Google Developer Student Clubs

Trees

and an intro to recursion

Dean Shin
https://www.linkedin.com/in/dean-shin/

Recursion

Introduction

- Recursion is when a function calls itself
- Every recursive function has two parts:
  - the base case
  - the recursive call (also known as the inductive clause)
    - The recursive call should **‘shrink’** the size of the problem.
- Understanding recursion is applicable not only to competitive coding, but also software engineering in general

Recursion

Factorial

- How do we define n! in a recursive way?
- n! is 1 * 2 * ... * (n - 1) * n
- (n - 1)! is 1 * 2 * ... * (n - 1)
- Therefore, n! = n * (n - 1)!
- However, we still need a base case!

Recursion

`Factorial`

- Base case: 0! = 1
- Recursive call: n! = n * (n - 1)!
- 3! = ?
- 5! = ?
- (-1)! = ?

Recursion

- Base case: 0! = 1
- Recursive call: n! = n * (n - 1)!

Factorial Code

```
public int factorial(int n) {
    if(n == 0) {
        return 1;
    }
    return n * factorial(n - 1);
}
```


Recursion

`Fibonacci`

- How do we define the fibonacci sequence in a recursive way?
- The Fibonacci numbers, commonly denoted F(n) form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1.
- F(n) = F(n - 1) + F(n - 2)
- F(0) = 0, F(1) = 1

Recursion

- Base case: F(0) = 0, F(1) = 1
- Recursive call: F(n) = F(n - 1) + F(n - 2)

Factorial Code

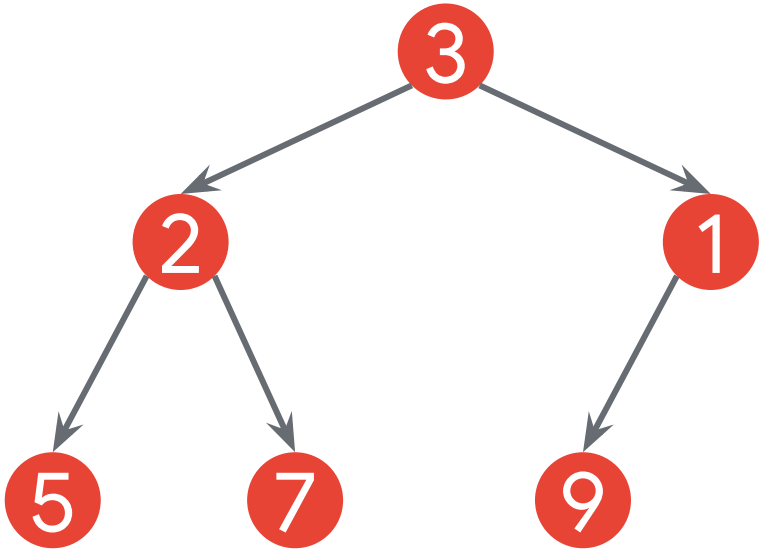
```
public int fib(int n) {
    if(n == 0 || n == 1) return n;
    return fib(n - 1) + fib(n - 2);
}
```


Trees

Introduction

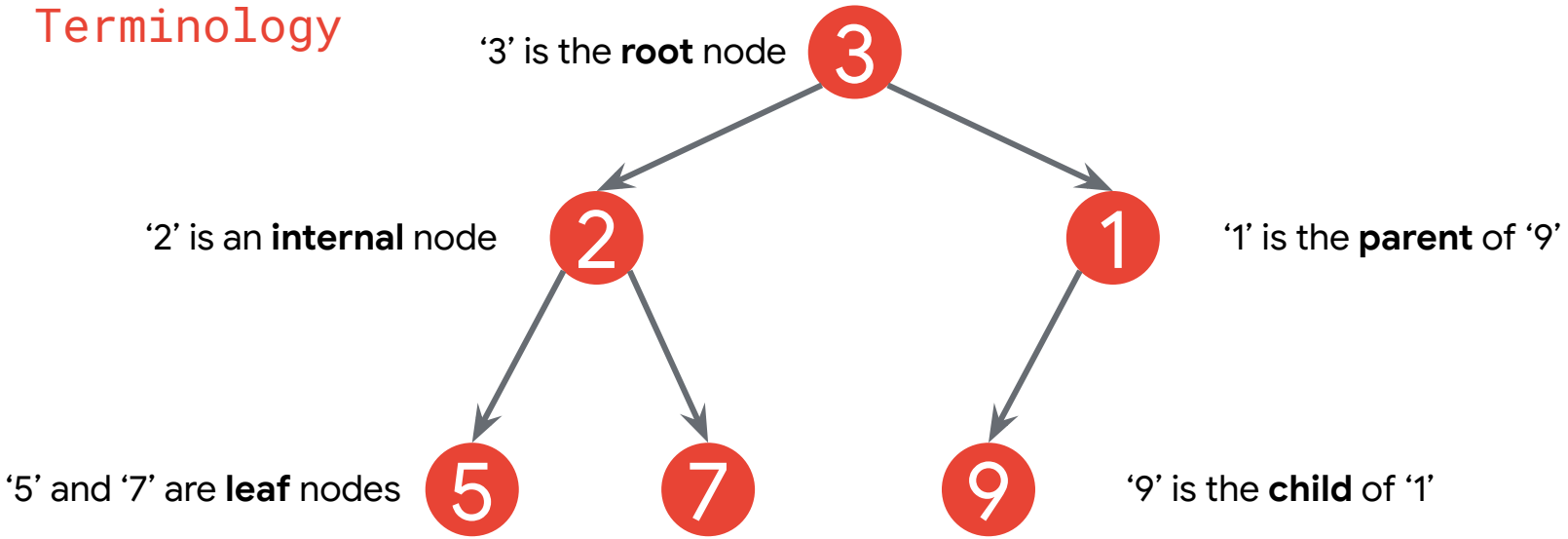
- **Trees** are a data structure similar to linked lists, where data is stored in nodes.
- Unlike linked lists, however, **a tree may have multiple children**.
- Usually, every tree node has **up to 2 children**. This is called a **binary tree.**
- A tree cannot have a loop. That would be a data structure called a graph.

Trees



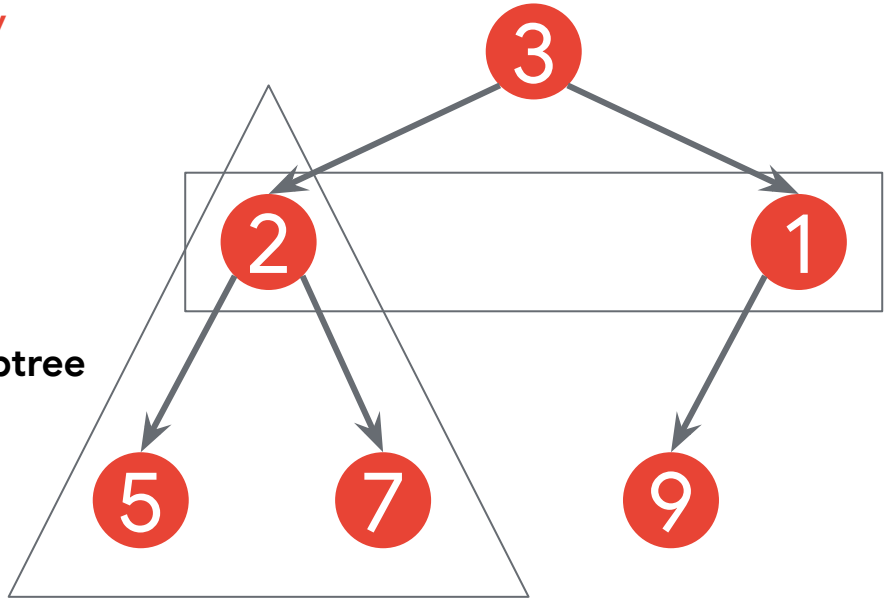
Trees

Terminology



Trees

Terminology



this is a **level**

this is a **subtree**

Trees

Code

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    public TreeNode(int val, TreeNode left, TreeNode right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }
}
```


Trees

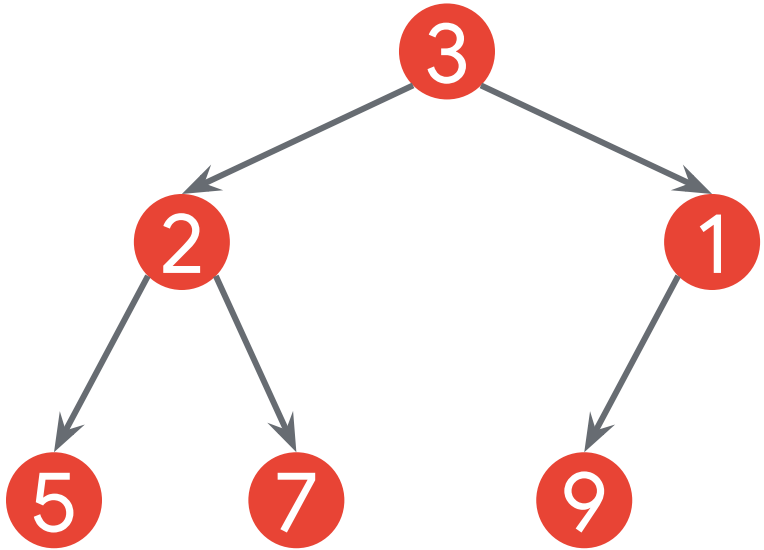
Methodology

- Leverage **recursion** to solve the problem
- The **base case** is usually when the node is **null**.
- The **recursive function** returns the answer for a ***subtree***.
- To calculate the answer for a node, **first** get the child **subtrees**' answers, then combine them with the current node's data to get the answer.

Trees

Tree Sum

Given a binary tree `root`, return the sum of all of the nodes in the tree.



Trees

Tree Sum Code

```
public int sumTree(TreeNode root) {
    if(root == null) {
        return 0;
    }
    int leftSubtreeSum = sumTree(root.left);
    int rightSubtreeSum = sumTree(root.right);
    return leftSubtreeSum + rightSubtreeSum + root.val;
}
```


Trees

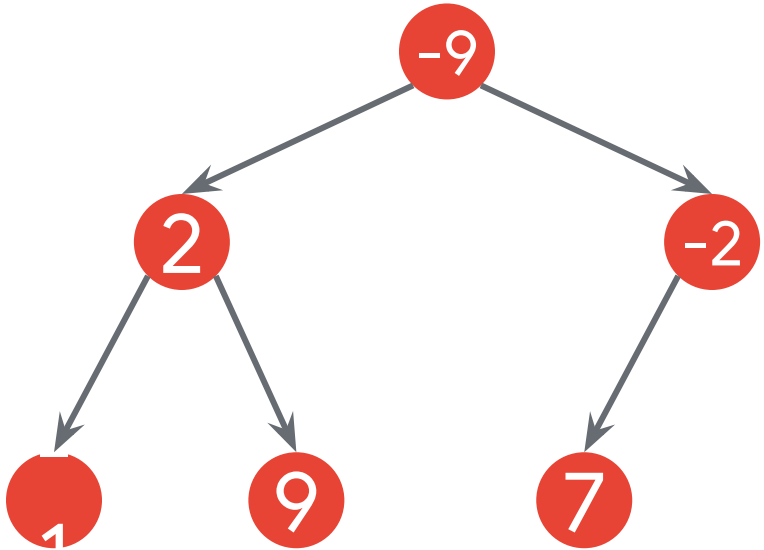
Methodology, part 2.

- Oftentimes, you will need a **helper function** with a different return type
- This is because you may need to return multiple pieces of information to calculate the answer for a node.
- Depending on your language, this process is easy (Python, JavaScript) or annoying and hard (C++, Java)

Trees

Subtree with Maximum Value

Given a binary tree `root`, return the maximum sum of a subtree. A subtree can be `null` in which case its sum is 0.



Trees

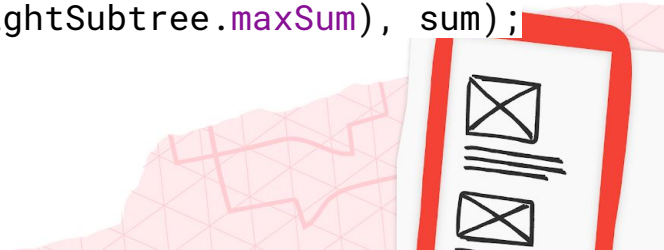
Subtree with Maximum Value Code Pt. 1

```
class SumAndMaxSum {
    public int sum;
    public int maxSum;

    public SumAndMaxSum(int sum, int maxSum) {
        this.sum = sum;
        this.maxSum = maxSum;
    }
}
```


Trees

Subtree with Maximum Value Code Pt. 2

```
public int subtreeWithMaximumValue(TreeNode root) {
    SumAndMaxSum sumAndMaxSum = helper(root);
    return sumAndMaxSum.maxSum;
}

private SumAndMaxSum helper(TreeNode root) {
    if(root == null) {
        return new SumAndMaxSum(0, 0);
    }
    SumAndMaxSum leftSubtree = helper(root.left);
    SumAndMaxSum rightSubtree = helper(root.right);
    int sum = leftSubtree.sum + rightSubtree.sum + root.val;
    int maxSum = Math.max(Math.max(leftSubtree.maxSum, rightSubtree.maxSum), sum);
    return new SumAndMaxSum(sum, maxSum);
}
```

Problems

- 509. Fibonacci Number
- 129. Sum Root to Leaf Numbers
- 112. Path Sum <- this uses a slightly different technique where you pass data INTO the recursive function (see whiteboard on left)
- 559. Maximum Depth of N-ary Tree
- 508. Most Frequent Subtree Sum