LeetCamp

A project dedicated to DS&A

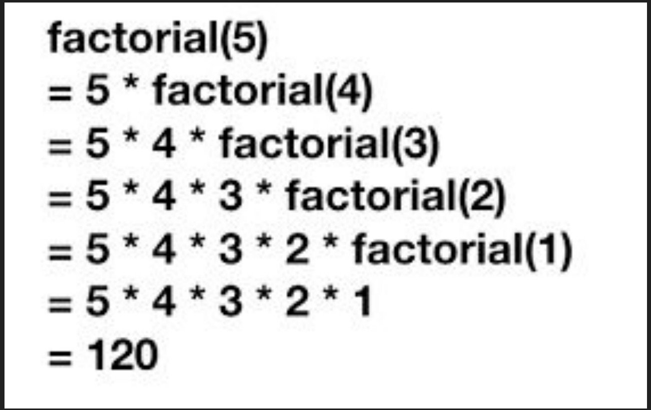
Agenda

1. Recursion Review
2. Binary Trees Review
3. DFS Fundamentals
4. LeetCode Problems

Recursion

Recursion Overview

- When a function calls itself
- In a way… you can kind of think of it as a loop
  - Instead you’re using the function itself
- Recursion works because your function argument changes throughout each call (aka “iteration”)



```
factorial(5)
= 5 * factorial(4)
= 5 * 4 * factorial(3)
= 5 * 4 * 3 * factorial(2)
= 5 * 4 * 3 * 2 * factorial(1)
= 5 * 4 * 3 * 2 * 1
= 120
```

Notice up there ^ how the argument is reduce by 1 each time… that’s our iterative procedure!

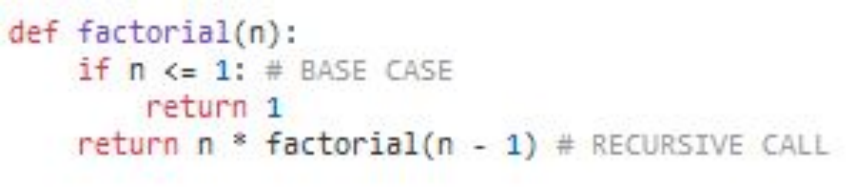
Recursion

So what do we need in a recursive function?

1. A base case!!!!!!!!!
    a. We ALWAYS need a base case (can you guess why?)
2. Our recursive call
    a. We need to call our function again

Recursion Example - Factorial

- Remember that picture from two slides ago?
  - It was to calculate the factorial
- Below is the code



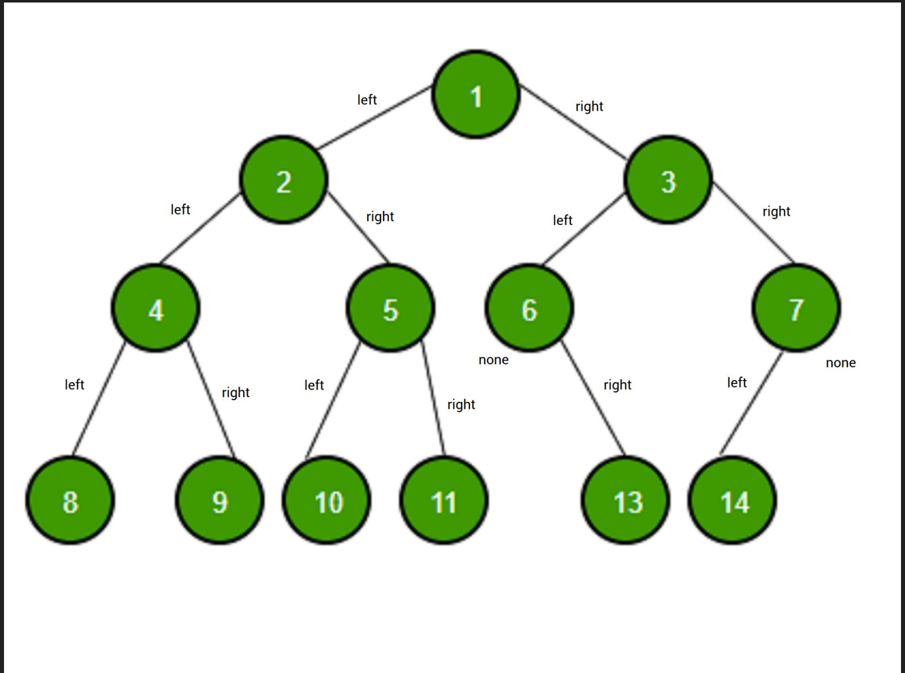
```python
def factorial(n):
    if n <= 1: # BASE CASE
        return 1
    return n * factorial(n - 1) # RECURSIVE CALL
```

← Look how simple it looks!

Binary Trees

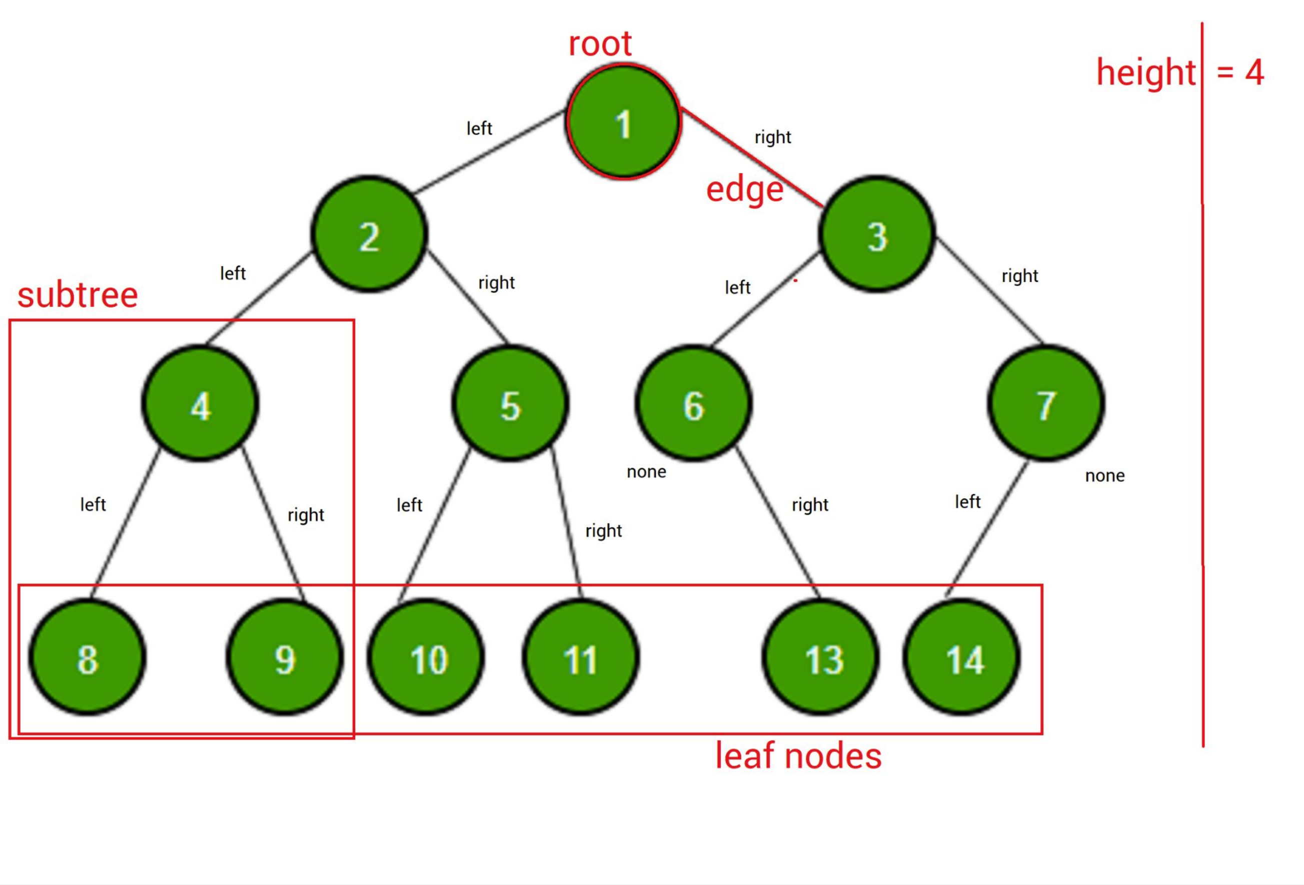
Binary Trees

- A binary tree is represented by a pointer to the topmost node of the tree
- If the tree is empty, then the value of the root is None



Tree Terminology

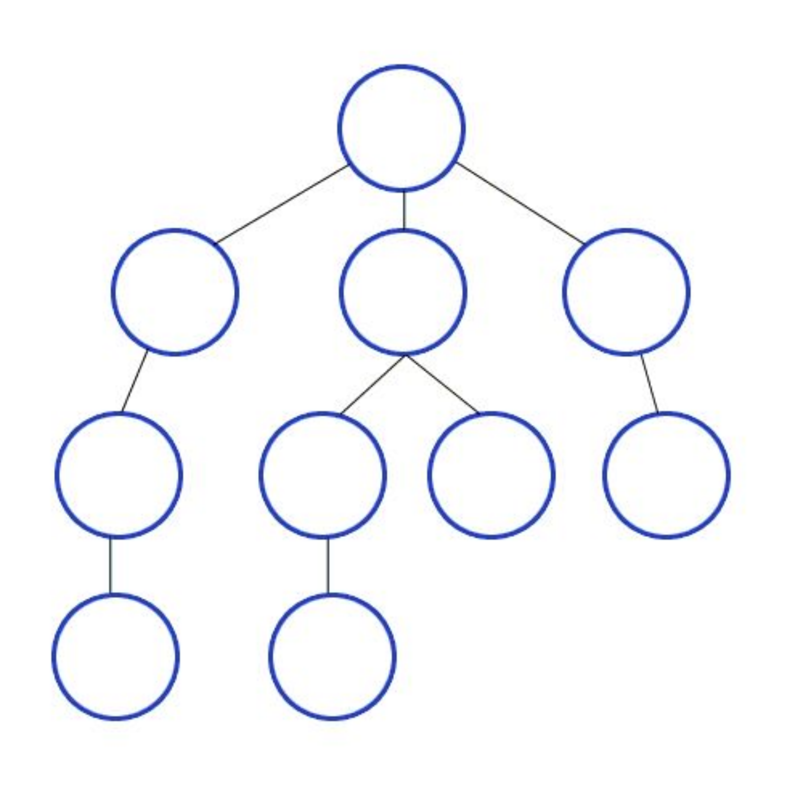
- Root
- Edge
- Leaves
- Subtree
- Height



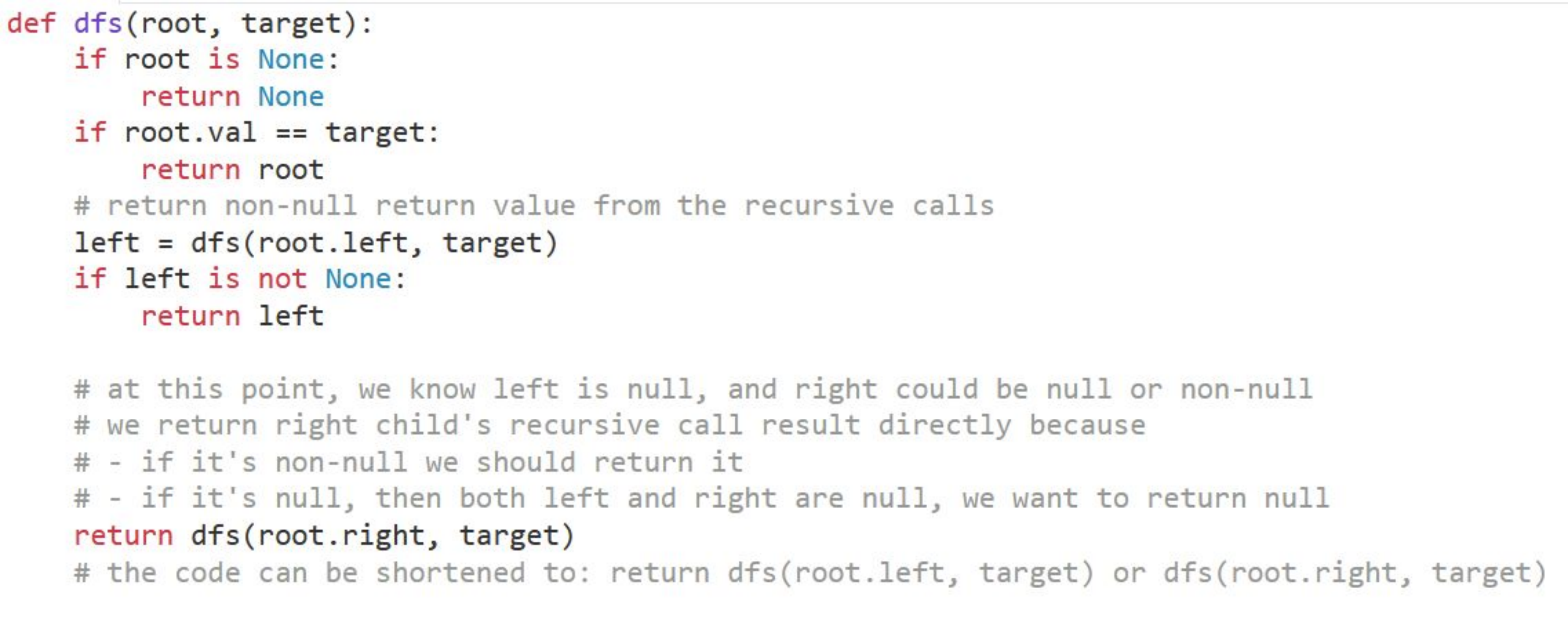
DFS Fundamentals

Depth First Search (DFS)

- Depth First means we go as deep as we can to look for a value
  - I.e. we go down the rabbit hole as far as possible, then come back up and try again with another rabbit hole
- Two different options
  - Iterative
  - Recursive



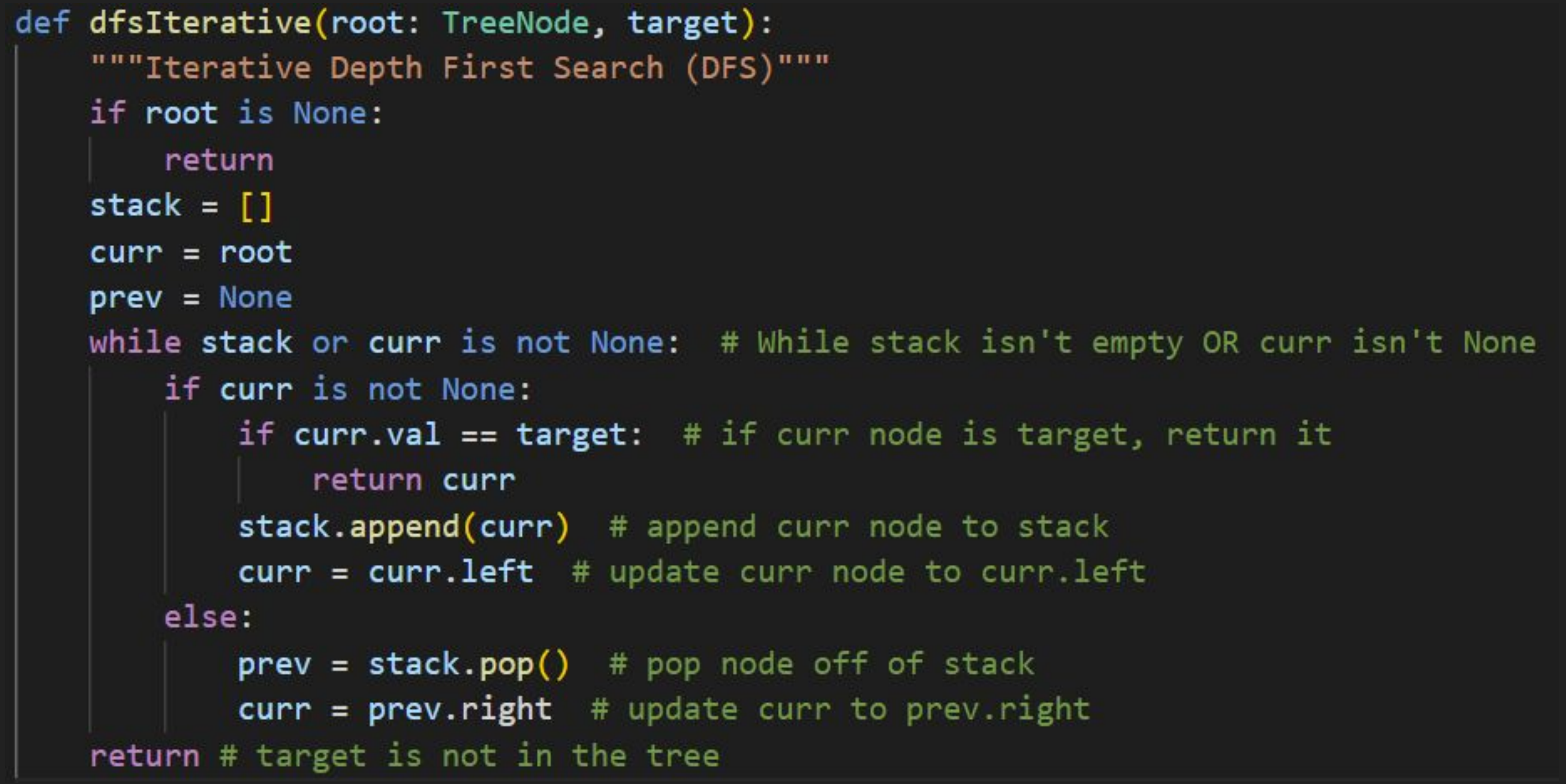
Recursive DFS Boilerplate Code



```python
def dfs(root, target):
    if root is None:
        return None
    if root.val == target:
        return root
    # return non-null return value from the recursive calls
    left = dfs(root.left, target)
    if left is not None:
        return left

    # at this point, we know left is null, and right could be null or non-null
    # we return right child's recursive call result directly because
    # - if it's non-null we should return it
    # - if it's null, then both left and right are null, we want to return null
    return dfs(root.right, target)
    # the code can be shortened to: return dfs(root.left, target) or dfs(root.right, target)
```

Iterative DFS Code



```python
def dfsIterative(root: TreeNode, target):
    """Iterative Depth First Search (DFS)"""
    if root is None:
        return
    stack = []
    curr = root
    prev = None
    while stack or curr is not None:  # While stack isn't empty OR curr isn't None
        if curr is not None:
            if curr.val == target:  # if curr node is target, return it
                return curr
            stack.append(curr)  # append curr node to stack
            curr = curr.left  # update curr node to curr.left
        else:
            prev = stack.pop()  # pop node off of stack
            curr = prev.right  # update curr to prev.right
    return # target is not in the tree
```

LeetCode Problems

Example Problem #1 - LeetCode 206.

- https://leetcode.com/problems/reverse-linked-list/

- Going back to Linked Lists...
- You (should have) already implemented an iterative solution...
- Can you do a recursive one?
- Time Complexity?
  - O(n)
- Space Complexity?
  - O(n)

Example Problem #2 - LeetCode 509.

- https://leetcode.com/problems/fibonacci-number/

- Classic recursion problem…
- Time Complexity?
  - $O(2^N)$ ← Ew, wtf???!!!!
- Space Complexity?
  - O(n)
- Found a better solution? Post your code into the discord!

Example Problem #3 - LeetCode 104.

- https://leetcode.com/problems/maximum-depth-of-binary-tree/

- Can we implement what we have learned to solve this problem?
  - DFS will work
    - You can use an iterative or recursive solution
  - Other algorithms will also work, but we haven’t talked about them yet.
- Time Complexity?
  - O(n)
- Space Complexity?
  - O(1)

Until next time…

Keep practicing

Practice Problems

Easy (supposedly):

- https://leetcode.com/problems/power-of-four
- https://leetcode.com/problems/path-sum/

A bit more difficult:

- https://leetcode.com/problems/add-two-numbers/

Quite difficult (but also cool):

- https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/

Questions?