

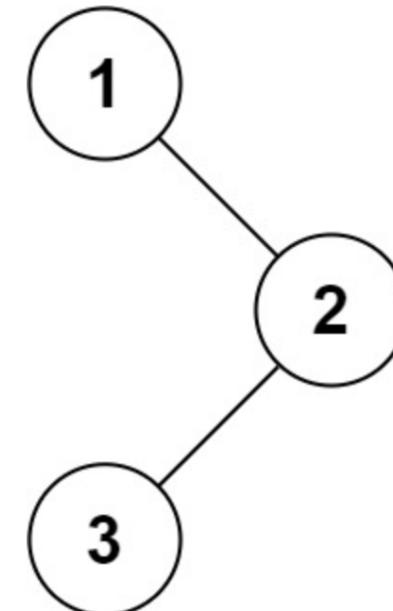
- 链接: <https://leetcode-cn.com/problems/minimum-absolute-difference-in-bst/solution/zhong-xu-bian-li-tuan-mie-xi-lie-er-cha-sou-su-sh/>
- 1.二叉树的中序遍历
 - 2.二叉树的前序遍历
 - 2.51.二叉树的后序遍历
 - 2.52.二叉树的层序遍历
 - 3.N叉树的后序遍历
 - 4.N叉树的前序遍历
 - 5.N叉树的层序遍历
 - 6.翻转二叉树
 - 7.验证二叉搜索树
 - 8.二叉树的最大深度
 - 9.二叉树的最小深度
 - 10.二叉树的序列化与反序列化
 - 11.二叉树的最近公共祖先
 - 12.从前序与中序遍历序列构造二叉树
 - 13.二叉搜索树的最小绝对差
 - 14.二叉树的完全性检验
 - 15.二叉树的锯齿形遍历
 - 16.路径总和
 - 17.路径总和2
 - 18.二叉树的最大路径和
 - 19.平衡二叉树
 - 20.二叉树的直径
 - 21.树的直径
 - 22.N叉树的直径
 - 23.带权树的直径
 - 24.修剪叶子
 - 25.将有序数组转换为二叉搜索树
 - 26.对称二叉树
 - 27.N叉树的路径和

94. 二叉树的中序遍历

难度 简单 1053 收藏 分享 切换为英文 接收动态

给定一个二叉树的根节点 `root`，返回它的 **中序** 遍历。

示例 1:



输入: `root = [1,null,2,3]`

输出: `[1,3,2]`

```

def inorderTraversal(self, root: TreeNode) -> List[int]:
    res = []
    def dfs(root):
        if not root:
            return
        dfs(root.left)
        res.append(root.val)
        dfs(root.right)
    dfs(root)
    return res
  
```

```

class Solution:
    def inorderTraversal(self, root: TreeNode) -> List[int]:
        res = []
        stack = []

        while stack or root:
            if root:
                stack.append(root)
                root = root.left
            else:
                tmp = stack.pop()
                res.append(tmp.val)
                root = tmp.right
        return res

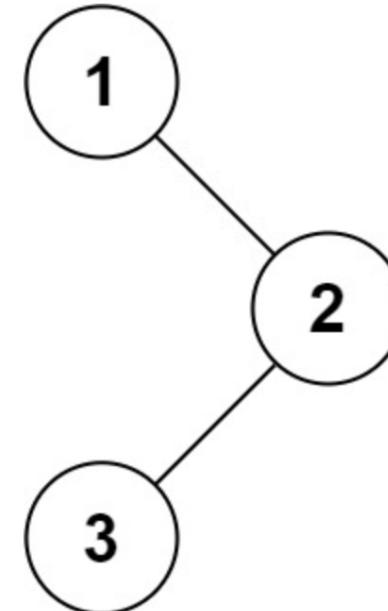
```

144. 二叉树的前序遍历

难度 简单 621 收藏 分享 切换为英文 接收动态 反

给你二叉树的根节点 `root`，返回它节点值的 **前序** 遍历。

示例 1：



输入: `root = [1,null,2,3]`

输出: `[1,2,3]`

```

class Solution:
    def preorderTraversal(self, root: TreeNode) -> List[int]:
        res = []
        stack = []
        while root or stack:
            while root:
                res.append(root.val)
                stack.append(root)
                root = root.left
            root = stack.pop().right
        return res

```

二叉树的后序遍历

```

while h or stack:
    while h:
        res[2].append(h.val)
        stack.append(h)
        h = h.right
    tmp = stack.pop()
    h = tmp.left
res[2] = res[2][::-1]
return res

```

二叉树的层序遍历

```

class Solution:
    def levelOrder(self, root):
        queue = [root]
        res = []
        while queue:
            n = len(queue)
            tmp = []
            for _ in range(n):
                cur = queue.pop(0)
                if not cur:
                    continue
                tmp.append(cur.val)
                if cur.left:
                    queue.append(cur.left)
                if cur.right:
                    queue.append(cur.right)
            if tmp:
                res.append(tmp)
        return res

```

```
def levelOrder1(self, root: TreeNode) -> List[List[int]]:
```

```

res = []
self.level(root, 0, res)
return res

```

```
def level(self, root, level, res):
```

```

if not root:
    return
if len(res) == level:
    res.append([])
res[level].append(root.val)
if root.left:
    self.level(root.left, level + 1, res)
if root.right:
    self.level(root.right, level + 1, res)

```

590. N 叉树的后序遍历

难度 简单 156 收藏 分享 切换为英文 接收动态 反馈

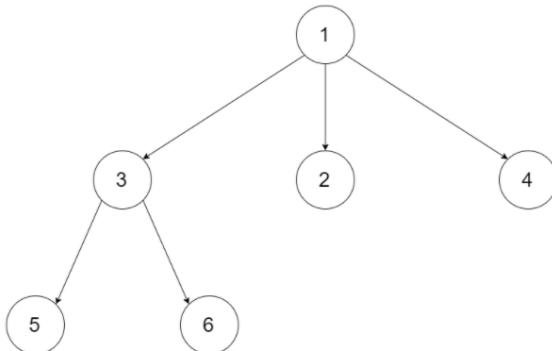
给定一个 N 叉树，返回其节点值的 **后序遍历**。

N 叉树 在输入中按层序遍历进行序列化表示，每组子节点由空值 `null` 分隔（请参见示例）

进阶：

递归法很简单，你可以使用迭代法完成此题吗？

示例 1：



输入: `root = [1,null,3,2,4,null,5,6]`

输出: `[5,6,3,2,4,1]`

class Solution:

```

def postorder(self, root: 'Node') -> List[int]:
    res = []
    if root is None:
        return res

    stack = [root]
    while stack:
        cur = stack.pop()
        res.append(cur.val)
        stack.extend(cur.children)
    return res[::-1]
  
```

class Solution:

```

def postorder(self, root: 'Node') -> List[int]:
    res = []
    if root is None: return res

    def recursion(root, res):
        for child in root.children:
            recursion(child, res)
        res.append(root.val)

    recursion(root, res)
    return res
  
```

589. N 叉树的前序遍历

难度 简单 171 收藏 分享 切换为英文 接收动态 反

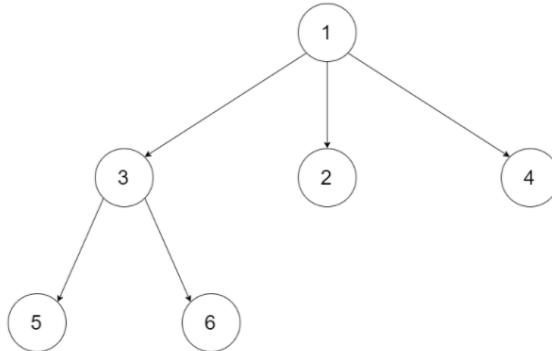
给定一个 N 叉树，返回其节点值的 **前序遍历**。

N 叉树 在输入中按层序遍历进行序列化表示，每组子节点由空值 null 分隔（请参见示例）

进阶：

递归法很简单，你可以使用迭代法完成此题吗？

示例 1：



输入: root = [1,null,3,2,4,null,5,6]

输出: [1,3,5,6,2,4]

class Solution:

```

def preorder(self, root: 'Node') -> List[int]:
    if not root:
        return []
    traversal = [root.val]
    for child in root.children:
        traversal.extend(self.preorder(child))
    return traversal
  
```

```

def preorder1(self, root: 'Node') -> List[int]:
    if not root:
        return []
    stack = [root]
    res = []
    while stack:
        temp = stack.pop()
        res.append(temp.val)
        stack.extend(reversed(temp.children))
    return res
  
```

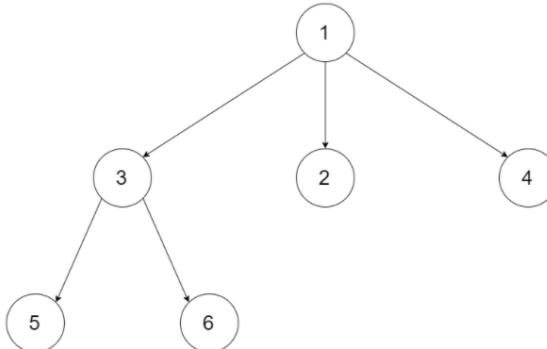
429. N 叉树的层序遍历

难度 中等 山 169 ☆ 收藏 ⚡ 分享 切换为英文 接收动态 反馈

给定一个 N 叉树，返回其节点值的层序遍历。 (即从左到右，逐层遍历)。

树的序列化输入是用层序遍历，每组子节点都由 null 值分隔 (参见示例)。

示例 1：



输入: root = [1,null,3,2,4,null,5,6]

输出: [[1],[3,2,4],[5,6]]

class Solution:

```

def levelOrder(self, root: 'Node') -> List[List[int]]:
    if root is None:
        return []

    result = []
    previous_layer = [root]

    while previous_layer:
        current_layer = []
        result.append([])
        for node in previous_layer:
            result[-1].append(node.val)
            current_layer.extend(node.children)
        previous_layer = current_layer
    return result
  
```

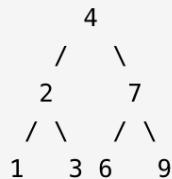
226. 翻转二叉树

难度 简单 山 941 ☆ 收藏 □ 分享 文切换为英文 ▲ 接收动态 □ 反馈

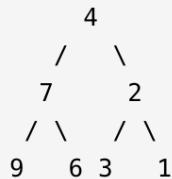
翻转一棵二叉树。

示例：

输入：



输出：



备注：

这个问题是受到 Max Howell 的 原问题 启发的：

谷歌：我们90%的工程师使用您编写的软件(Homebrew)，但是您却无法在面试时在白板二叉树这道题，这太糟糕了。

class Solution:

```

def invertTree(self, root: TreeNode) -> TreeNode:
    if not root:
        return None

    root.left, root.right = root.right, root.left

    self.invertTree(root.left)
    self.invertTree(root.right)

    return root
  
```

class Solution:

```

def invertTree(self, root: TreeNode) -> TreeNode:
    if not root:
        return None

    queue = [root]
    while queue:
        tmp = queue.pop(0)
        tmp.left, tmp.right = tmp.right, tmp.left

        if tmp.left:
            queue.append(tmp.left)
        if tmp.right:
            queue.append(tmp.right)

    return root
  
```

98. 验证二叉搜索树

难度 中等 1156 收藏 分享 切换为英文 接收动态

给定一个二叉树，判断其是否是一个有效的二叉搜索树。

假设一个二叉搜索树具有如下特征：

- 节点的左子树只包含**小于**当前节点的数。
- 节点的右子树只包含**大于**当前节点的数。
- 所有左子树和右子树自身必须也是二叉搜索树。

示例 1：

```
输入:
 2
 / \
1   3
输出: true
```

示例 2：

```
输入:
 5
 / \
1   4
 / \
3   6
输出: false
解释: 输入为: [5,1,4,null,null,3,6]。
根节点的值为 5，但是其右子节点值为 4。
```

class Solution:

```
def isValidBST(self, root: TreeNode) -> bool:
    stack, inorder = [], float('-inf')
```

```
while stack or root:
    while root:
        stack.append(root)
        root = root.left
    root = stack.pop()
```

```
if root.val <= inorder:
    return False
inorder = root.val
root = root.right
return True
```

```
def isValidBST1(self, root: TreeNode) -> bool:
```

```
def helper(node, lower=float('-inf'), upper=float('inf')):
    if not node:
        return True

    val = node.val
    if val <= lower or val >= upper:
        return False

    if not helper(node.right, val, upper):
        return False
    if not helper(node.left, lower, val):
        return False
    return True
return helper(root)
```

二叉树的最大深度

104. 二叉树的最大深度

难度 简单 937 收藏 分享 切换为英文 接收动态 反馈

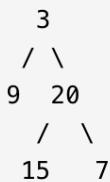
给定一个二叉树，找出其最大深度。

二叉树的深度为根节点到最远叶子节点的最长路径上的节点数。

说明: 叶子节点是指没有子节点的节点。

示例:

给定二叉树 [3,9,20,null,null,15,7]，



返回它的最大深度 3。

class Solution:

```

def maxDepth(self, root: TreeNode) -> int:
    if root is None:
        return 0

    left_height = self.maxDepth(root.left)
    right_height = self.maxDepth(root.right)
    return max(left_height, right_height) + 1
    
```

def maxDepth1(self, root: TreeNode) -> int:

```

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
    
```

二叉树的最大深度

111. 二叉树的最小深度

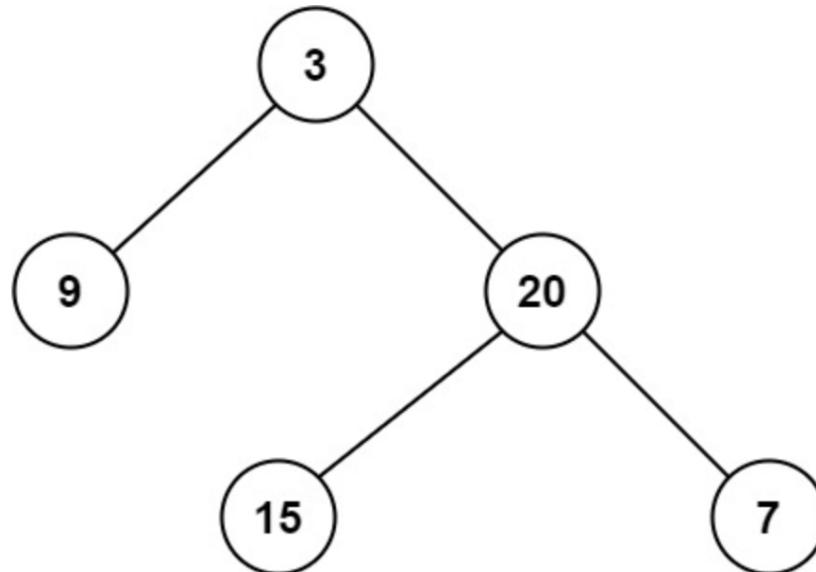
难度 简单 560 收藏 分享 切换为英文 接收动态 反馈

给定一个二叉树，找出其最小深度。

最小深度是从根节点到最近叶子节点的最短路径上的节点数量。

说明：叶子节点是指没有子节点的节点。

示例 1：



输入: root = [3,9,20,null,null,15,7]

输出: 2

class Solution:

```

def minDepth(self, root: TreeNode) -> int:
    if not root:
        return 0

    left_min = self.minDepth(root.left)
    right_min = self.minDepth(root.right)

    if not root.left and not root.right:
        return 1
    elif not root.left or not root.right:
        return left_min + 1 if root.left else right_min + 1
    else:
        return min(left_min, right_min) + 1
  
```

class Solution:

```

def minDepth(self, root: TreeNode) -> int:
    if not root:
        return 0
    queue = [root]
    depth = 1
    while queue:
        n = len(queue)
        for i in range(n):
            cur = queue.pop(0)
            if not cur.left and not cur.right:
                return depth
            if cur.left:
                queue.append(cur.left)
            if cur.right:
                queue.append(cur.right)
        depth += 1
  
```

二叉树的序列化与反序列化

297. 二叉树的序列化与反序列化

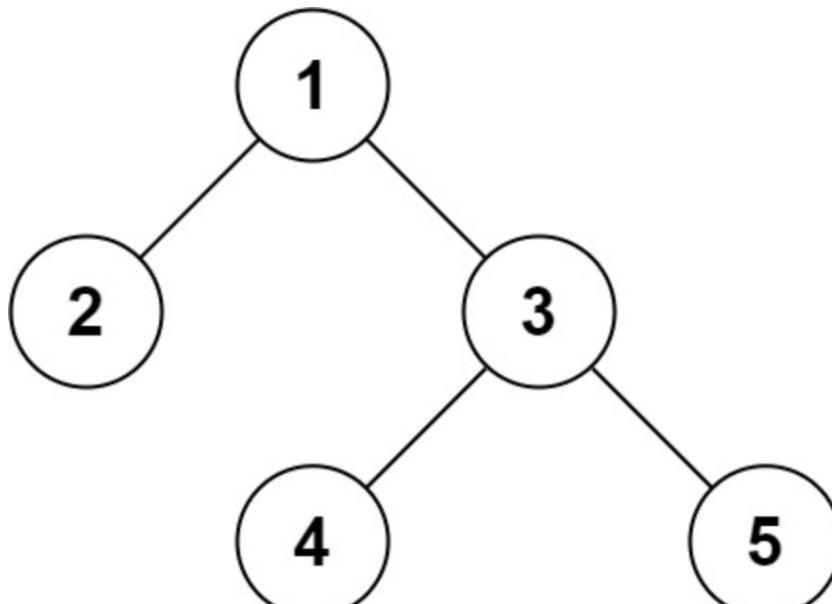
难度 困难 609 收藏 分享 切换为英文 接收动态 反馈

序列化是将一个数据结构或者对象转换为连续的比特位的操作，进而可以将转换后的数据存储在文件或者内存中，同时也可以通过网络传输到另一个计算机环境，采取相反方式重构得到原数据。

请设计一个算法来实现二叉树的序列化与反序列化。这里不限定你的序列 / 反序列化算法执行顺序。只需要保证一个二叉树可以被序列化为一个字符串并且将这个字符串反序列化为原始的树结构。

提示: 输入输出格式与 LeetCode 目前使用的方式一致，详情请参阅 LeetCode 序列化二叉树，并非必须采取这种方式，你也可以采用其他的方法解决这个问题。

示例 1:



输入: root = [1,2,3,null,null,4,5]

输出: [1,2,3,null,null,4,5]

示例 2:

输入: root = []

输出: []

示例 3:

输入: root = [1]

输出: [1]

示例 4:

输入: root = [1,2]

输出: [1,2]

```

class TreeNode(object):
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None
  
```

```

class Codec:
    def serialize(self, root):
        if root is None:
            return 'X,'
```

```

        left = self.serialize(root.left)
        right = self.serialize(root.right)
```

```

        return str(root.val) + ',' + left + right

    def deserialize(self, data):
        data = data.split(',')
        root = self.buildTree(data)
        return root

    def buildTree(self, data):
        val = data.pop(0)
        if val == 'X':
            return None
        node = TreeNode(val)
        node.left = self.buildTree(data)
        node.right = self.buildTree(data)
        return node

```

```

from collections import deque
class Codec:

    def serialize(self, root):
        """Encodes a tree to a single string.

        :type root: TreeNode
        :rtype: str
        """

        if not root:
            return ''
        q = deque()
        q.append(root)
        res = ''
        while q:
            node = q.popleft()
            if node != None:
                res += str(node.val) + ','
                q.append(node.left)
                q.append(node.right)
            else:
                res += 'X,'

        return res

```

```

def deserialize(self, data):
    """Decodes your encoded data to tree.

:type data: str
:rtype: TreeNode
"""

if not data:
    return None
data = data.split(',')
root = TreeNode(data.pop(0))
q = [root]
while q:
    node = q.pop(0)
    if data:
        val = data.pop(0)
        if val != 'X':
            node.left = TreeNode(val)
            q.append(node.left)
    if data:
        val = data.pop(0)
        if val != 'X':
            node.right = TreeNode(val)
            q.append(node.right)
return root

```

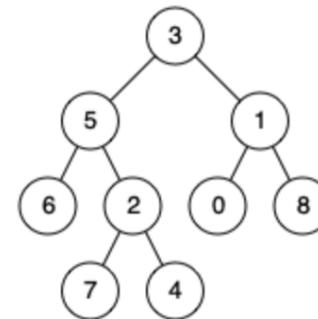
236. 二叉树的最近公共祖先

难度 中等 1238 收藏 分享 切换为英文 接收动态

给定一个二叉树，找到该树中两个指定节点的最近公共祖先。

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个节点 p、q，最近公共祖先表示为 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（一个节点也可以是它自己的祖先）。”

示例 1：



输入: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1

输出: 3

解释: 节点 5 和节点 1 的最近公共祖先是节点 3。

```

class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode',
                           q: 'TreeNode'):
        if not root or root == p or root == q:
            return root
        left = self.lowestCommonAncestor(root.left, p, q)
        right = self.lowestCommonAncestor(root.right, p, q)
        if not left and not right:
            return
        if not left:
            return right
        if not right:
            return left
        return root

```

描述

给定一棵二叉树(保证非空)以及这棵树上的两个节点对应的val值 o1 和 o2, 请找到 o1

注: 本题保证二叉树中每个节点的val值均不相同。

示例1

输入: [3,5,1,6,2,0,8,#,#,7,4],5,1

返回值: 3

```

4 class Solution:
5     def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode',
6                               q: 'TreeNode'):
7         def dfs(root, p, q):
8             if not root or root.val == p or root.val == q:
9                 return root
10            left = dfs(root.left, p, q)
11            right = dfs(root.right, p, q)
12            if not left and not right:
13                return
14            if not left:
15                return right
16            if not right:
17                return left
18            return root
19     return dfs(root, p, q).val

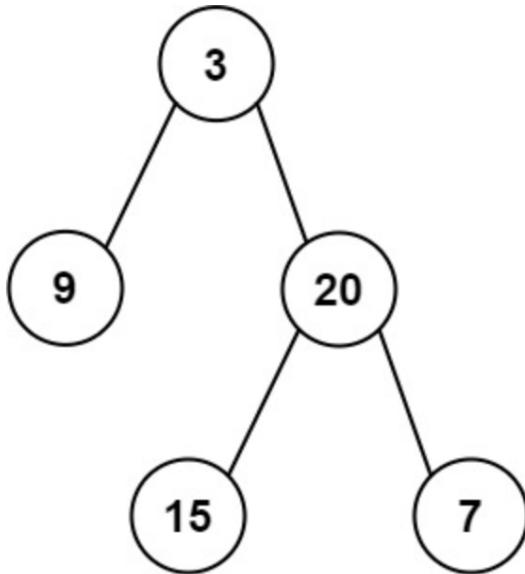
```

105. 从前序与中序遍历序列构造二叉树

难度 中等 1149 收藏 分享 切换为英文 接收动态 反馈

给定一棵树的前序遍历 `preorder` 与中序遍历 `inorder`。请构造二叉树并返回其根节点。

示例 1：



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

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

```

class Solution:
    def buildTree(self, preorder: List[int], inorder: List[int]) ->
        if len(preorder) == 0:
            return None
        root = TreeNode(preorder[0])
        mid = inorder.index(preorder[0])

        root.left = self.buildTree(preorder[1:mid + 1], inorder[:mid])
        root.right = self.buildTree(preorder[mid + 1:], inorder[mid + 1:])

        return root
  
```

530. 二叉搜索树的最小绝对差

难度 简单 264 收藏 分享 切换为英文 接收动态 反馈

给你一棵所有节点为非负值的二叉搜索树，请你计算树中任意两节点的差的绝对值的最小值。

示例：

输入：

```

1
 \
 3
 /
2
  
```

输出：

1

解释：

最小绝对差为 1，其中 2 和 1 的差的绝对值为 1（或者 2 和 3）。

```

class Solution:
    def getMinimumDifference(self, root: TreeNode) -> int:
        p = root
        pre = float('-inf')
        minval = float('inf')
        stack = []
        while p or stack:
            while p:
                stack.append(p)
                p = p.left
            p = stack.pop()
            minval = min(minval, p.val - pre)
            pre = p.val
            p = p.right
        return minval

```

958. 二叉树的完全性检验

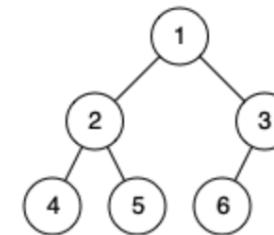
难度 中等 山 142 ★ 收藏 ▾ 分享 ⚖ 切换为英文 ⚙ 接收动态 反馈

给定一个二叉树，确定它是否是一个完全二叉树。

百度百科中对完全二叉树的定义如下：

若设二叉树的深度为 h ，除第 h 层外，其它各层 ($1 \sim h-1$) 的结点数都达到最大个数，第 h 层都连续集中在最左边，这就是完全二叉树。（注：第 h 层可能包含 $1 \sim 2^h$ 个节点。）

示例 1：

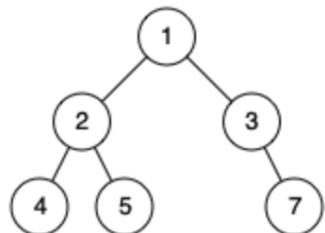


输入: [1,2,3,4,5,6]

输出: true

解释: 最后一层前的每一层都是满的（即，结点值为 {1} 和 {2,3} 的两层），且最后所有结点 ({4,5,6}) 都尽可能地向左。

示例 2：



输入: [1,2,3,4,5,null,7]

输出: false

解释: 值为 7 的结点没有尽可能靠向左侧。

```

class Solution:
    def isCompleteTree(self, root: TreeNode) -> bool:
        queue = []
        pre = root
        queue.append(root)
        while queue:
            cur = queue.pop(0)
            if pre is None and cur != None:
                return False
            if cur:
                queue.append(cur.left)
                queue.append(cur.right)
            pre = cur
        return True
  
```

103. 二叉树的锯齿形层序遍历

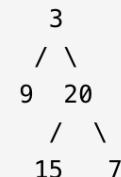
难度 中等

494 收藏 分享 切换为英文 接收动态 反馈

给定一个二叉树，返回其节点值的锯齿形层序遍历。（即先从左往右，再从右往左进行下一层此类推，层与层之间交替进行）。

例如：

给定二叉树 [3,9,20,null,null,15,7]，



返回锯齿形层序遍历如下：

```

[
  [3],
  [20,9],
  [15,7]
]
  
```

```

class Solution:
    def zigzagLevelOrder(self, root: TreeNode) -> List[List[int]]:
        if not root:
            return []
        res = []
        depth = 0
        cur_level = [root]
        while cur_level:
            tmp = []
            next_level = []
            for node in cur_level:
                tmp.append(node.val)
                if node.left:
                    next_level.append(node.left)
                if node.right:
                    next_level.append(node.right)
            if depth % 2 == 1:
                res.append(tmp[::-1])
            else:
                res.append(tmp)
            depth += 1
            cur_level = next_level
        return res

```

路径总和

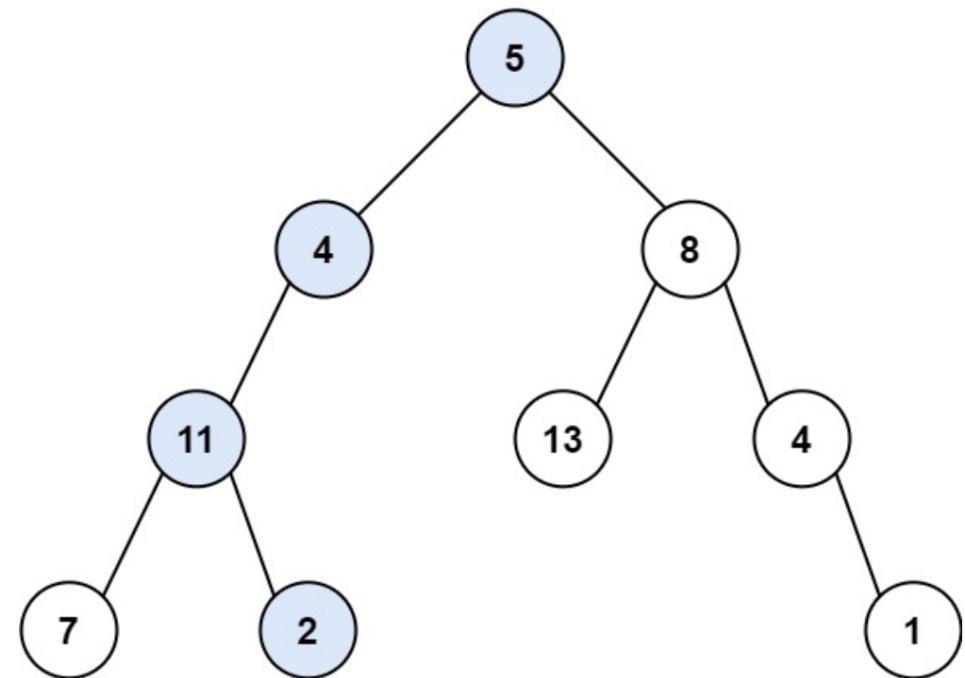
112. 路径总和

难度 简单 636 收藏 分享 切换为英文 接收动态 反馈

给你二叉树的根节点 `root` 和一个表示目标和的整数 `targetSum`，判断该树中是否存在一条路径，这条路径上所有节点值相加等于目标和 `targetSum`。

叶子节点 是指没有子节点的节点。

示例 1:



```

import collections
class Solution:
    def hasPathSum(self, root: TreeNode, sum: int) -> bool:
        if not root:
            return False
        if not root.left and not root.right:
            return sum == root.val
        return self.hasPathSum(root.left, sum - root.val) or se

    def hasPathSum1(self, root: TreeNode, sum: int) -> bool:
        if not root:
            return False
        queue = collections.deque()
        queue.append((root, root.val))
        while queue:
            node, path = queue.popleft()
            if not node.left and not node.right and path == sum:
                return True
            if node.left:
                queue.append((node.left, path + node.left.val))
            if node.right:
                queue.append((node.right, path + node.right.val))
        return False

```

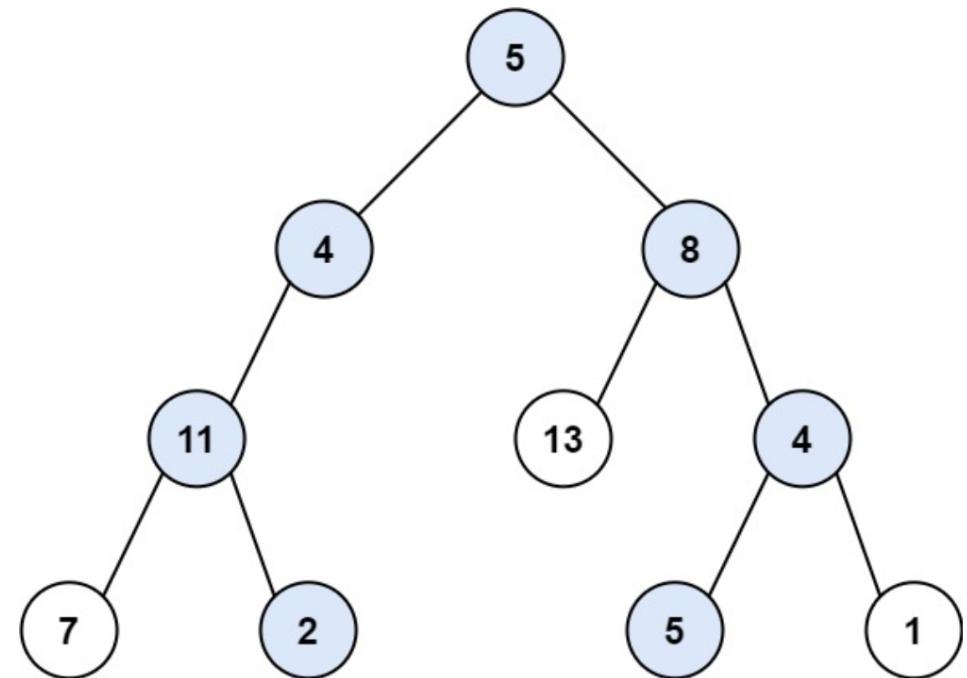
113. 路径总和 II

难度 中等 ⚡ 549 收藏 分享 切换为英文 接收动态 反馈

给你二叉树的根节点 `root` 和一个整数目标和 `targetSum`，找出所有 **从根节点到叶子节点** 等于给定目标和的路径。

叶子节点 是指没有子节点的节点。

示例 1:



```

class Solution:
    def pathSum(self, root: TreeNode, sum_: int) -> List[List[int]]:
        if not root:
            return []
        queue = collections.deque()
        queue.append((root, [root.val]))
        res = []
        while queue:
            node, path = queue.popleft()
            if not node.left and not node.right and sum(path) == sum_:
                res.append(path)
            if node.left:
                queue.append((node.left, path + [node.left.val]))
            if node.right:
                queue.append((node.right, path + [node.right.val]))
        return res

    def pathSum(self, root: TreeNode, sum_: int) -> List[List[int]]:
        def helper(root, tmp, sum_):
            if not root:
                return
            if not root.left and not root.right and sum_ - root.val == 0:
                tmp += [root.val]
                res.append(tmp)
            helper(root.left, tmp + [root.val], sum_ - root.val)
            helper(root.right, tmp + [root.val], sum_ - root.val)
        res = []
        helper(root, [], sum_)
        return res

```

二叉树的最大路径和

124. 二叉树中的最大路径和

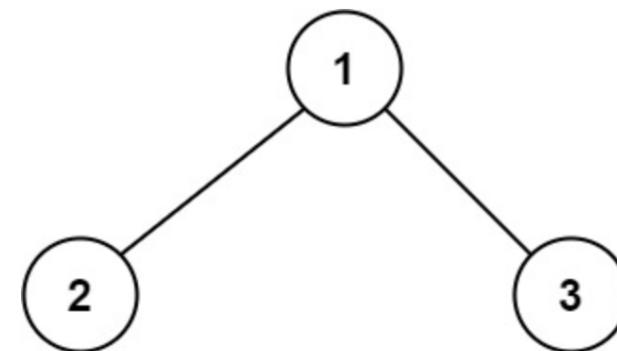
难度 困难 ⚡ 1160 ☆ 收藏 ⌂ 分享 文 切换为英文 ⌂ 接收动态 ⌂ 反馈

路径 被定义为一条从树中任意节点出发，沿父节点-子节点连接，达到任意节点的序列。同一条路径序列中 **至多出现一次**。该路径 **至少包含一个** 节点，且不一定经过根节点。

路径和 是路径中各节点值的总和。

给你一个二叉树的根节点 `root`，返回其 **最大路径和**。

示例 1：

输入: `root = [1,2,3]`

输出: 6

解释: 最优路径是 $2 \rightarrow 1 \rightarrow 3$ ，路径和为 $2 + 1 + 3 = 6$

```

class Solution:
    def maxPathSum(self, root: TreeNode) -> int:
        self.res = root.val
        def dfs(root):
            if not root:
                return 0

            left = dfs(root.left) # 左子树的最大路径和
            right = dfs(root.right) # 右子树的最大路径和

            innermax = left + root.val + right # 当前子树内部的最大路径和
            self.res = max(self.res, innermax) # 更新最大记录

            outputmax = root.val + max(left, right)
            return max(0, outputmax)

        dfs(root)
        return self.res

```

平衡二叉树

110. 平衡二叉树

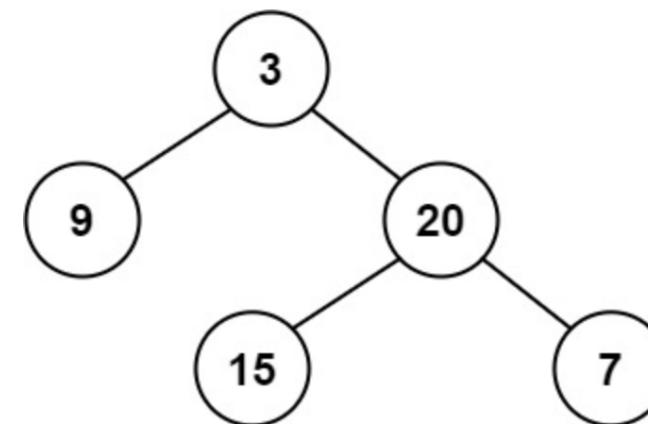
难度 简单 744 收藏 分享 切换为英文 接收动态 反馈

给定一个二叉树，判断它是否是高度平衡的二叉树。

本题中，一棵高度平衡二叉树定义为：

一个二叉树每个节点的左右两个子树的高度差的绝对值不超过 1。

示例 1：



输入: root = [3,9,20,null,null,15,7]

输出: true

```

class Solution:
    def isBalanced(self, root: TreeNode) -> bool:
        if not root:
            return True
        return abs(self.depth(root.left) - self.depth(root.right)) <= 1 and self.isBalanced(root.left) and self.isBalanced(root.right)

    def depth(self, root):
        if not root:
            return 0
        return max(self.depth(root.left), self.depth(root.right)) + 1

    def isBalanced1(self, root: TreeNode) -> bool:
        return self.recur(root) >= 0

    def recur(self, root):
        if not root:
            return 0
        left = self.recur(root.left)
        right = self.recur(root.right)
        if left == -1 or right == -1 or abs(left - right) > 1:
            return -1
        else:
            return max(left, right) + 1

```

二叉树的直径

543. 二叉树的直径

难度 简单 766 收藏 分享 切换为英文 接收动态 反饋

给定一棵二叉树，你需要计算它的直径长度。一棵二叉树的直径长度是任意两个结点路径长度值。这条路径可能穿过也可能不穿过根结点。

示例：

给定二叉树



返回 3，它的长度是路径 [4,2,1,3] 或者 [5,2,1,3]。

注意：两结点之间的路径长度是以它们之间边的数目表示。

```

class Solution:
    def diameterOfBinaryTree(self, root: TreeNode) -> int:
        self.res = 0
        def dfs(root):
            if not root:
                return 0
            left = dfs(root.left)
            right = dfs(root.right)
            self.res = max(self.res, left + right)
            return max(left, right) + 1
        dfs(root)
        return self.res

```

树的直径

1245. 树的直径

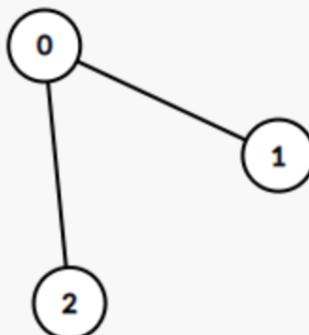
难度 中等 69 收藏 分享 切换为英文 接收动态 反馈

给你这棵「无向树」，请你测算并返回它的「直径」：这棵树上最长简单路径的边数。

我们用一个由所有「边」组成的数组 `edges` 来表示一棵无向树，其中 `edges[i] = [u, v]` 表示点 `u` 和 `v` 之间的双向边。

树上的节点都已经用 `{0, 1, ..., edges.length}` 中的数做了标记，每个节点上的标记是唯一的。

示例 1：



输入: `edges = [[0,1],[0,2]]`

输出: 2

解释:

这棵树上最长的路径是 `1 - 0 - 2`，边数为 2。

```

import collections

class Solution:
    def treeDiameter(self, edges: List[List[int]]) -> int:
        self.res = 0
        n = len(edges) + 1
        adj = collections.defaultdict(list)
        visited = dict()
        for a, b in edges:
            adj[a].append(b)
            adj[b].append(a)
            visited[a] = False
            visited[b] = False

        self.dfs(adj, 0, visited)
        return self.res

    def dfs(self, adj, index, visited):
        visited[index] = True
        max_path1, max_path2 = 0, 0
        for child in adj[index]:
            if not visited[child]:
                max_path = self.dfs(adj, child, visited)
                if max_path > max_path1:
                    max_path2 = max_path1
                    max_path1 = max_path
                elif max_path > max_path2:
                    max_path2 = max_path
        visited[index] = False
        self.res = max(self.res, max_path1 + max_path2)
        return max(max_path1, max_path2) + 1
  
```

N叉树的直径

1522. N 叉树的直径

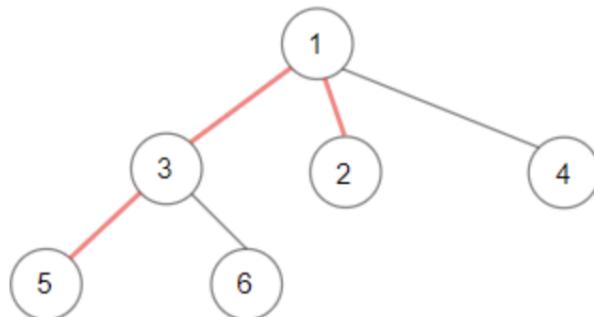
难度 中等  12  收藏  分享  切换为英文  接收动态  反馈

给定一棵 N 叉树的根节点 `root`，计算这棵树的直径长度。

N 叉树的直径指的是树中任意两个节点间路径中 **最长** 路径的长度。这条路径可能经过根节点，经过根节点。

(N 叉树的输入序列以层序遍历的形式给出，每组子节点用 `null` 分隔)

示例 1：



输入: `root = [1,null,3,2,4,null,5,6]`

输出: 3

解释: 直径如图中红线所示。

Solution:

```

class Solution:
    def diameter(self, root: 'Node') -> int:
        self.res = 0
        def dfs(root):
            if not root:
                return 0
            max_path1, max_path2 = 0, 0
            for child in root.children:
                max_path = dfs(child)
                if max_path > max_path1:
                    max_path2 = max_path1
                    max_path1 = max_path
                elif max_path > max_path2:
                    max_path2 = max_path
            self.res = max(self.res, max_path1 + max_path2)
            return max(max_path1, max_path2) + 1
        dfs(root)
        return self.res
  
```

带权树的直径

NC99 树的直径

较难 通过率: 36.15% 时间限制: 1秒 空间限制: 128M

知识点: 树

题目

题解(9)

讨论(27)

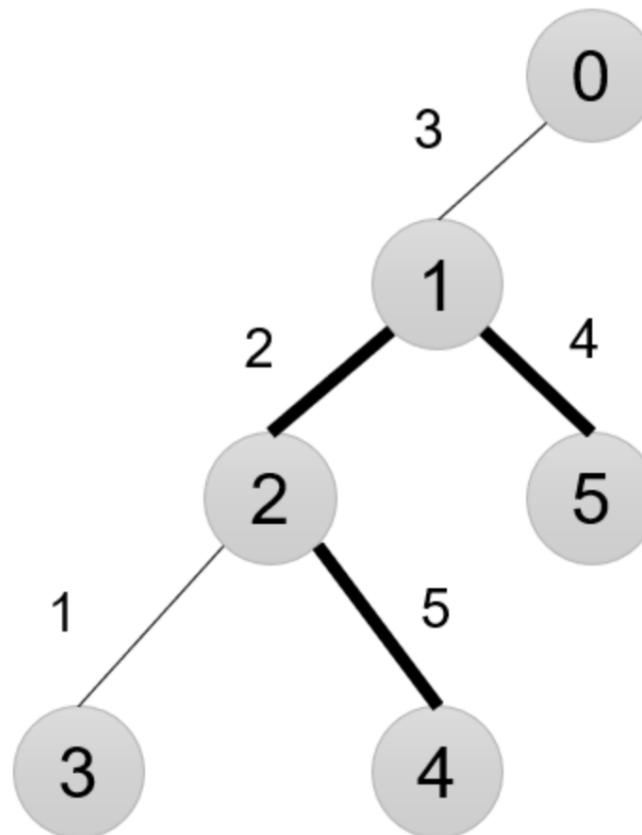
排行

描述

给定一棵树，求出这棵树的直径，即树上最远两点的距离。

包含n个结点，n-1条边的连通图称为树。

示例1的树如下图所示。其中4到5之间的路径最长，是树的直径，距离为5+2+4=11



```

# class Interval:
#     def __init__(self, a=0, b=0):
#         self.start = a
#         self.end = b
class Solution:
    def solve(self, n, Tree_edge, Edge_value):
        from collections import defaultdict
        adj = defaultdict(list)
        for i in range(n - 1):
            a = Tree_edge[i].start
            b = Tree_edge[i].end
            adj[a].append((b, Edge_value[i]))
            adj[b].append((a, Edge_value[i]))

    self.res = 0
    visited = set()

    def dfs(index):
        visited.add(index)
        max_path1, max_path2 = 0, 0
        for child, path in adj[index]:
            if child not in visited:
                max_path = dfs(child) + path
                if max_path > max_path1:
                    max_path2 = max_path1
                    max_path1 = max_path
                elif max_path > max_path2:
                    max_path2 = max_path
        visited.remove(index)
        self.res = max(self.res, max_path1 + max_path2)
        return max(max_path1, max_path2)

    dfs(Tree_edge[0].start)
    return self.res
  
```

■ 题目描述

牛牛有一棵有 n 个节点的二叉树，其根节点为 $root$ 。牛牛想修剪掉当前二叉树的叶子节点，但是牛牛不能直接删除叶子节点。他只能修剪叶子节点的父节点，修剪了父节点之后，叶子节点也会对应删掉，牛牛想在留下尽可能多的节点前提下，修剪掉所有的叶子节点。请你返回修剪后的二叉树。

有如下二叉树：



修剪过后仅会留下根节点。

示例1 输入输出示例仅供调试，后台判题数据一般不包含示例

输入

牛客@牛客网3号

```

def pruneLeaves(root):
    if not root:
        return None

    if root.left and root.left.left is None and root.left.right is None:
        return None

    if root.right and not root.right.left and not root.right.right:
        return None

    root.left = pruneLeaves(root.left)
    root.right = pruneLeaves(root.right)

    return root
  
```

108. 将有序数组转换为二叉搜索树

难度 简单 820 收藏 分享 切换为英文 接收动态 反馈

给你一个整数数组 $nums$ ，其中元素已经按 **升序** 排列，请你将其转换为一棵 **高度平衡** 二叉搜索树。

高度平衡 二叉树是一棵满足「每个节点的左右两个子树的高度差的绝对值不超过 1」的二叉树。

示例 1：

输入: $nums = [-10, -3, 0, 5, 9]$

输出: $[0, -3, 9, -10, null, 5]$

解释: $[0, -10, 5, null, -3, null, 9]$ 也将被视为正确答案:

示例 2：

输入: $nums = [1, 3]$

输出: $[3, 1]$

解释: $[1, 3]$ 和 $[3, 1]$ 都是高度平衡二叉搜索树。

```

class Solution:
    def sortedArrayToBST(self, nums: List[int]) -> TreeNode:
        if not nums:
            return None

        mid = len(nums) // 2
        root = TreeNode(nums[mid])

        left = nums[:mid]
        right = nums[mid + 1:]

        root.left = self.sortedArrayToBST(left)
        root.right = self.sortedArrayToBST(right)

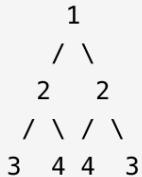
        return root
  
```

101. 对称二叉树

难度 简单 1512 收藏 分享 切换为英文 接收动态 反

给定一个二叉树，检查它是否是镜像对称的。

例如，二叉树 `[1,2,2,3,4,4,3]` 是对称的。



但是下面这个 `[1,2,2,null,3,null,3]` 则不是镜像对称的：



class Solution:

```

def isSymmetric(self, root: TreeNode) -> bool:
    if not root:
        return True

    def dfs(left, right):
        if not left and not right:
            return True
        if not left or not right:
            return False
        if left.val != right.val:
            return False
        return dfs(left.left, right.right) and dfs(left.right, right.left)

    return dfs(root.left, root.right)

```

def isSymmetric1(self, root: TreeNode) -> bool:

```

if not root:
    return True

queue = [root.left, root.right]
while queue:
    left = queue.pop(0)
    right = queue.pop(0)
    if not left and not right:
        continue
    if not left or not right:
        return False
    if left.val != right.val:
        return False
    queue.append(left.left)
    queue.append(right.right)
    queue.append(left.right)
    queue.append(right.left)
return True

```

N叉树的路径和

```
class Solution:
```

```
    def findsum(self, root, target):
        if not root:
            return []

        queue = [(root, [root.val], root.val)]
        # res = []
        while queue:
            node, path, cum = queue.pop(0)
            if cum == target:
                # res.append(path[:])
                print(path)
            if node.children:
                for child in node.children:
                    queue.append((child, path + [child.val], cum + child.val))
        # return res
```