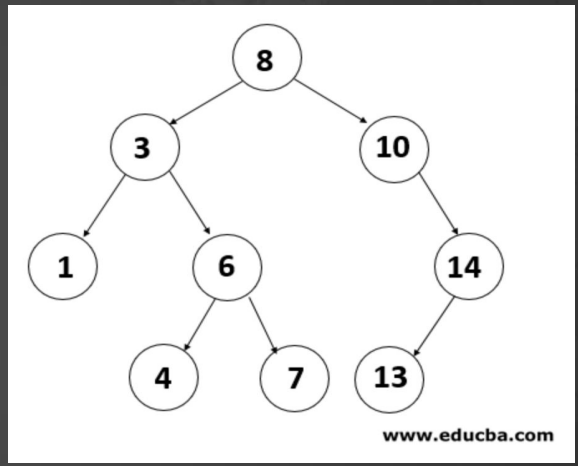

No constraints
A node can have any # of children

## Binary Tree




A node can have 2 children at max
They are called Left and Right child

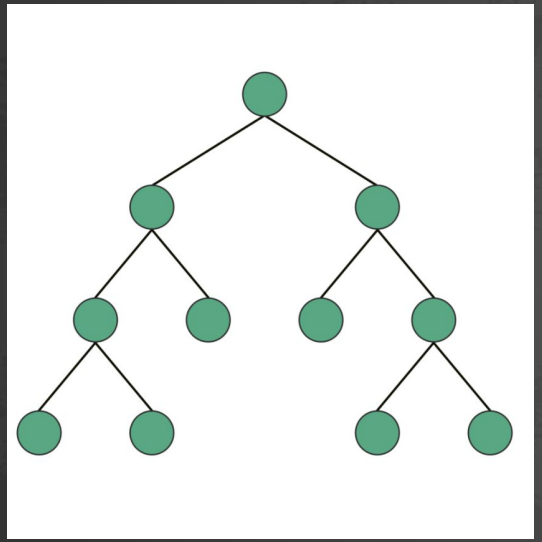
## Binary Search Tree




Special binary tree
A node can have 2 children at max
Value of Left child <= Value of Parent
Value of Right child >= Value of Parent

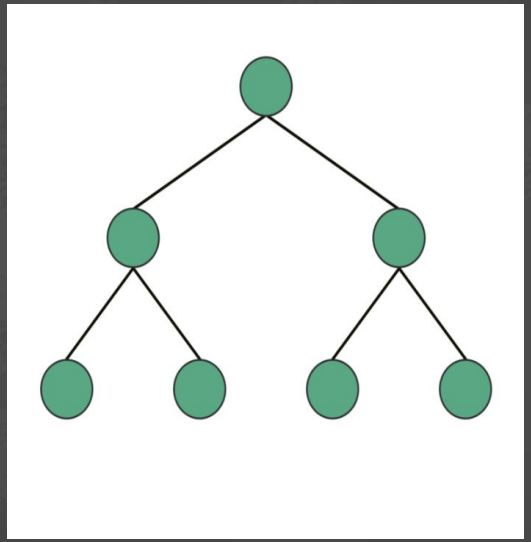
# Types of Binary Trees

## Full Binary Tree



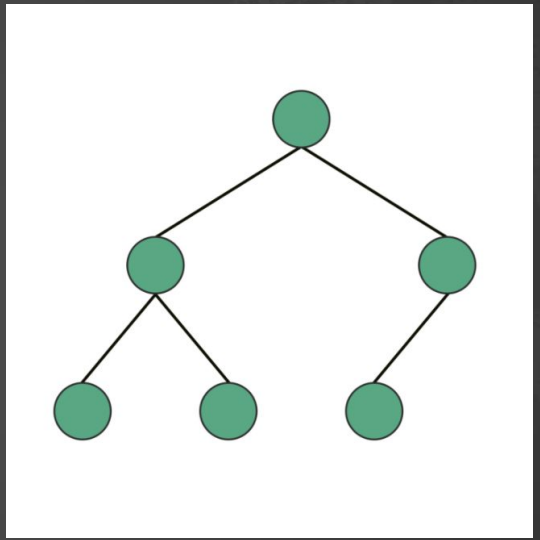
Every node has exactly 0 or 2 children

## Perfect Binary Tree



All internal nodes have 2 children and all leaves are at the same level.

## Complete Binary Tree

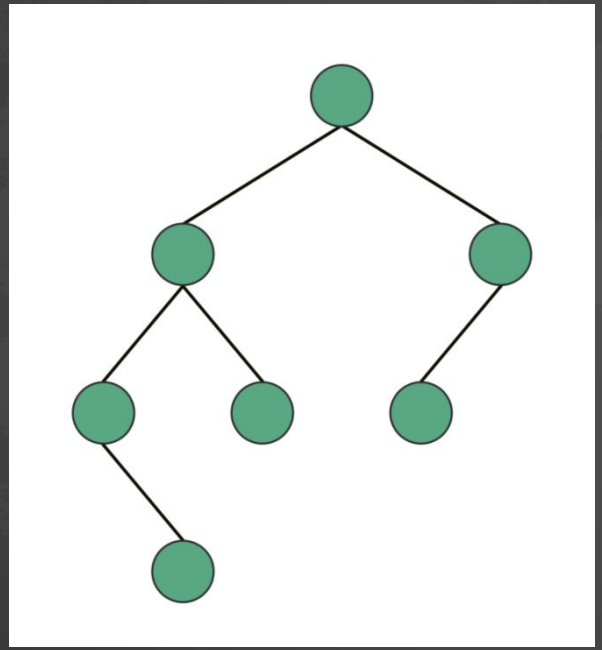


All levels completely filled with nodes except the last level and in the last level, all the nodes are as left side as possible.

Image Credit

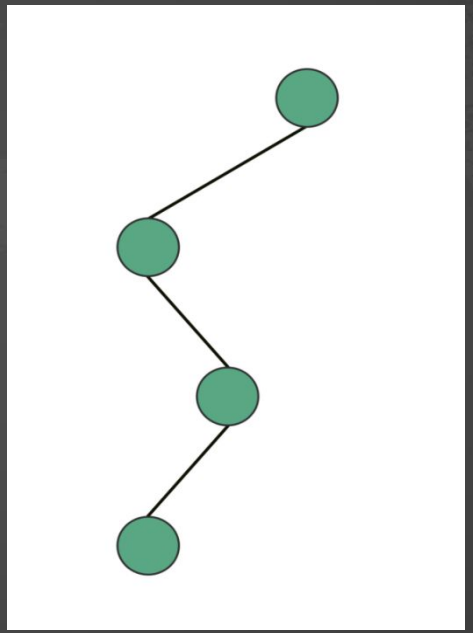
# Types of Binary Trees

## Balanced Binary Tree



Height of the left and the right sub-trees of every node may differ by at most 1.

## Degenerate Binary Tree



Every parent node has only one child node



Image Credit

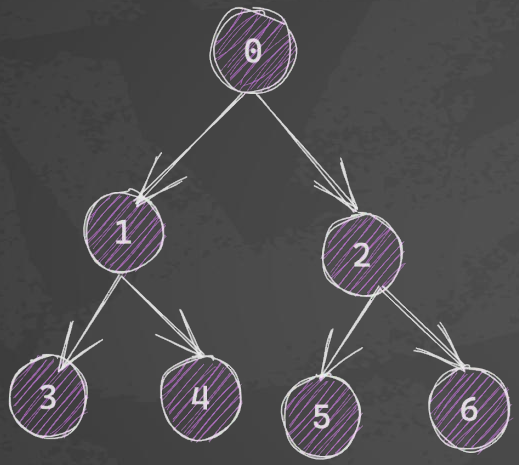
# Some advanced trees

- AVL Trees
- B-Trees
- B+ Trees
- Splay Trees
- Red Black Trees
- Treap

# 03 Traversal

Let's walk the trees...

# Tree Traversals




➔ Pre-Order

Root -> Left -> Right

0 - 1 - 3 - 4 - 2 - 5 - 6

➔ In-Order (Most commonly used)

Left -> Root -> Right

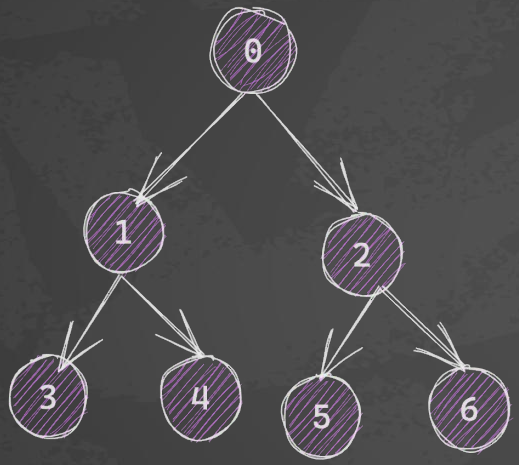
3 - 1 - 4 - 0 - 5 - 2 - 6

➔ Post-Order

Left -> Right -> Root

3 - 4 - 1 - 5 - 6 - 2 - 0

# Tree Traversals




➔ BFS

One level at a time

0 - 1 - 2 - 3 - 4 - 5 - 6

➔ DFS (Same as pre-order)

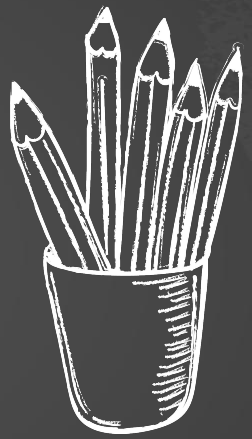
Go till the end of the path and then backtrack

0 - 1 - 3 - 4 - 2 - 5 - 6

# ⇨ 04

# Code Examples

For interviews...



# A Tree Node

```
class Node {
    int value;
    Node left;
    Node right;

    Node(int value) {
        this.value = value;
        right = null;
        left = null;
    }
}
```

```
class BinaryTree {
    Node root;

    // ...
}
```

# Traversals - Using Recursion

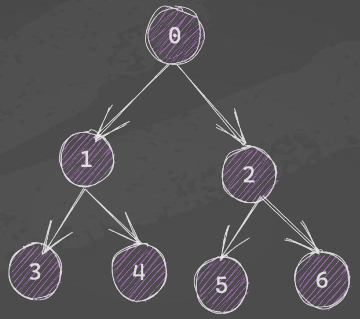
```
public void inOrder(Node node) {
    if (node != null) {
        inOrder(node.left);
        System.out.print(" " + node.value);
        inOrder(node.right);
    }
}
```

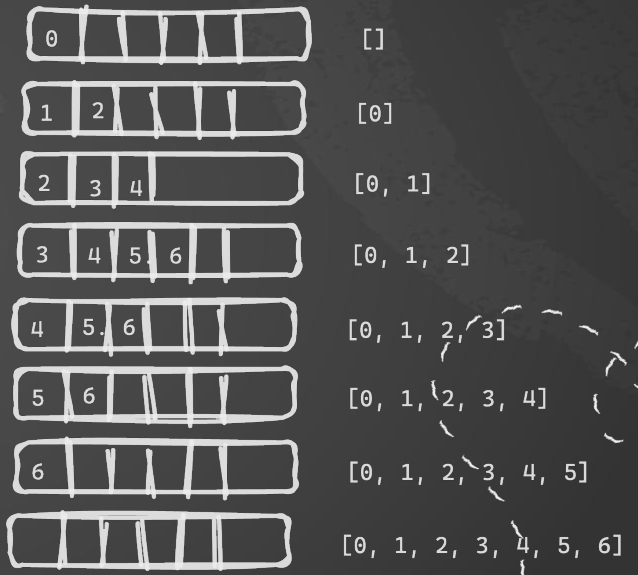
```
public void preOrder(Node node) {
    if (node != null) {
        System.out.print(" " + node.value);
        preOrder(node.left);
        preOrder(node.right);
    }
}
```

```
public void postOrder(Node node) {
    if (node != null) {
        postOrder(node.left);
        postOrder(node.right);
        System.out.print(" " + node.value);
    }
}
```

# Level Order Traversal - Use Queue

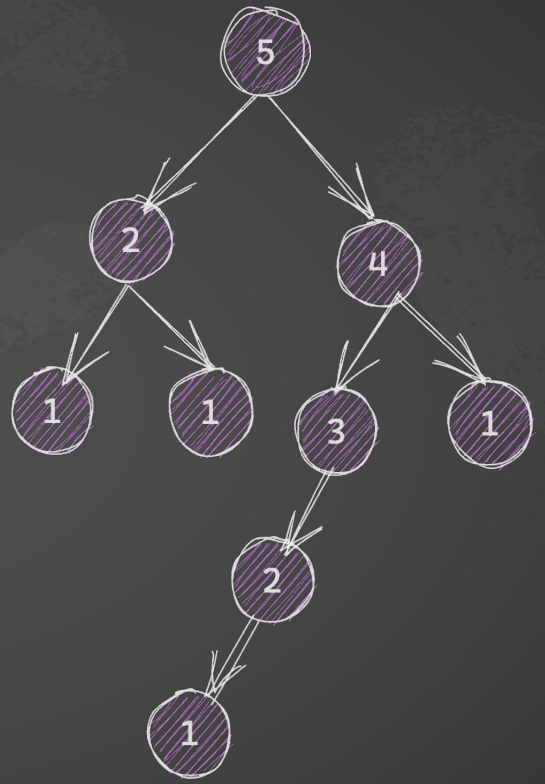
```
public void levelOrder(Node root) {
    if (root == null) {
        return;
    }

    Queue<Node> nodes = new LinkedList<>();
    nodes.add(root);

    while (!nodes.isEmpty()) {

        Node node = nodes.remove();

        System.out.print(" " + node.value);

        if (node.left != null) {
            nodes.add(node.left);
        }

        if (node.right != null) {
            nodes.add(node.right);
        }
    }
}
```



# Height of a Tree

```
public int height(Node root)
{
    if (root == null) {
        return 0;
    }

    int LeftSubtreeHt = height(root.left);
    int RightSubtreeHt = height(root.left);
    return 1 + Math.max(LeftSubtreeHt, RightSubtreeHt);
}
```



Smart data structures and dumb code works a lot better than the other way around.

-Eric S. Raymond

# Thanks!

**WWCode Slack Handle: Harini Rajendran**

https://www.linkedin.com/in/harinirajendrancmu

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**.