DSA Questions(Trees)

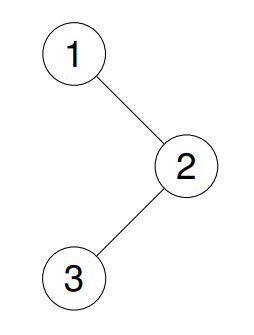
**Very Easy:   
  
1. Binary Tree Inorder Traversal**Given the root of a binary tree, return the inorder traversal of its nodes' values.

**Example 1:**

**Input: root = [1,null,2,3]**

**Output: [1,3,2]**

**Explanation:**

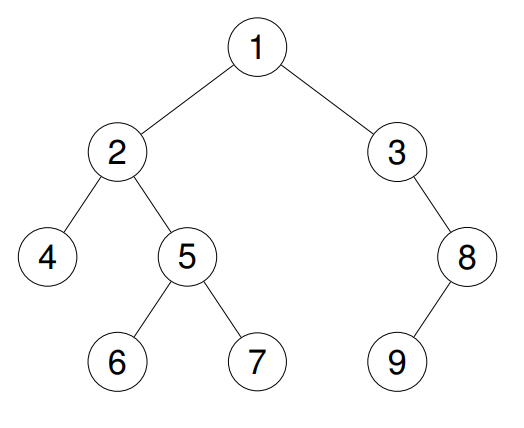
****

**Example 2:**

**Input: root = [1,2,3,4,5,null,8,null,null,6,7,9]**

**Output: [4,2,6,5,7,1,3,9,8]**

**Explanation:**

****

**Constraints:**

**The number of nodes in the tree is in the range [0, 100].**

**-100 <= Node.val <= 100**

**Reference:** [**https://leetcode.com/problems/binary-tree-inorder-traversal/**](https://leetcode.com/problems/binary-tree-inorder-traversal/)

1. **Count Complete Tree Nodes**

Given the root of a complete binary tree, return the number of the nodes in the tree.

According to Wikipedia, every level, except possibly the last, is completely filled in a complete binary tree, and all nodes in the last level are as far left as possible. It can have between 1 and 2h nodes inclusive at the last level h.

Design an algorithm that runs in less than O(n) time complexity.  
  
  
**Example 1:  
Input: root = [1,2,3,4,5,6]**

**Output: 6**

**Example 2:**

**Input: root = []**

**Output: 0**

**Example 3:**

**Input: root = [1]**

**Output: 1**

**Constraints:**

**The number of nodes in the tree is in the range [0, 5 \* 104].**

**0 <= Node.val <= 5 \* 104**

**The tree is guaranteed to be complete.  
  
Reference:** [**https://leetcode.com/problems/count-complete-tree-nodes/description/**](https://leetcode.com/problems/count-complete-tree-nodes/description/)

**3.Binary Tree - Find 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: [3,9,20,null,null,15,7]**

**Output: 3**

**Example 2:**

**Input: [1,null,2]**

**Output: 2  
  
Constraints:**

**The number of nodes in the tree is in the range [0, 104].**

**-100 <= Node.val <= 100  
  
Reference:** [**https://leetcode.com/problems/maximum-depth-of-binary-tree/description/**](https://leetcode.com/problems/maximum-depth-of-binary-tree/description/)

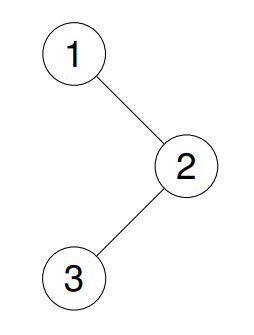
**4.**[**Binary Tree Preorder Traversal**](https://leetcode.com/problems/binary-tree-preorder-traversal/)Given the root of a binary tree, return the preorder traversal of its nodes' values.

**Example 1:**

**Input: root = [1,null,2,3]**

**Output: [1,2,3]**

**Explanation:**

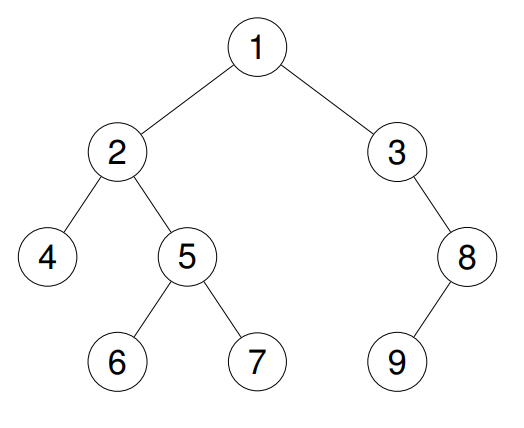
****

**Example 2:**

**Input: root = [1,2,3,4,5,null,8,null,null,6,7,9]**

**Output: [1,2,4,5,6,7,3,8,9]**

**Explanation:**

****

**Constraints:**

**The number of nodes in the tree is in the range [1, 100].**

**1 <= Node.val <= 1000**

**Reference:** [**https://leetcode.com/problems/binary-tree-preorder-traversal/description/**](https://leetcode.com/problems/check-completeness-of-a-binary-tree/description/)

**5. Binary Tree - Sum of All Nodes**Given the root of a binary tree, you need to find the sum of all the node values in the binary tree.  
  
**Example 1:  
Input: root = [1, 2, 3, 4, 5, null, 6]**

**Output: 21**

**Explanation: The sum of all nodes is 1 + 2 + 3 + 4 + 5 + 6 = 21.  
  
Example 2:  
Input: root = [5, 2, 6, 1, 3, 4, 7]**

**Output: 28**

**Explanation: The sum of all nodes is 5 + 2 + 6 + 1 + 3 + 4 + 7 = 28.**

**Reference:** [**http://leetcode.com/problems/sum-of-left-leaves/**](http://leetcode.com/problems/sum-of-left-leaves/)  
**Easy:   
  
1. Same Tree**Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.  
  
**Example 1:**

**Input: p = [1,2,3], q = [1,2,3]**

**Output: true**

**Example 2:**

**Input: p = [1,2], q = [1,null,2]**

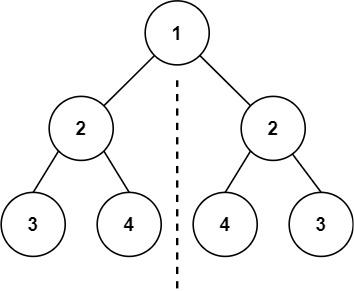
**Output: false  
  
Constraints:**

**The number of nodes in both trees is in the range [0, 100].**

**-104 <= Node.val <= 104  
  
Reference:** [**https://leetcode.com/problems/same-tree/description/?envType=study-plan-v2&envId=top-interview-150**](https://leetcode.com/problems/same-tree/description/?envType=study-plan-v2&envId=top-interview-150)

**2. Symmetric Tree**

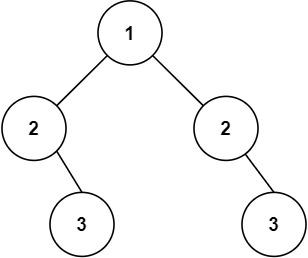
**Example 1:**

****

**Input: root = [1,2,2,3,4,4,3]**

**Output: true**

**Example 2:**

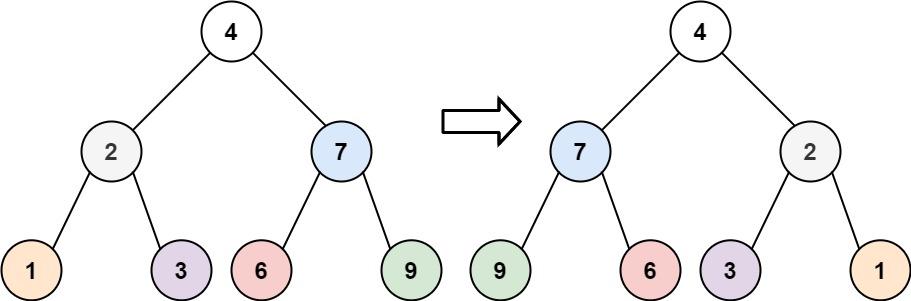
****

**Input: root = [1,2,2,null,3,null,3]**

**Output: false**

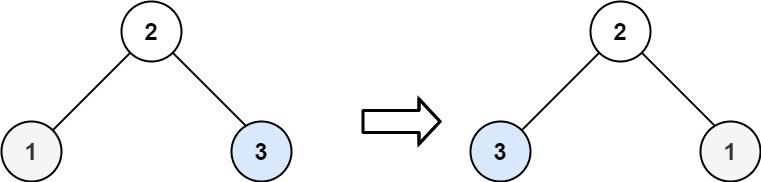
**Constraints:  
The number of nodes in the tree is in the range [1, 1000].  
-100 <= Node.val <= 100**  
**Reference:** [**https://leetcode.com/problems/symmetric-tree/description/**](https://leetcode.com/problems/symmetric-tree/description/)  
  
**3. Invert Binary Tree**Given the root of a binary tree, invert the tree, and return its root.

**Example 1:**

****

**Input: root = [4,2,7,1,3,6,9]  
Output: [4,7,2,9,6,3,1]**

**Example 2:**

****

**Input: root = [2,1,3]  
Output: [2,3,1]  
  
Constraints:**

**The number of nodes in the tree is in the range [0, 100].**

**-100 <= Node.val <= 100**

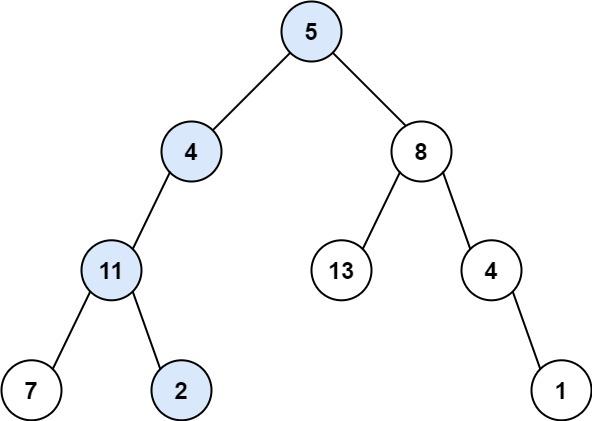
**Refrence:** [**http://leetcode.com/problems/invert-binary-tree/**](http://leetcode.com/problems/invert-binary-tree/)

**4. Leaf Nodes of a Binary Tree**Given a Binary Tree, the task is to count leaves in it. A node is a leaf node if both left and right child nodes of it are NULL.   
  
**Examples:  
Input:   
IMG_256**

**Output: 3  
Explanation: Three leaf nodes are 3, 4 and 5 as both of their left and right child is NULL.  
  
Input:   
IMG_257**

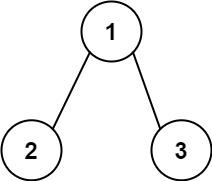
**Output: 3  
Explanation: Three leaf nodes are 4, 6 and 7 as both of their left and right child is NULL.**

Refrence:<http://practice.geeksforgeeks.org/problems/count-leaves-in-binary-tree/1>  
**5. Path Sum**Given a binary tree and a sum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum. Return false if no such path can be found. **Example 1:**

****

**Input: root = [5,4,8,11,null,13,4,7,2,null,null,null,1], targetSum = 22  
Output: true  
Explanation: The root-to-leaf path with the target sum is shown.**

**Example 2:**

****

**Input: root = [1,2,3], targetSum = 5  
Output: false  
Explanation: There are two root-to-leaf paths in the tree:**

**(1 --> 2): The sum is 3.**

**(1 --> 3): The sum is 4.**

**There is no root-to-leaf path with sum = 5.**

**Example 3:**

**Input: root = [], targetSum = 0  
Output: false  
Explanation: Since the tree is empty, there are no root-to-leaf paths.**

**Reference:** [**http://leetcode.com/problems/path-sum/**](http://leetcode.com/problems/path-sum/)

**Medium:   
  
1. Construct Binary Tree from Preorder and Inorder Traversal**Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return the binary tree.

**Example 1:**

**Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]**

**Output: [3,9,20,null,null,15,7]**

**Example 2:**

**Input: preorder = [-1], inorder = [-1]**

**Output: [-1]  
  
Constraints:**

**1 <= preorder.length <= 3000**

**inorder.length == preorder.length**

**-3000 <= preorder[i], inorder[i] <= 3000**

**preorder and inorder consist of unique values.**

**Each value of inorder also appears in preorder.**

**preorder is guaranteed to be the preorder traversal of the tree.**

**inorder is guaranteed to be the inorder traversal of the tree.  
  
Reference:** [**https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/description/?envType=study-plan-v2&envId=top-interview-150**](https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/description/?envType=study-plan-v2&envId=top-interview-150)

**2. Construct Binary Tree from Inorder and Postorder Traversal**Given two integer arrays inorder and postorder where inorder is the inorder traversal of a binary tree and postorder is the postorder traversal of the same tree, construct and return the binary tree.  
  
**Example 1:**

**Input: inorder = [9,3,15,20,7], postorder = [9,15,7,20,3]**

**Output: [3,9,20,null,null,15,7]**

**Example 2:**

**Input: inorder = [-1], postorder = [-1]**

**Output: [-1]  
  
Constraints:**

**1 <= inorder.length <= 3000**

**postorder.length == inorder.length**

**-3000 <= inorder[i], postorder[i] <= 3000**

**inorder and postorder consist of unique values.**

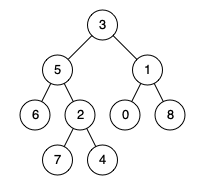
**Each value of postorder also appears in inorder.**

**inorder is guaranteed to be the inorder traversal of the tree.**

**postorder is guaranteed to be the postorder traversal of the tree.  
  
Reference:** [**https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/description/?envType=study-plan-v2&envId=top-interview-150**](https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/description/?envType=study-plan-v2&envId=top-interview-150)

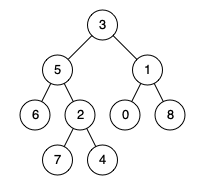
**3. Lowest Common Ancestor of a Binary Tree**Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

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.**

**Example 2:**

****

**Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4  
Output: 5  
Explanation: The LCA of nodes 5 and 4 is 5, since a node can be a descendant of itself according to the LCA definition.**

**Example 3:**

**Input: root = [1,2], p = 1, q = 2  
Output: 1  
  
Constraints:**

**The number of nodes in the tree is in the range [2, 105].**

**-109 <= Node.val <= 109**

**All Node.val are unique.**

**p != q**

**p and q will exist in the tree.**

**Reference:** [**https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/description/?envType=study-plan-v2&envId=top-interview-150**](https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/description/?envType=study-plan-v2&envId=top-interview-150)  
  
  
**4. Sum Root to Leaf Numbers**You are given the root of a binary tree containing digits from 0 to 9 only.

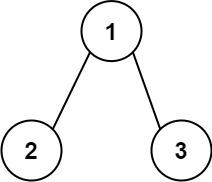
Each root-to-leaf path in the tree represents a number.

For example, the root-to-leaf path 1 -> 2 -> 3 represents the number 123.

Return the total sum of all root-to-leaf numbers. Test cases are generated so that the answer will fit in a 32-bit integer.

A leaf node is a node with no children.

**Example 1:**

****

**Input: root = [1,2,3]Output: 25**

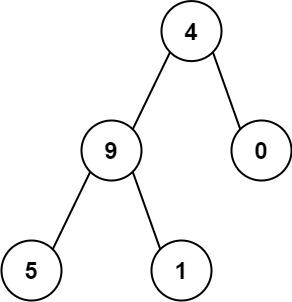
**Explanation:**

**The root-to-leaf path 1->2 represents the number 12.**

**The root-to-leaf path 1->3 represents the number 13.**

**Therefore, sum = 12 + 13 = 25.**

**Example 2:**

****

**Input: root = [4,9,0,5,1]Output: 1026**

**Explanation:**

**The root-to-leaf path 4->9->5 represents the number 495.**

**The root-to-leaf path 4->9->1 represents the number 491.**

**The root-to-leaf path 4->0 represents the number 40.**

**Therefore, sum = 495 + 491 + 40 = 1026.**

**Constraints:**

**The number of nodes in the tree is in the range [1, 1000].**

**0 <= Node.val <= 9**

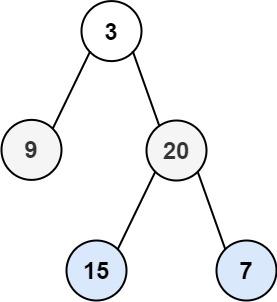
**The depth of the tree will not exceed 10.**

**Reference:** [**https://leetcode.com/problems/sum-root-to-leaf-numbers/description/?envType=study-plan-v2&envId=top-interview-150**](https://leetcode.com/problems/sum-root-to-leaf-numbers/description/?envType=study-plan-v2&envId=top-interview-150)

1. **Binary Tree Level Order Traversal**

Given the root of a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).

**Example 1:**

****

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

**Output: [[3],[9,20],[15,7]]**

**Example 2:**

**Input: root = [1]**

**Output: [[1]]**

**Example 3:**

**Input: root = []**

**Output: []  
  
Constraints:**

**The number of nodes in the tree is in the range [0, 2000].**

**-1000 <= Node.val <= 1000**

**Reference:**[**https://leetcode.com/problems/binary-tree-level-order-traversal/description/?envType=study-plan-v2&envId=top-interview-150**](https://leetcode.com/problems/binary-tree-level-order-traversal/description/?envType=study-plan-v2&envId=top-interview-150)

**Hard :**

1. **Populating Next Right Pointers in Each Node**

Given a binary tree

struct Node {

int val;

Node \*left;

Node \*right;

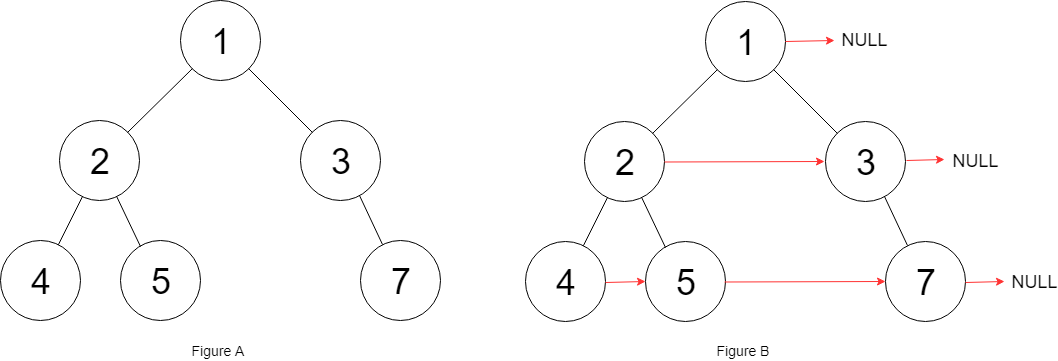
Node \*next;

}

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

Initially, all next pointers are set to NULL.

**Example 1:**

****

**Input: root = [1,2,3,4,5,null,7]**

**Output: [1,#,2,3,#,4,5,7,#]**

**Explanation: Given the above binary tree (Figure A), your function should populate each next pointer to point to its next right node, just like in Figure B. The serialized output is in level order as connected by the next pointers, with '#' signifying the end of each level.**

**Example 2:**

**Input: root = []Output: []**

**Constraints:**

**The number of nodes in the tree is in the range [0, 6000].**

**-100 <= Node.val <= 100**

**Follow-up:**

**You may only use constant extra space.**

**The recursive approach is fine. You may assume implicit stack space does not count as extra space for this problem.**

**Reference:** <http://leetcode.com/problems/populating-next-right-pointers-in-each-node>

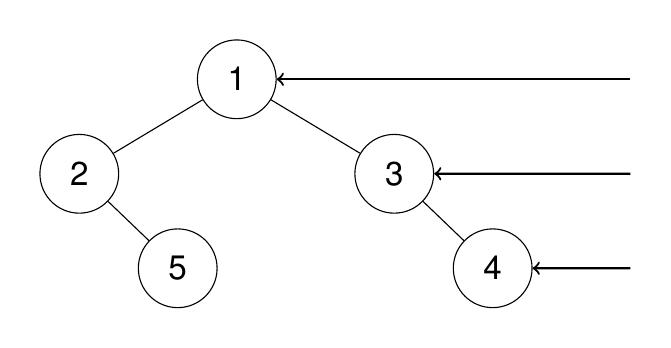
1. **Binary Tree Right Side View**

Given the root of a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.

**Example 1:**

**Input: root = [1,2,3,null,5,null,4]**

**Output: [1,3,4]**

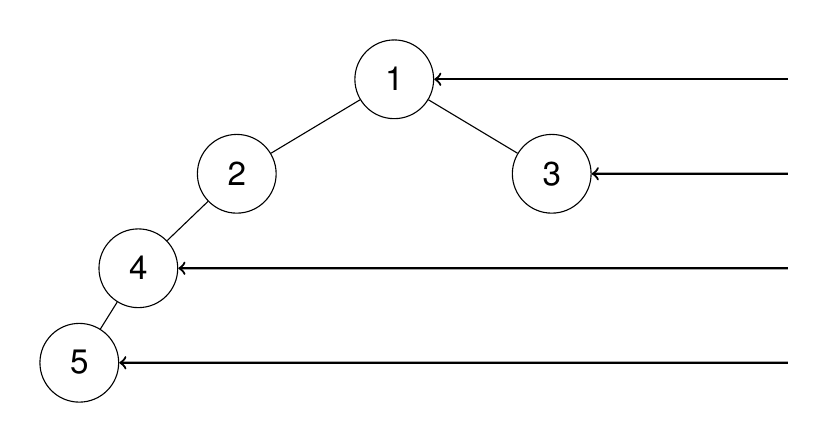
**Explanation:**

**Example 2:**

**Input: root = [1,2,3,4,null,null,null,5]**

**Output: [1,3,4,5]**

**Explanation:**

****

**Example 3:**

**Input: root = [1,null,3]**

**Output: [1,3]**

**Example 4:**

**Input: root = []**

**Output: []**

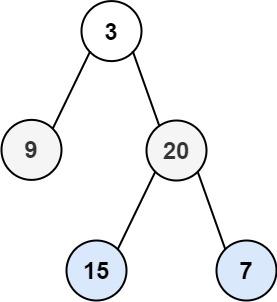
**Constraints:**

**The number of nodes in the tree is in the range [0, 100].**

**-100 <= Node.val <= 100**

**Reference:** [**https://leetcode.com/problems/binary-tree-right-side-view/description/?envType=study-plan-v2&envId=top-interview-150**](https://leetcode.com/problems/binary-tree-right-side-view/description/?envType=study-plan-v2&envId=top-interview-150)

1. **Binary Tree Zigzag Level Order Traversal**Given the root of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between).  
     
   **Example 1:**

****

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

**Output: [[3],[20,9],[15,7]]**

**Example 2:**

**Input: root = [1]**

**Output: [[1]]**

**Example 3:**

**Input: root = []**

**Output: []**

**Constraints:**

**The number of nodes in the tree is in the range [0, 2000].**

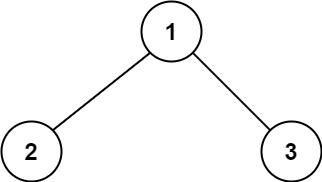
**-100 <= Node.val <= 100**

**Reference:** [**https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/description/?envType=study-plan-v2&envId=top-interview-150**](https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/description/?envType=study-plan-v2&envId=top-interview-150)

1. **Binary Tree Maximum 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.  
  
**Example 1:**

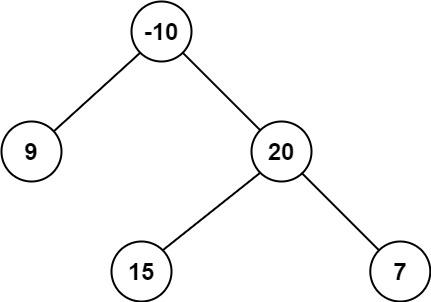
****

**Input: root = [1,2,3]**

**Output: 6**

**Explanation: The optimal path is 2 -> 1 -> 3 with a path sum of 2 + 1 + 3 = 6.**

**Example 2:**

****

**Input: root = [-10,9,20,null,null,15,7]**

**Output: 42**

**Explanation: The optimal path is 15 -> 20 -> 7 with a path sum of 15 + 20 + 7 = 42.**

**Constraints:**

**The number of nodes in the tree is in the range [1, 3 \* 104].**

**-1000 <= Node.val <= 1000**

**Reference:** [**https://leetcode.com/problems/binary-tree-maximum-path-sum/description/?envType=study-plan-v2&envId=top-interview-150**](https://leetcode.com/problems/binary-tree-maximum-path-sum/description/?envType=study-plan-v2&envId=top-interview-150)

**5. Kth Smallest Element in a BST (Binary Search Tree)**

Given a binary search tree (BST), write a function to find the kth smallest element in the tree.

**Input: root = [3,1,4,null,2], k = 1**

**Output: 1**

**Explanation: The inorder traversal of the BST is [1, 2, 3, 4], and the 1st smallest element is 1.**

**Input: root = [5,3,6,2,4,null,null,1], k = 3**

**Output: 3**

**Explanation: The inorder traversal of the BST is [1, 2, 3, 4, 5, 6], and the 3rd smallest element is 3.**

**Constraints:**

**The number of nodes in the tree is in the range [1, 1000].**

**-10^4 ≤ Node.val ≤ 10^4.**

Reference: <http://leetcode.com/problems/kth-smallest-element-in-a-bst/>

**Very Hard :**

1. **Count Paths That Can Form a Palindrome in a Tree**

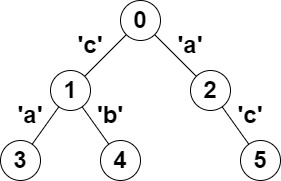
You are given a tree (i.e. a connected, undirected graph that has no cycles) rooted at node 0 consisting of n nodes numbered from 0 to n - 1. The tree is represented by a 0-indexed array parent of size n, where parent[i] is the parent of node i. Since node 0 is the root, parent[0] == -1.

You are also given a string s of length n, where s[i] is the character assigned to the edge between i and parent[i]. s[0] can be ignored.

Return the number of pairs of nodes (u, v) such that u < v and the characters assigned to edges on the path from u to v can be rearranged to form a palindrome.

A string is a palindrome when it reads the same backwards as forwards.

**Example 1:**

****

**Input: parent = [-1,0,0,1,1,2], s = "acaabc"**

**Output: 8**

**Explanation: The valid pairs are:**

**- All the pairs (0,1), (0,2), (1,3), (1,4) and (2,5) result in one character which is always a palindrome.**

**- The pair (2,3) result in the string "aca" which is a palindrome.**

**- The pair (1,5) result in the string "cac" which is a palindrome.**

**- The pair (3,5) result in the string "acac" which can be rearranged into the palindrome "acca".**

**Example 2:**

**Input: parent = [-1,0,0,0,0], s = "aaaaa"**

**Output: 10**

**Explanation: Any pair of nodes (u,v) where u < v is valid.**

**Constraints:**

**n == parent.length == s.length**

**1 <= n <= 105**

**0 <= parent[i] <= n - 1 for all i >= 1**

**parent[0] == -1**

**parent represents a valid tree.**

**s consists of only lowercase English letters.  
  
Reference :** [**https://leetcode.com/problems/count-paths-that-can-form-a-palindrome-in-a-tree/description/?envType=problem-list-v2&envId=tree**](https://leetcode.com/problems/count-paths-that-can-form-a-palindrome-in-a-tree/description/?envType=problem-list-v2&envId=tree)

1. **Maximum Number of K-Divisible Components**

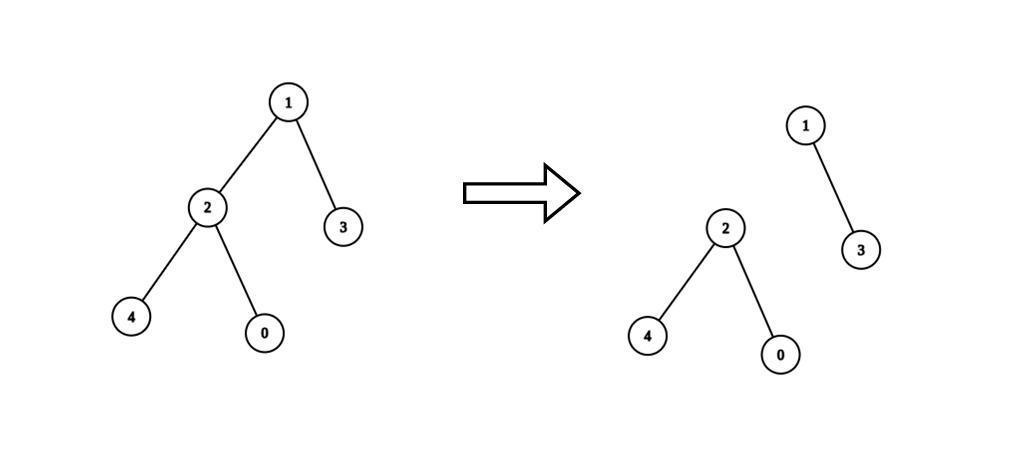
There is an undirected tree with n nodes labeled from 0 to n - 1. You are given the integer n and a 2D integer array edges of length n - 1, where edges[i] = [ai, bi] indicates that there is an edge between nodes ai and bi in the tree.

You are also given a 0-indexed integer array values of length n, where values[i] is the value associated with the ith node, and an integer k.

A valid split of the tree is obtained by removing any set of edges, possibly empty, from the tree such that the resulting components all have values that are divisible by k, where the value of a connected component is the sum of the values of its nodes.

Return the maximum number of components in any valid split.

**Example 1:**

****

**Input: n = 5, edges = [[0,2],[1,2],[1,3],[2,4]], values = [1,8,1,4,4], k = 6**

**Output: 2**

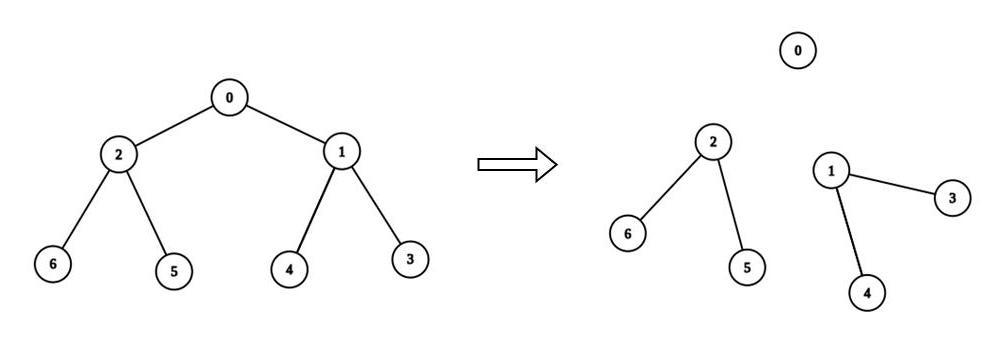
**Explanation: We remove the edge connecting node 1 with 2. The resulting split is valid because:**

**- The value of the component containing nodes 1 and 3 is values[1] + values[3] = 12.**

**- The value of the component containing nodes 0, 2, and 4 is values[0] + values[2] + values[4] = 6.**

**It can be shown that no other valid split has more than 2 connected components.**

**Example 2:**

****

**Input: n = 7, edges = [[0,1],[0,2],[1,3],[1,4],[2,5],[2,6]], values = [3,0,6,1,5,2,1], k = 3**

**Output: 3**

**Explanation: We remove the edge connecting node 0 with 2, and the edge connecting node 0 with 1. The resulting split is valid because:**

**- The value of the component containing node 0 is values[0] = 3.**

**- The value of the component containing nodes 2, 5, and 6 is values[2] + values[5] + values[6] = 9.**

**- The value of the component containing nodes 1, 3, and 4 is values[1] + values[3] + values[4] = 6.**

**It can be shown that no other valid split has more than 3 connected components.**

**Constraints:**

**1 <= n <= 3 \* 104**

**edges.length == n - 1**

**edges[i].length == 2**

**0 <= ai, bi < n**

**values.length == n**

**0 <= values[i] <= 109**

**1 <= k <= 109**

**Sum of values is divisible by k.**

**The input is generated such that edges represents a valid tree.**

**Reference:** [**https://leetcode.com/problems/maximum-number-of-k-divisible-components/description/?envType=problem-list-v2&envId=tree**](https://leetcode.com/problems/maximum-number-of-k-divisible-components/description/?envType=problem-list-v2&envId=tree)

1. **Count Number of Possible Root Nodes**

Alice has an undirected tree with n nodes labeled from 0 to n - 1. The tree is represented as a 2D integer array edges of length n - 1 where edges[i] = [ai, bi] indicates that there is an edge between nodes ai and bi in the tree.

Alice wants Bob to find the root of the tree. She allows Bob to make several guesses about her tree. In one guess, he does the following:

Chooses two distinct integers u and v such that there exists an edge [u, v] in the tree.

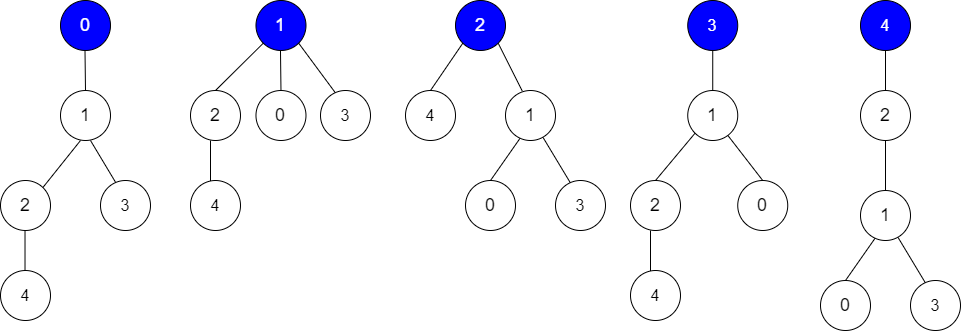
He tells Alice that u is the parent of v in the tree.

Bob's guesses are represented by a 2D integer array guesses where guesses[j] = [uj, vj] indicates Bob guessed uj to be the parent of vj.

Alice being lazy, does not reply to each of Bob's guesses, but just says that at least k of his guesses are true.

Given the 2D integer arrays edges, guesses and the integer k, return the number of possible nodes that can be the root of Alice's tree. If there is no such tree, return 0.

**Example 1:**

****

**Input: edges = [[0,1],[1,2],[1,3],[4,2]], guesses = [[1,3],[0,1],[1,0],[2,4]], k = 3Output: 3Explanation:**

**Root = 0, correct guesses = [1,3], [0,1], [2,4]**

**Root = 1, correct guesses = [1,3], [1,0], [2,4]**

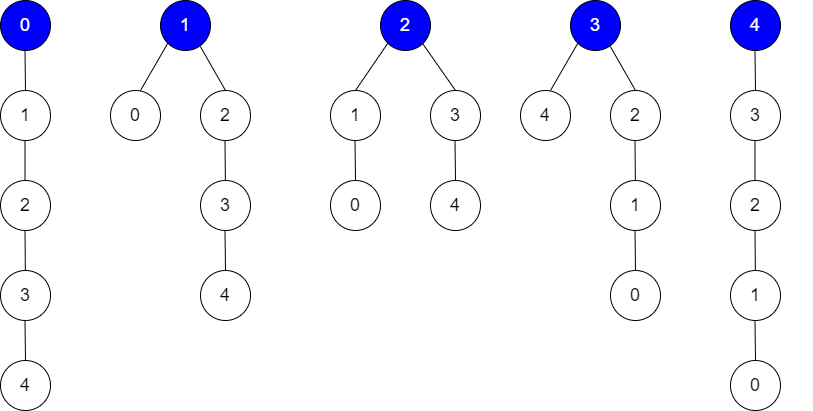
**Root = 2, correct guesses = [1,3], [1,0], [2,4]**

**Root = 3, correct guesses = [1,0], [2,4]**

**Root = 4, correct guesses = [1,3], [1,0]**

**Considering 0, 1, or 2 as root node leads to 3 correct guesses.**

**Example 2:**

****

**Input: edges = [[0,1],[1,2],[2,3],[3,4]], guesses = [[1,0],[3,4],[2,1],[3,2]], k = 1Output: 5Explanation:**

**Root = 0, correct guesses = [3,4]**

**Root = 1, correct guesses = [1,0], [3,4]**

**Root = 2, correct guesses = [1,0], [2,1], [3,4]**

**Root = 3, correct guesses = [1,0], [2,1], [3,2], [3,4]**

**Root = 4, correct guesses = [1,0], [2,1], [3,2]**

**Considering any node as root will give at least 1 correct guess.**

**Constraints:**

**edges.length == n - 1**

**2 <= n <= 105**

**1 <= guesses.length <= 105**

**0 <= ai, bi, uj, vj <= n - 1**

**ai != bi**

**uj != vj**

**edges represents a valid tree.**

**guesses[j] is an edge of the tree.**

**guesses is unique.**

**0 <= k <= guesses.length**

**Reference:** [**https://leetcode.com/problems/count-number-of-possible-root-nodes/description/?envType=problem-list-v2&envId=tree**](https://leetcode.com/problems/count-number-of-possible-root-nodes/description/?envType=problem-list-v2&envId=tree)

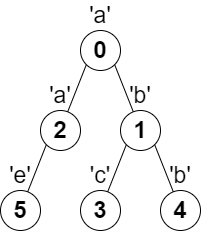
1. **Longest Path With Different Adjacent Characters**

You are given a tree (i.e. a connected, undirected graph that has no cycles) rooted at node 0 consisting of n nodes numbered from 0 to n - 1. The tree is represented by a 0-indexed array parent of size n, where parent[i] is the parent of node i. Since node 0 is the root, parent[0] == -1.

You are also given a string s of length n, where s[i] is the character assigned to node i.

Return the length of the longest path in the tree such that no pair of adjacent nodes on the path have the same character assigned to them.

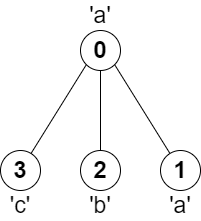
**Example 1:**

****

**Input: parent = [-1,0,0,1,1,2], s = "abacbe"Output: 3Explanation: The longest path where each two adjacent nodes have different characters in the tree is the path: 0 -> 1 -> 3. The length of this path is 3, so 3 is returned.**

**It can be proven that there is no longer path that satisfies the conditions.**

**Example 2:**

****

**Input: parent = [-1,0,0,0], s = "aabc"Output: 3Explanation: The longest path where each two adjacent nodes have different characters is the path: 2 -> 0 -> 3. The length of this path is 3, so 3 is returned.**

**Constraints:**

**n == parent.length == s.length**

**1 <= n <= 105**

**0 <= parent[i] <= n - 1 for all i >= 1**

**parent[0] == -1**

**parent represents a valid tree.**

**s consists of only lowercase English letters.**

**Reference:** [**https://leetcode.com/problems/longest-path-with-different-adjacent-characters/description/?envType=problem-list-v2&envId=tree**](https://leetcode.com/problems/longest-path-with-different-adjacent-characters/description/?envType=problem-list-v2&envId=tree)

1. **Number of Good Paths**

There is a tree (i.e. a connected, undirected graph with no cycles) consisting of n nodes numbered from 0 to n - 1 and exactly n - 1 edges.

You are given a 0-indexed integer array vals of length n where vals[i] denotes the value of the ith node. You are also given a 2D integer array edges where edges[i] = [ai, bi] denotes that there exists an undirected edge connecting nodes ai and bi.

A good path is a simple path that satisfies the following conditions:

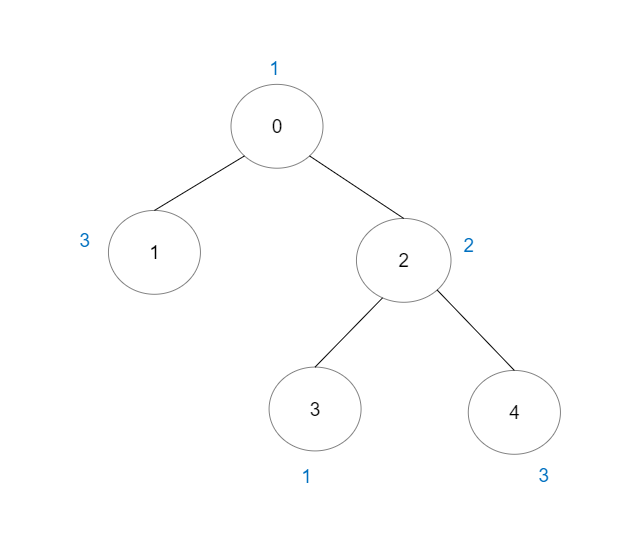
The starting node and the ending node have the same value.

All nodes between the starting node and the ending node have values less than or equal to the starting node (i.e. the starting node's value should be the maximum value along the path).

Return the number of distinct good paths.

Note that a path and its reverse are counted as the same path. For example, 0 -> 1 is considered to be the same as 1 -> 0. A single node is also considered as a valid path.

**Example 1:**

****

**Input: vals = [1,3,2,1,3], edges = [[0,1],[0,2],[2,3],[2,4]]**

**Output: 6**

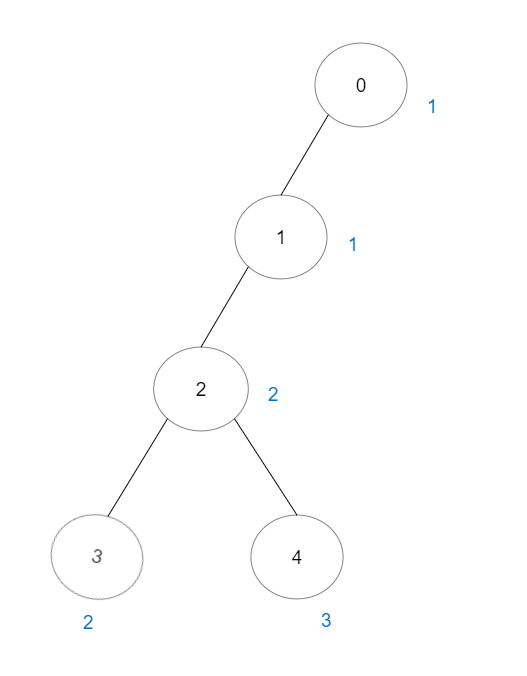
**Explanation: There are 5 good paths consisting of a single node.**

**There is 1 additional good path: 1 -> 0 -> 2 -> 4.**

**(The reverse path 4 -> 2 -> 0 -> 1 is treated as the same as 1 -> 0 -> 2 -> 4.)**

**Note that 0 -> 2 -> 3 is not a good path because vals[2] > vals[0].**

**Example 2:**

****

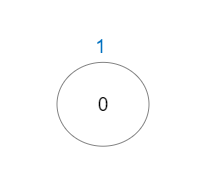
**Input: vals = [1,1,2,2,3], edges = [[0,1],[1,2],[2,3],[2,4]]**

**Output: 7**

**Explanation: There are 5 good paths consisting of a single node.**

**There are 2 additional good paths: 0 -> 1 and 2 -> 3.**

**Example 3:**

****

**Input: vals = [1], edges = []**

**Output: 1**

**Explanation: The tree consists of only one node, so there is one good path.**

**Constraints:**

**n == vals.length**

**1 <= n <= 3 \* 104**

**0 <= vals[i] <= 105**

**edges.length == n - 1**

**edges[i].length == 2**

**0 <= ai, bi < n**

**ai != bi**

**edges represents a valid tree.**

**Reference:** [**https://leetcode.com/problems/number-of-good-paths/description/?envType=problem-list-v2&envId=tree**](https://leetcode.com/problems/number-of-good-paths/description/?envType=problem-list-v2&envId=tree)