Lecture 2: Tree

Xiyu Li
2022.8.13

Content

Basic Concepts about Tree

- Binary Tree
- Binary Search Tree
- Balanced Tree
- Complete Binary Tree
- Perfect Binary Tree
- Full Binary Tree

Tree Traversal

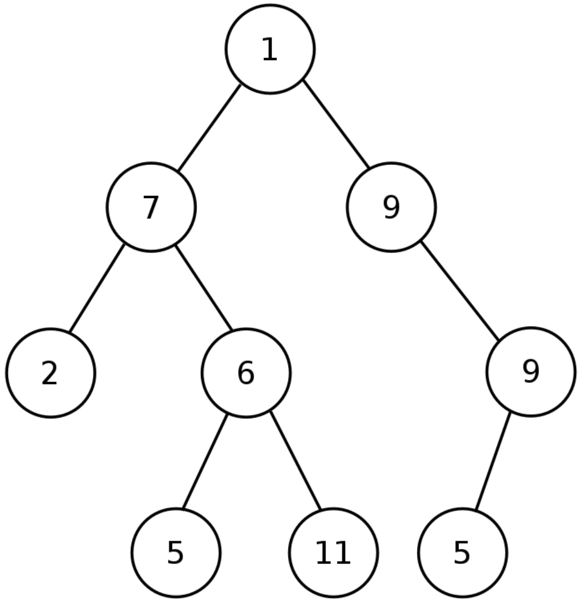
- DFS: In/pre/post order traversal
- BFS: Level order traversal

Example Questions

Basic Concepts

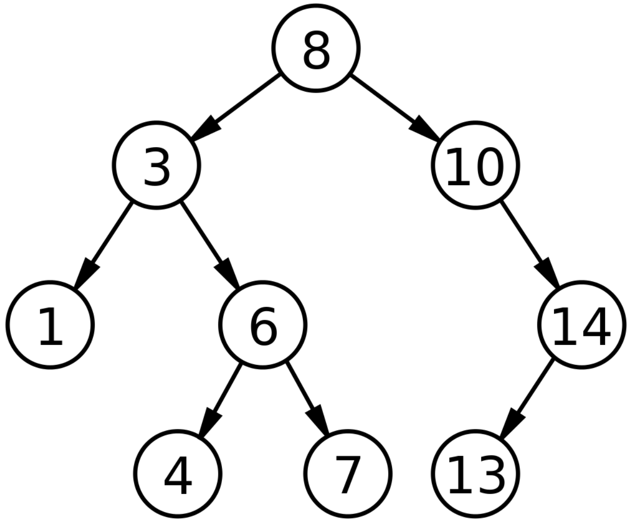
Binary Tree

A tree data structure in which each node has at most two children, which are referred to as the left child, the right child.



Binary Search Tree

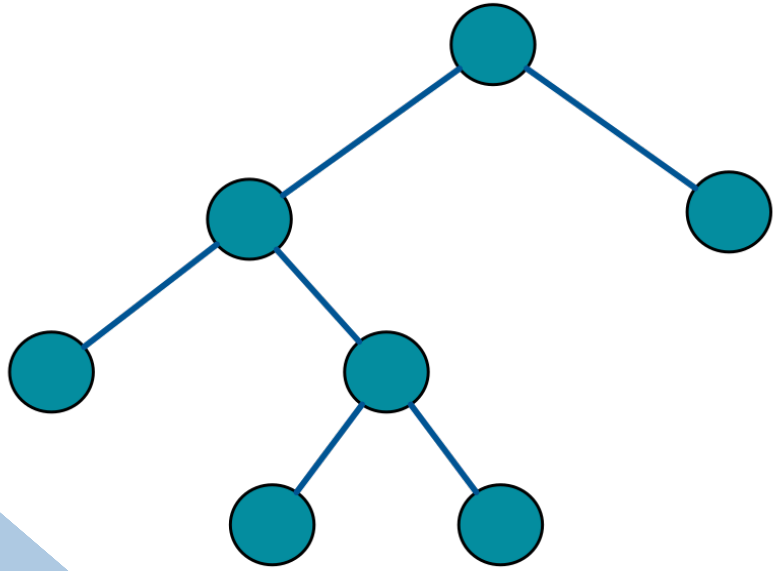
BST also called an ordered or sorted binary tree, is a rooted binary tree data structure with the key of each internal node being greater than all the keys in the respective node's left subtree and less than the ones in its right subtree.



Basic Concepts

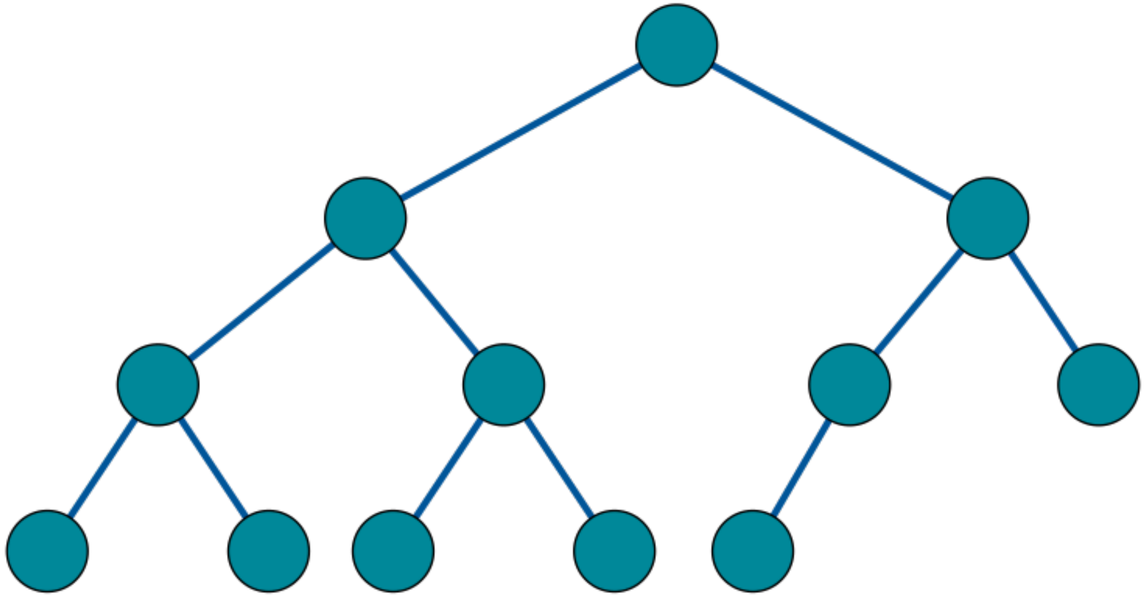
Full Binary Tree

A binary tree where every node either has two children or is a leaf.



Complete Binary Tree

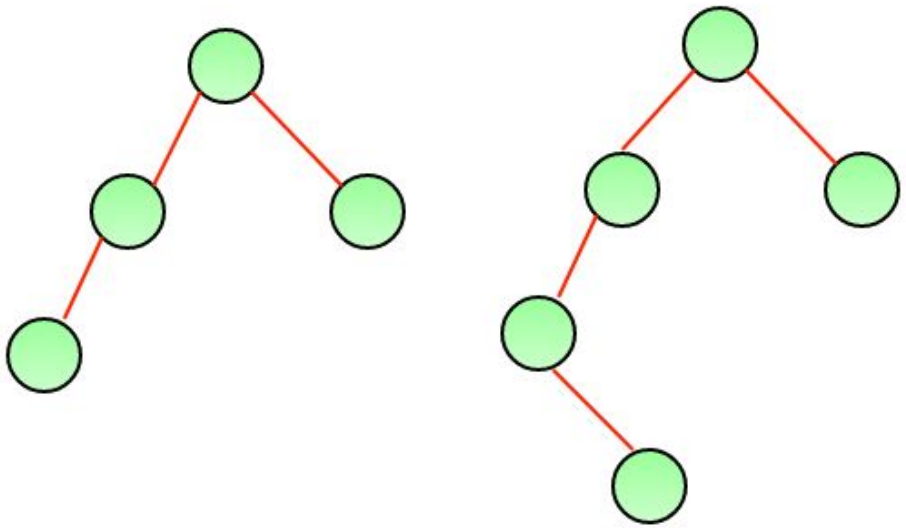
A binary tree in which every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible. It can have between 1 and 2h nodes at the last level h.



Basic Concepts

Balanced Binary Tree

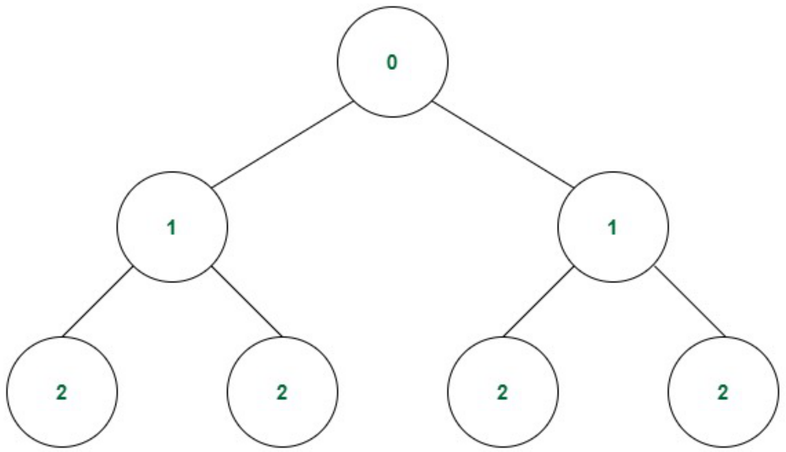
A binary tree structure in which the left and right subtrees of every node differ in height by no more than 1



A height balanced tree   Not a height balanced tree

Perfect Binary Tree

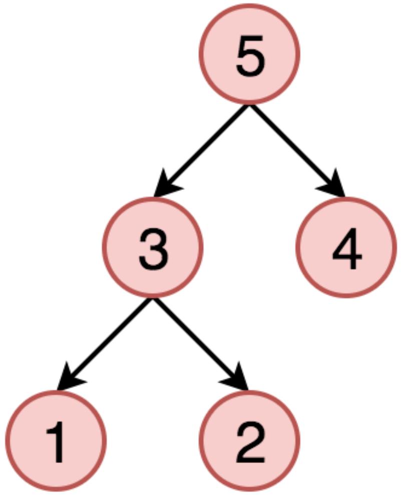
A perfect binary tree is a binary tree in which all interior nodes have two children, and all leaves have the same depth or same level



Tree Traversal

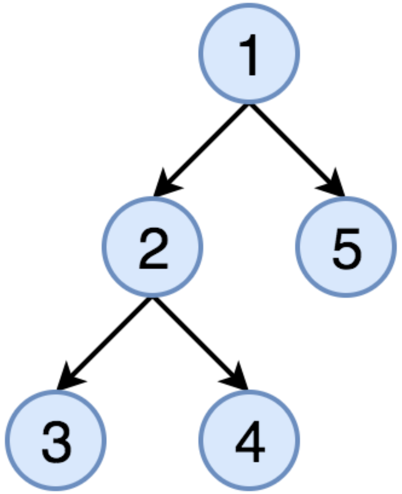
DFS Postorder

Bottom -> Top
Left -> Right



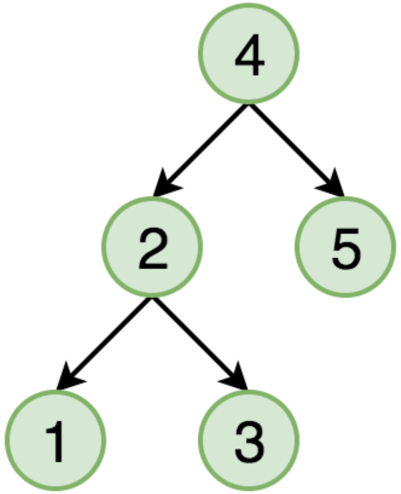
DFS Preorder

Top -> Bottom
Left -> Right



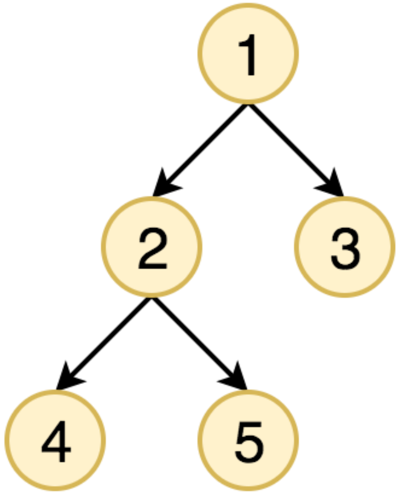
DFS Inorder

Left -> Node -> Right



BFS

Left -> Right
Top -> Bottom



Depth-First Search – In-order

Recursion

```
class Solution:
    def inorderTraversal(self, root: Optional[TreeNode]) -> List[int]:
        def helper(node, ans):
            if node:
                helper(node.left,ans)
                ans.append(node.val)
                helper(node.right, ans)
        result = []
        helper(root, result)
        return result
```

Iteration

```
def inorderTraversal(self, root):
    res, stack = [], []
    while True:
        while root:
            stack.append(root)
            root = root.left
        if not stack:
            return res
        node = stack.pop()
        res.append(node.val)
        root = node.right
```

Depth-First Search – Pre-order

Recursion

```
# recursively
def preorderTraversal1(self, root):
    res = []
    self.dfs(root, res)
    return res

def dfs(self, root, res):
    if root:
        res.append(root.val)
        self.dfs(root.left, res)
        self.dfs(root.right, res)
```

Iteration

```
# iteratively
def preorderTraversal(self, root):
    stack, res = [root], []
    while stack:
        node = stack.pop()
        if node:
            res.append(node.val)
            stack.append(node.right)
            stack.append(node.left)
    return res
```

Depth-First Search – Post-order

Recursion

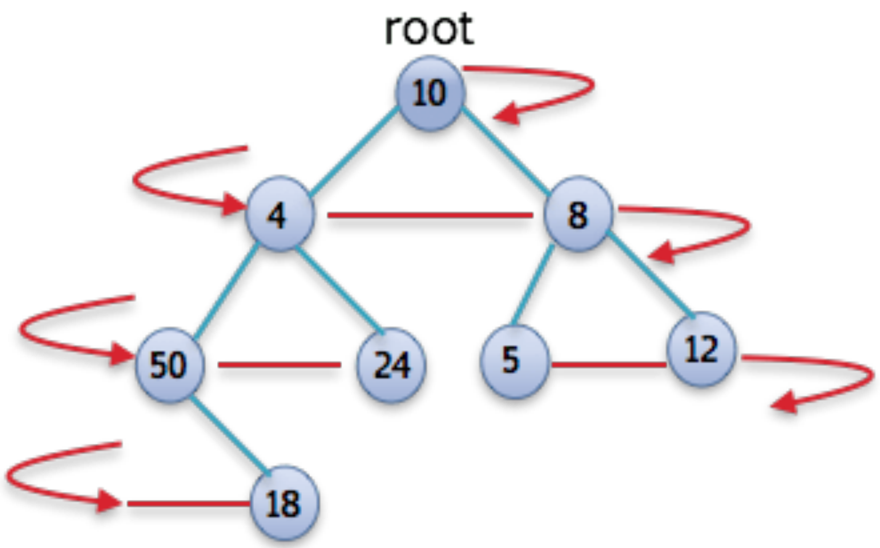
```
# recursively
def postorderTraversal1(self, root):
    res = []
    self.dfs(root, res)
    return res

def dfs(self, root, res):
    if root:
        self.dfs(root.left, res)
        self.dfs(root.right, res)
        res.append(root.val)
```

Iteration

```
# iteratively
def postorderTraversal(self, root):
    res, stack = [], [root]
    while stack:
        node = stack.pop()
        if node:
            res.append(node.val)
            stack.append(node.left)
            stack.append(node.right)
    return res[::-1]
```

Breath-First Search – Level Traversal

Iteration




```python
from collections import deque
class Solution:
    def levelOrder(self, root):
        levels = []
        if not root:
            return levels
        level = 0
        queue = deque([root,])
        while queue:
            # start the current level
            levels.append([])
            # number of elements in the current level
            level_length = len(queue)
            for i in range(level_length):
                node = queue.popleft()
                # fulfill the current level
                levels[level].append(node.val)
                # add child nodes of the current level
                # in the queue for the next level
                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)
            # go to next level
            level += 1
        return levels
```

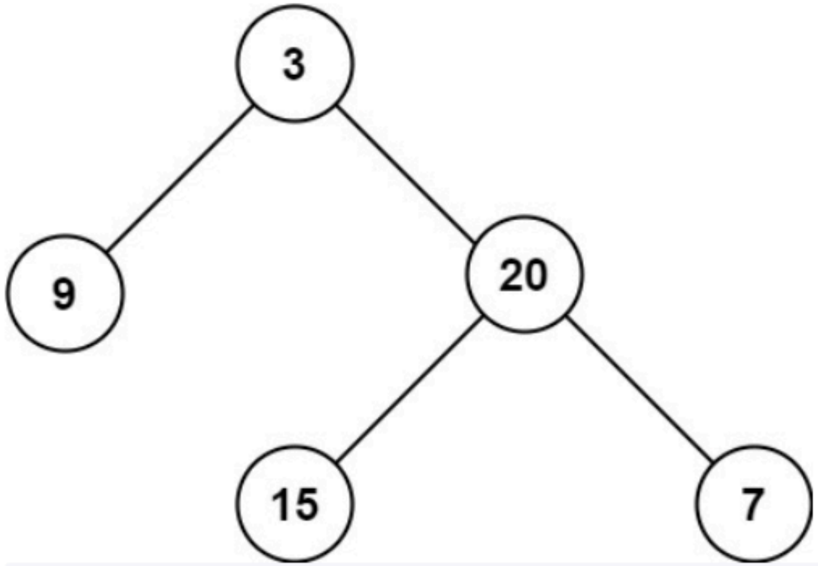
Question 1: A Single Tree

LC104: Maximum Depth of Binary Tree

Given the `root` of a binary tree, return *its maximum depth*.

A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: 3
```

Recursion

```python
class Solution:
    def maxDepth(self, root: Optional[TreeNode]) -> int:
        if root is None:
            return 0
        return
max(self.maxDepth(root.left),self.maxDepth(root.right))+1
```

Iteration

```python
class Solution:
    def maxDepth(self, root):
        stack = []
        if root is not None:
            stack.append((1, root))
        depth = 0
        while stack != []:
            current_depth, root = stack.pop()
            if root is not None:
                depth = max(depth, current_depth)
                stack.append((current_depth + 1, root.left))
                stack.append((current_depth + 1, root.right))
        return depth
```

Question 1: A Single Tree

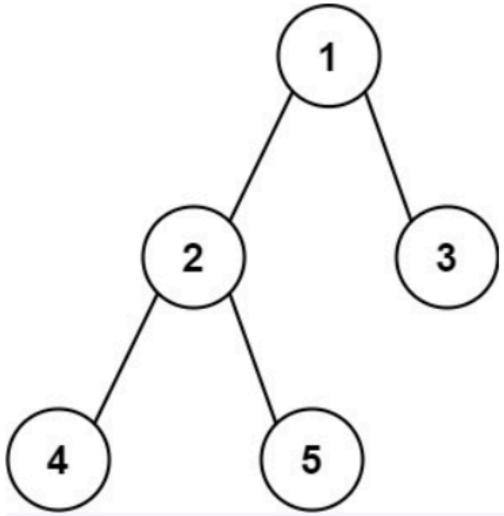
LC543: Diameter of a Binary Tree

Given the `root` of a binary tree, return *the length of the **diameter** of the tree.*

The **diameter** of a binary tree is the **length** of the longest path between any two nodes in a tree. This path may or may not pass through the `root`.

The **length** of a path between two nodes is represented by the number of edges between them.

Example 1:




```
Input: root = [1,2,3,4,5]
Output: 3
Explanation: 3 is the length of the path [4,2,1,3] or
[5,2,1,3].
```

Recursion

```python
class Solution:
    def diameterOfBinaryTree(self, root: TreeNode) -> int:
        diameter = 0

        def longest_path(node):
            if not node:
                return 0
            nonlocal diameter
            # recursively find the longest path in
            # both left child and right child
            left_path = longest_path(node.left)
            right_path = longest_path(node.right)

            # update the diameter if left_path plus right_path is larger
            diameter = max(diameter, left_path + right_path)

            # return the longest one between left_path and right_path;
            # remember to add 1 for the path connecting the node and its pare
            return max(left_path, right_path) + 1

        longest_path(root)
        return diameter
```

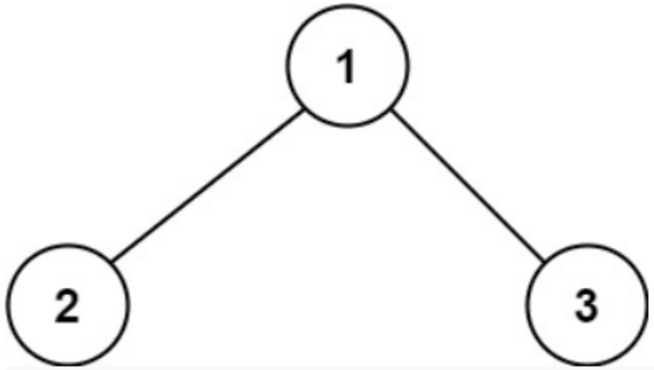
Question 1: A Single Tree

LC124: Binary Tree Max Path Sum

A **path** in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence **at most once**. Note that the path does not need to pass through the root.

The **path sum** of a path is the sum of the node's values in the path.

Given the `root` of a binary tree, return *the maximum* ***path sum*** *of any* ***non-empty*** *path*.



Input: root = [1,2,3]
Output: 6
Explanation: The optimal path is 2 -> 1 -> 3 with a path sum of 2 + 1 + 3 = 6.

Recursion

```
class Solution:
    def maxPathSum(self, root):
        def max_gain(node):
            nonlocal max_sum
            if not node:
                return 0
            # max sum on the left and right sub-trees of node
            left_gain = max(max_gain(node.left), 0)
            right_gain = max(max_gain(node.right), 0)
            # the price to start a new path where `node` is a highes
            price_newpath = node.val + left_gain + right_gain
            # update max_sum if it's better to start a new path
            max_sum = max(max_sum, price_newpath)
            # for recursion :
            # return the max gain if continue the same path
            return node.val + max(left_gain, right_gain)
        max_sum = float('-inf')
        max_gain(root)
        return max_sum
```

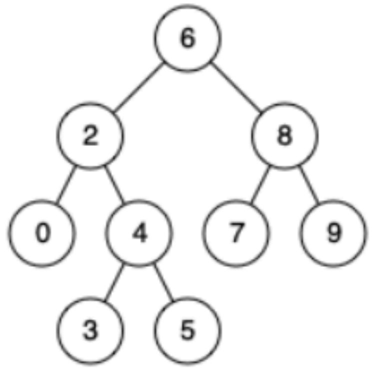
Question 2: Two Trees/Two Nodes

235. Lowest Common Ancestor of a Binary Search Tree

Given a binary search tree (BST), find the lowest common ancestor (LCA) node of two given nodes in the BST.

According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where we allow **a node to be a descendant of itself**)."

Example 1:




```
Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8
Output: 6
Explanation: The LCA of nodes 2 and 8 is 6.
```

Recursion

```python
class Solution:
    def lowestCommonAncestor(self, root, p, q):
        # Value of current node or parent node.
        parent_val = root.val
        # Value of p
        p_val = p.val
        # Value of q
        q_val = q.val
        # If both p and q are greater than parent
        if p_val > parent_val and q_val > parent_val:
            return self.lowestCommonAncestor(root.right, p, q)
        # If both p and q are lesser than parent
        elif p_val < parent_val and q_val < parent_val:
            return self.lowestCommonAncestor(root.left, p, q)
        # We have found the split point, i.e. the LCA node.
        else:
            return root
```

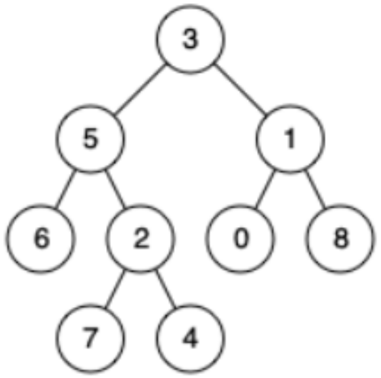
Question 2: Two Trees/Two Nodes

236. Lowest Common Ancestor of a Binary Tree

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where we allow **a node to be a descendant of itself**)."

Example 1:




```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p =
5, q = 1
Output: 3
Explanation: The LCA of nodes 5 and 1 is 3.
```

Recursion - Intuition

- Traverse the tree in a depth first manner.
- The moment you encounter either of the nodes p or q, return some boolean flag.
- The flag helps to determine if we found the required nodes in any of the paths.
- The least common ancestor would then be the node for which both the subtree recursions return a True flag.
- It can also be the node which itself is one of p or q and for which one of the subtree recursions returns a True flag.

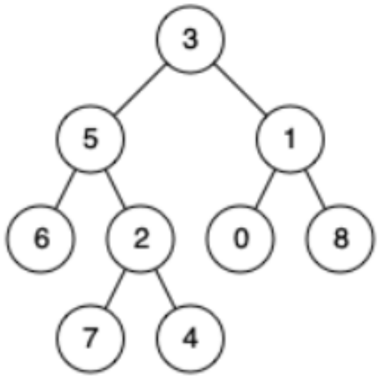
Question 2: Two Trees/Two Nodes

236. Lowest Common Ancestor of a Binary Tree

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where we allow **a node to be a descendant of itself**)."

Example 1:




```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p =
5, q = 1
Output: 3
Explanation: The LCA of nodes 5 and 1 is 3.
```

Recursion - Intuition

1. Start traversing the tree from the root node.
2. If the current node itself is one of `p` or `q`, we would mark a variable `mid` as `True` and continue the search for the other node in the left and right branches.
3. If either of the left or the right branch returns `True`, this means one of the two nodes was found below.
4. If at any point in the traversal, any two of the three flags `left`, `right` or `mid` become `True`, this means we have found the lowest common ancestor for the nodes `p` and `q`.

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

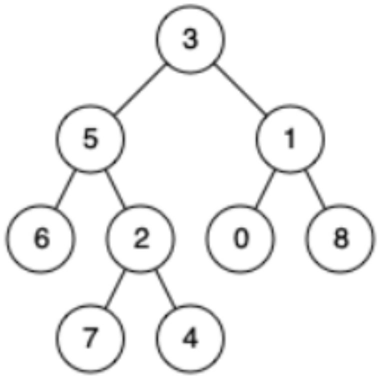
Question 2: Two Trees/Two Nodes

236. Lowest Common Ancestor of a Binary Tree

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where we allow **a node to be a descendant of itself**).”

Example 1:




```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p =
5, q = 1
Output: 3
Explanation: The LCA of nodes 5 and 1 is 3.
```

Recursion

```python
class Solution:
    def __init__(self):
        # Variable to store LCA node.
        self.ans = None
    def lowestCommonAncestor(self, root, p, q):
        def recurse_tree(current_node):
            # If reached the end of a branch, return False.
            if not current_node:
                return False
            # Left Recursion
            left = recurse_tree(current_node.left)
            # Right Recursion
            right = recurse_tree(current_node.right)
            # If the current node is one of p or q
            mid = current_node == p or current_node == q
            # If any two of the three flags left, right or mid
            if mid + left + right >= 2:
                self.ans = current_node
            # Return True if either of the three bool values is
            return mid or left or right
        # Traverse the tree
        recurse_tree(root)
        return self.ans
```

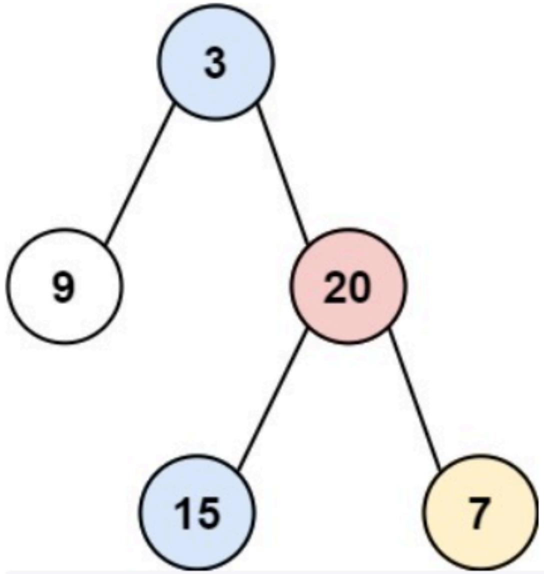
Question 3: Tree Traversal

314. Binary Tree Vertical Order Traversal

Given the `root` of a binary tree, return ***the vertical order traversal of its nodes' values***. (i.e., from top to bottom, column by column).

If two nodes are in the same row and column, the order should be from **left to right**.

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: [[9],[3,15],[20],[7]]
```

Iteration – Level Order Traversal

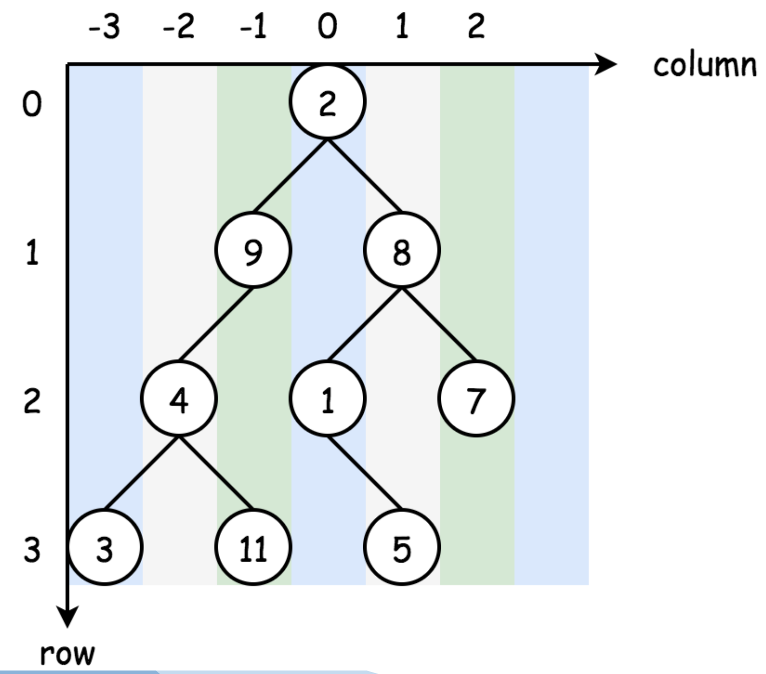
```python
class Solution(object):
    def verticalOrder(self, root):
        if root is None:
            return []
        q = []
        node = root
        q.append([node,0])
        output = {}
        while len(q) > 0:
            node,level = q.pop(0)
            if not (level in output):
                output[level] = [node.val]
            else:
                output[level].append(node.val)
            if node.left is not None:
                q.append([node.left,level-1])
            if node.right is not None:
                q.append([node.right,level+1])
        sortedkeys = sorted(output.keys())
        vertorder = []
        for i in sortedkeys:
            vertorder.append(output[i])
        return vertorder
```

Question 3: Tree Traversal

314. Binary Tree Vertical Order Traversal

Given the `root` of a binary tree, return ***the vertical order traversal of its nodes' values***. (i.e., from top to bottom, column by column).

If two nodes are in the same row and column, the order should be from **left to right**.



Iteration – Level Order Traversal

```python
class Solution(object):
    def verticalOrder(self, root):
        if root is None:
            return []
        q = []
        node = root
        q.append([node,0])
        output = {}
        while len(q) > 0:
            node,level = q.pop(0)
            if not (level in output):
                output[level] = [node.val]
            else:
                output[level].append(node.val)
            if node.left is not None:
                q.append([node.left,level-1])
            if node.right is not None:
                q.append([node.right,level+1])
        sortedkeys = sorted(output.keys())
        vertorder = []
        for i in sortedkeys:
            vertorder.append(output[i])
        return vertorder
```

Homework

Chill~ Not gonna let u do the whole list for HW >_0

	#	Title	Acceptance	Difficulty	Frequency
✓	1650	Lowest Common Ancestor of a Binary Tree III	77.3%	Medium	
✓	236	Lowest Common Ancestor of a Binary Tree	57.3%	Medium	
✓	2096	Step-By-Step Directions From a Binary Tree Node to Another	48.8%	Medium	
✓	366	Find Leaves of Binary Tree	79.8%	Medium	
✓	272	Closest Binary Search Tree Value II	57.9%	Hard	
✓	297	Serialize and Deserialize Binary Tree	54.5%	Hard	
✓	987	Vertical Order Traversal of a Binary Tree	42.1%	Hard	
✓	99	Recover Binary Search Tree	49.6%	Medium	
✓	124	Binary Tree Maximum Path Sum	38.1%	Hard	
✓	426	Convert Binary Search Tree to Sorted Doubly Linked List	64.5%	Medium	
✓	968	Binary Tree Cameras	46.8%	Hard	
✓	199	Binary Tree Right Side View	60.8%	Medium	
✓	314	Binary Tree Vertical Order Traversal	51.7%	Medium	
✓	863	All Nodes Distance K in Binary Tree	61.8%	Medium	
✓	938	Range Sum of BST	85.3%	Easy	
✓	103	Binary Tree Zigzag Level Order Traversal	54.5%	Medium	
	545	Boundary of Binary Tree	43.9%	Medium	
	979	Distribute Coins in Binary Tree	71.9%	Medium	

Homework is here~

- LC98 : Validate Binary Search Tree
- LC230: Kth smallest element in a BST
- LC199: Binary Tree Right Side View
- LC297: Serialize/Deserialize Binary Tree

助教课后点评:
1.Recursion是比tree本身更加宽泛的概念，容易写，但是实际工作中出问题难debug、容易出现stack overflow。
2.Morris Algorithm for Tree Traversal “穿针引线法”: Morris traversal is a tree traversal algorithm that does not employ the use of recursion or a stack. In this traversal, links are created as successors and nodes are printed using these links. Finally, the changes are reverted back to restore the original tree. Space complexity is O(1)
3.Recursion 有类似divide and conquer的思路，把问题大化小
4.汉诺塔题目值得一看
5.本次重点在binary tree，此外Trie面试常考