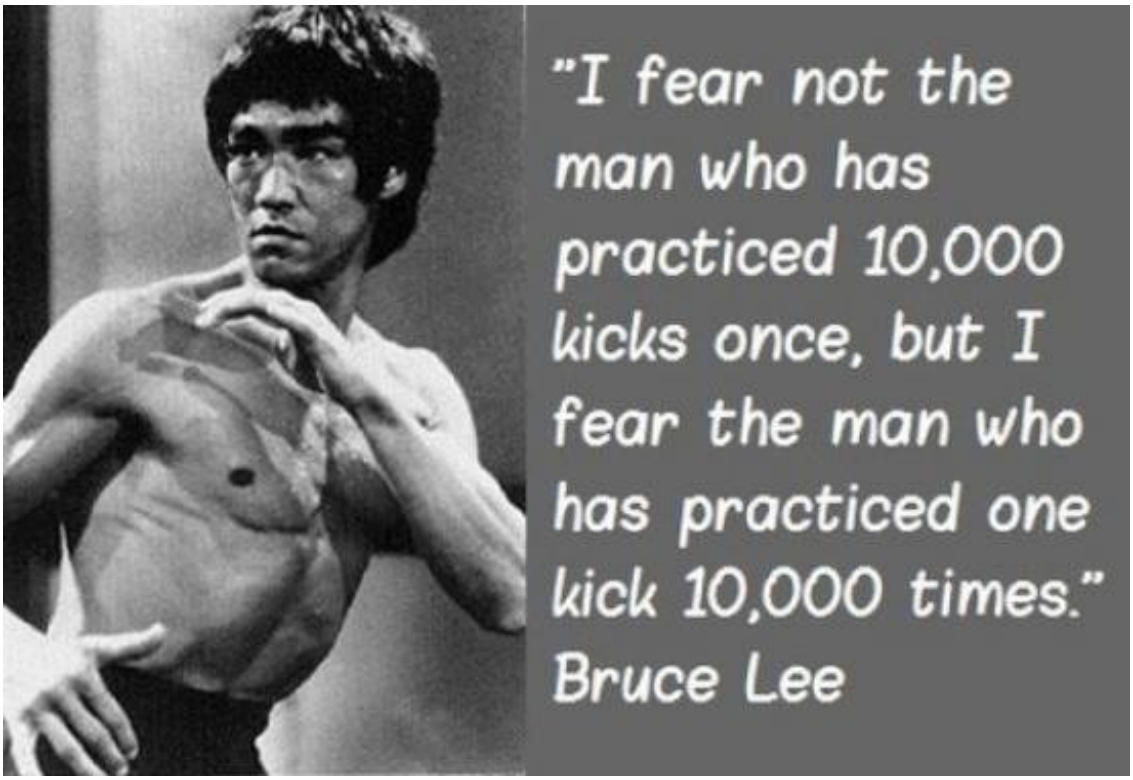
一套拳法👊刷掉n个遍历树的问题 - 相同的树 - 力扣（LeetCode）

leetcode-cn.com/problems/same-tree/solution/yi-tao-quan-fa-shua-diao-nge-bian-li-shu-de-wen--2

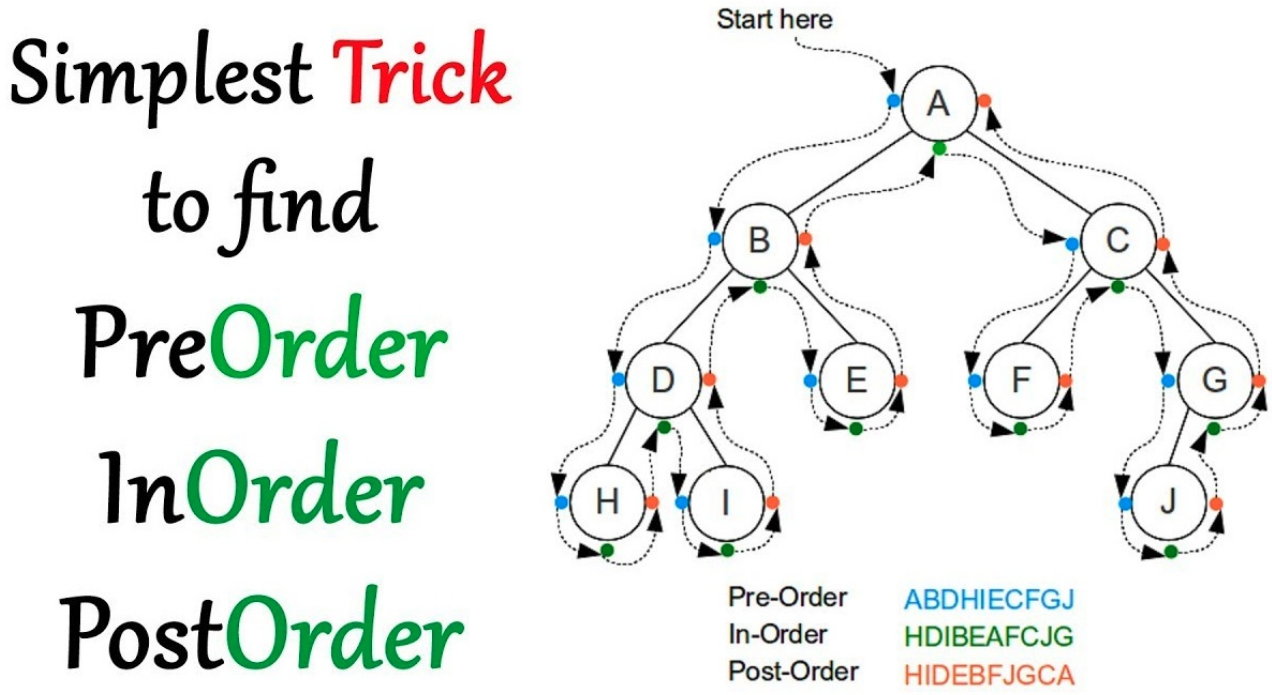
> 本文将带你用树的一种遍历算法解决N个 `leetcode` 相关算法题(算法小渣渣致敬叶师傅)

> 我不害怕曾經練過一萬種踢法的人，但我害怕一種踢法練過一萬次的人(by 叶师傅的徒弟Bruce Lee)



树的遍历(Traversal)

如下图, 三种遍历方式, 可用同一种递归思想实现



先序遍历(PreOrder, 按照先访问根节点的顺序)

```
var preorderTraversal = function(root) {
  const res = []
  function traversal (root) {
    if (root !== null) {
      res.push(root.val)
      traversal(root.left)
      traversal(root.right)
    }
  }
  traversal(root)
  return res
}
```

94 中序遍历(InOrder, 按照根节点在中间访问的顺序)

力扣 探索 题库 圈子 竞赛 力扣杯 面试 职位 商店

题目描述 评论 (460) 题解(436) 提交记录

94. 二叉树的中序遍历

难度 中等 474 收藏 分享 切换为英文 关注 反馈

给定一个二叉树，返回它的*中序* 遍历。

示例:

```
输入: [1,null,2,3]
   1
    \
     2
    /
   3

输出: [1,3,2]
```

进阶: 递归算法很简单，你可以通过迭代算法完成吗?

```
var inorderTraversal = function(root) {
  const res = []
  function traversal (root) {
    if (root !== null) {
      traversal(root.left)
      res.push(root.val)
      traversal(root.right)
    }
  }
  traversal(root)
  return res
}
```

145 后续遍历(PosterOrder, 按照根节点在后面访问的顺序)

力扣 探索 题库 圈子 竞赛 面试 职位 商店

题目描述 评论 (316) 题解(322) 提交记录

145. 二叉树的后序遍历

难度 困难 281 收藏 分享 切换为英文 关注 反馈

给定一个二叉树，返回它的 *后序* 遍历。

示例:

```
输入: [1,null,2,3]
   1
    \
     2
    /
   3

输出: [3,2,1]
```

进阶: 递归算法很简单，你可以通过迭代算法完成吗?

```
var postorderTraversal = function(root) {
  const res = []
  function traversal (root) {
    if (root !== null) {
      traversal(root.left)
      traversal(root.right)
      res.push(root.val)
    }
  }
  traversal(root)
  return res
}
```

100 相同的树

力扣 探索 题库 圈子 竞赛 面试 职位 商店

题目描述 | 评论 (629) | 题解(465) | 提交记录

100. 相同的树

难度 简单 344 收藏 分享 切换为英文 关注 反馈

给定两个二叉树，编写一个函数来检验它们是否相同。

如果两个树在结构上相同，并且节点具有相同的值，则认为它们是相同的。

示例 1:

```
输入:       1         1
          / \       / \
         2   3     2   3

        [1,2,3],   [1,2,3]

输出: true
```

示例 2:

```
输入:      1          1
          /            \
         2              2

        [1,2],     [1,null,2]

输出: false
```

可以利用这种递归思想并发同时爬两棵树

```
var isSameTree = function(p, q) {
  function traversal (root1, root2) {
    if (root1 === null && root2 !== null) {
      return false
    } else if (root1 !== null && root2 === null) {
      return false
    } else if (root1 === null && root2 === null) {
      return true
    } else {
      return  root1.val === root2.val && traversal(root1.left, root2.left) && traversal(root1.right,
root2.right)
    }
  }
  return traversal(p, q)
}
```

226 翻转二叉树

力扣 探索 题库 圈子 竞赛 力扣杯 面试 职位 商店

题目描述 | 评论 (547) | 题解(401) | 提交记录

226. 翻转二叉树

难度 简单 412 收藏 分享 切换为英文 关注 反馈

翻转一棵二叉树。

示例：

输入：

```
     4
   /   \
  2     7
 / \   / \
1   3 6   9
```

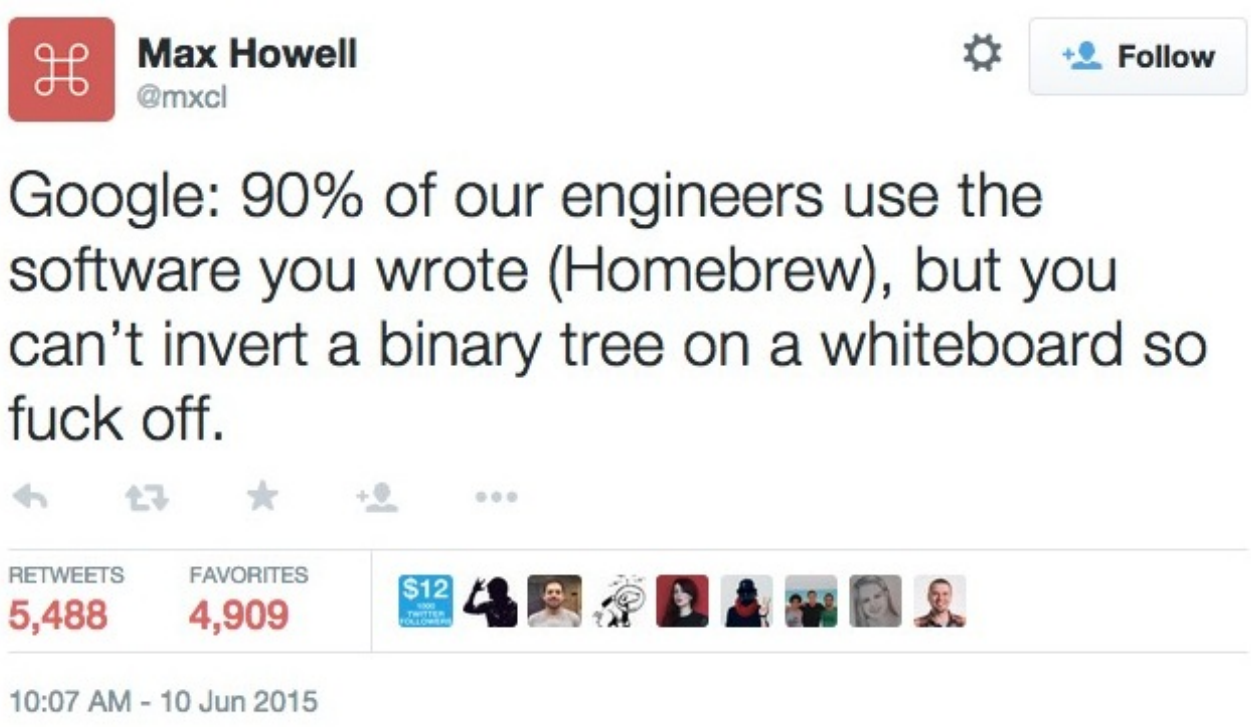
输出：

```
     4
   /   \
  7     2
 / \   / \
9   6 3   1
```

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上写出翻转二叉树这道题，这太糟糕了。

这种算法可以帮助 Homebrew 作者 Max Howell 解开 Google 的算法面试题




```
var invertTree = function(root) {
  function traversal (root) {
    if (root === null) {
      return null
    } else {
      [root.left, root.right] = [traversal(root.right), traversal(root.left)]
      return root
    }
  }
  return  traversal(root)
}
```

590 N叉树的后序遍历

力扣 探索 题库 圈子 竞赛 面试 职位 商店

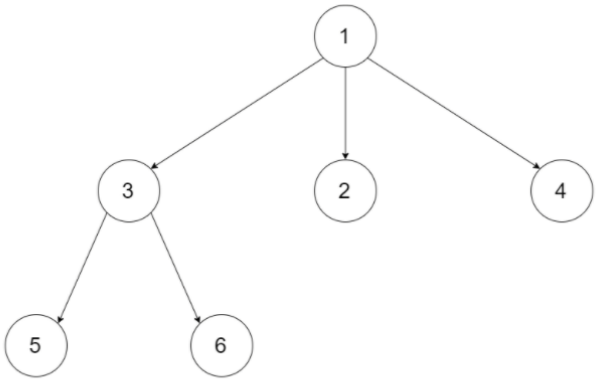
题目描述 评论 (143) 题解(111) 提交记录

590. N叉树的后序遍历

难度 简单 61 收藏 分享 切换为英文 关注 反馈

给定一个 N 叉树，返回其节点值的*后序遍历*。

例如，给定一个 `3叉树` ：



返回其后序遍历: `[5,6,3,2,4,1]`.

说明: 递归法很简单，你可以使用迭代法完成此题吗?

我们还可以用此种算法解决N叉树的问题

```
var postorder = function(root) {
  const res = []
  function traversal (root) {
    if (root !== null) {
      root.children.forEach(child => {
        traversal(child)
      })
      res.push(root.val)
    }
  }
  traversal(root)
  return res
}
```

如果你已对这种写法审美疲劳, 可以换个写法, 使用匿名函数

```
var postorder = function(root) {
  const res = []
  ;(function (root) {
    if (root !== null) {
      root.children.forEach(child => {
        arguments.callee(child)
      })
      res.push(root.val)
    }
  })(root)
  return res
}
```

还可以利用栈来迭代

```
var postorder = function(root) {
  if (root === null) {
    return []
  }
  const res = []
  const arr = [root]
  while (arr.length) {
    const cur = arr.pop()
    res.push(cur.val)
    for (let i = cur.children.length - 1; i >= 0; i--) {
      arr.push(cur.children[i])
    }
  }
  return res.reverse()
}
```

103 二叉树的锯齿形层次遍历

103. 二叉树的锯齿形层次遍历

难度 中等 182 收藏 分享 切换为英文 关注 反馈

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

例如：
给定二叉树 `[3,9,20,null,null,15,7]`，

```
    3
   / \
  9  20
    /  \
   15   7
```

返回锯齿形层次遍历如下：

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

大白话，蛇皮走位爬树

```
var zigzagLevelOrder = function(root) {
  if (root === null) {
    return []
  } else {
    let res = []
    function traversal (root, depth) {
      if (root !== null) {
        if (res[depth] === undefined) {
          res[depth] = []
        }
        res[depth].push(root.val)
        traversal(root.left, depth + 1)
        traversal(root.right, depth + 1)
      }
    }
    traversal(root, 0)
    res.forEach((item, index) => {
      if (index & 1) {
        res[index] = item.reverse()
      }
    })
    return res
  }
}
```

优化

```
var zigzagLevelOrder = function(root) {
  if (root === null) {
    return []
  } else {
    let res = []
    function traversal (root, depth) {
      if (root !== null) {
        if (res[depth] === undefined) {
          res[depth] = []
        }
        if (depth & 1) {
          res[depth].unshift(root.val)
        } else {
          res[depth].push(root.val)
        }
        traversal(root.left, depth + 1)
        traversal(root.right, depth + 1)
      }
    }
    traversal(root, 0)
    return res
  }
}
```

230 二叉搜索树中第K小的元素

230. 二叉搜索树中第K小的元素

难度 中等 188 收藏 分享 切换为英文 关注 反馈

给定一个二叉搜索树，编写一个函数 `kthSmallest` 来查找其中第 **k** 个最小的元素。

说明：
你可以假设 k 总是有效的，1 ≤ k ≤ 二叉搜索树元素个数。

示例 1:

```
输入: root = [3,1,4,null,2], k = 1
   3
  / \
 1   4
  \
   2
输出: 1
```

示例 2:

```
输入: root = [5,3,6,2,4,null,null,1], k = 3
       5
      / \
     3   6
    / \
   2   4
  /
 1
输出: 3
```

```
var kthSmallest = function (root, k) {
  let arr = []
  function traversal (node) {
    if (node !== null) {
      traversal(node.left)
      arr.push(node.val)
      traversal(node.right)
    }
  }
  traversal(root)
  return arr[k - 1]
}
```

优化, 减少遍历次数

```
var kthSmallest = function (root, k) {
  let arr = []
  function traversal(node) {
    if (node !== null && arr.length < k) {
      traversal(node.left)
      arr.push(node.val)
      traversal(node.right)
    }
  }
  traversal(root)
  return arr[k - 1]
}
```

进一步优化, 使用O(1)的额外空间

```
var kthSmallest = function (root, k) {
  let res
  let count = 0
  function traversal(node) {
    if (node !== null) {
      if (count < k) {
        traversal(node.left)
      }
      if (++count === k) {
        res = node.val
      }
      if (count < k) {
        traversal(node.right)
      }
    }
  }
  traversal(root)
  return res
}
```

102 二叉树的层序遍历

102. 二叉树的层序遍历

难度 中等 454 收藏 分享 切换为英文 关注 反馈

给你一个二叉树，请你返回其按 **层序遍历** 得到的节点值。 （即逐层地，从左到右访问所有节点）。

示例：
二叉树：`[3,9,20,null,null,15,7]`,

```
    3
   / \
  9  20
    /  \
   15   7
```

返回其层次遍历结果：

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

```
var levelOrder = function(root) {
  const res = []
  function traversal (root, depth) {
    if (root !== null) {
      if (!res[depth]) {
        res[depth] = []
      }
      traversal(root.left, depth + 1)
      res[depth].push(root.val)
      traversal(root.right, depth + 1)
    }
  }
  traversal(root, 0)
  return res
}
```

199 二叉树的右视图

基本思路: 先序遍历, 记录每一层深度下的节点的值, 并先记录左节点再记录右节点, 则最后记录的值即为该层深度的右视图看到的值

```
var rightSideView = function(root) {
  const arr = []
  function traversal (root, depth) {
    if (root) {
      if (arr[depth] === undefined) {
        arr[depth] = []
      }
      arr[depth].push(root.val)
      traversal(root.left, depth + 1)
      traversal(root.right, depth + 1)
    }
  }
  traversal(root, 0)
  const res = []
  for (let i = 0; i < arr.length; ++i) {
    res.push(arr[i][arr[i].length - 1])
  }
  return res
};
```

104 二叉树的最大深度

104. 二叉树的最大深度

难度 简单 513 收藏 分享 切换为英文 关注 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]`,

```
    3
   / \
  9  20
    /  \
   15   7
```

返回它的最大深度 3 。

```
var maxDepth = function (root) {
  let res = 0
  function traversal (root, depth) {
    if (root !== null) {
      if (depth > res) {
        res = depth
      }
      if (root.left) {
        traversal(root.left, depth + 1)
      }
      if (root.right) {
        traversal(root.right, depth + 1)
      }
    }
  }
  traversal(root, 1)
  return res
}
```

107 二叉树的层次遍历 II

107. 二叉树的层次遍历 II

难度 简单 226 收藏 分享 切换为英文 关注 反馈

给定一个二叉树，返回其节点值自底向上的层次遍历。（即按从叶子节点所在层到根节点所在的层，逐层从左向右遍历）

例如：
给定二叉树 `[3,9,20,null,null,15,7]`，

```
    3
   / \
  9  20
    /  \
   15   7
```

返回其自底向上的层次遍历为：

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

```
var levelOrderBottom = function(root) {
  if (root === null) {
    return []
  }
  let res = []
  function traversal (root, depth) {
    if (root !== null) {
      if (!res[depth]) {
        res[depth] = []
      }
      traversal(root.left, depth + 1)
      res[depth].push(root.val)
      traversal(root.right, depth + 1)
    }
  }
  traversal(root, 0)
  return res.reverse()
}
```

671 二叉树中第二小的节点

671. 二叉树中第二小的节点

难度 简单 71 收藏 分享 切换为英文 关注 反馈

给定一个非空特殊的二叉树，每个节点都是正数，并且每个节点的子节点数量只能为 `2` 或 `0`。如果一个节点有两个子节点的话，那么这个节点的值不大于它的子节点的值。

给出这样的一个二叉树，你需要输出所有节点中的**第二小的值**。如果第二小的值不存在的话，输出 -1 。

示例 1:

输入：

```
    2
   / \
  2   5
     / \
    5   7
```

输出： 5
说明： 最小的值是 2 ，第二小的值是 5 。

示例 2:

输入：

```
    2
   / \
  2   2
```

输出： -1
说明： 最小的值是 2，但是不存在第二小的值。

```
var findSecondMinimumValue = function(root) {
  let arr = []
  ;(function traversal (root) {
    if (root !== null) {
      traversal(root.left)
      arr.push(root.val)
      traversal(root.right)
    }
  })(root)
  let _arr = [...new Set(arr)].sort()
  return _arr[1] ? _arr[1] : -1
}
```

1038 从二叉搜索树到更大和树

1038. 从二叉搜索树到更大和树

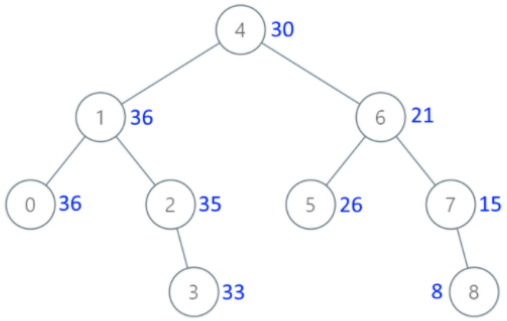
难度 中等 45 收藏 分享 切换为英文 关注 反馈

给出二叉 **搜索** 树的根节点，该二叉树的节点值各不相同，修改二叉树，使每个节点 `node` 的新值等于原树中大于或等于 `node.val` 的值之和。

提醒一下，二叉搜索树满足下列约束条件：

- 节点的左子树仅包含键 **小于** 节点键的节点。
- 节点的右子树仅包含键 **大于** 节点键的节点。
- 左右子树也必须是二叉搜索树。

示例：




```
输入：[4,1,6,0,2,5,7,null,null,null,3,null,null,null,8]
输出：[30,36,21,36,35,26,15,null,null,null,33,null,null,null,8]
```

```
var bstToGst = function(root) {
  let sum = 0
  function traversal (root) {
    if (root !== null) {
      traversal(root.right)
      root.val += sum
      sum = root.val
      traversal(root.left)
    }
  }
  traversal(root)
  return root
}
```

538 把二叉搜索树转换为累加树

力扣 探索 题库 圈子 竞赛 面试 职位 商店

题目描述 评论 (181) 题解(152) 提交记录

538. 把二叉搜索树转换为累加树

难度 简单 239 收藏 分享 切换为英文 关注 反馈

给定一个二叉搜索树（Binary Search Tree），把它转换成为累加树（Greater Tree)，使得每个节点的值是原来的节点值加上所有大于它的节点值之和。

例如：

输入： 原始二叉搜索树：

```
              5
            /   \
           2     13
```

输出： 转换为累加树：

```
             18
            /   \
          20     13
```

注意：本题和 1038: https://leetcode-cn.com/problems/binary-search-tree-to-greater-sum-tree/ 相同

```
var convertBST = function(root) {
  let sum = 0
  function traversal (root) {
    if (root !== null) {
      traversal(root.right)
      sum += root.val
      root.val = sum
      traversal(root.left)
    }
  }
  traversal(root)
  return root
}
```

700 二叉搜索树中的搜索

700. 二叉搜索树中的搜索

难度 简单 56 收藏 分享 切换为英文 关注 反馈

给定二叉搜索树（BST）的根节点和一个值。 你需要在BST中找到节点值等于给定值的节点。 返回以该节点为根的子树。 如果节点不存在，则返回 NULL。

例如，

```
给定二叉搜索树：

        4
       / \
      2   7
     / \
    1   3

和值：2
```

你应该返回如下子树:

```
      2
     / \
    1   3
```

在上述示例中，如果要找的值是 `5`，但因为没有节点值为 `5`，我们应该返回 `NULL`。

```
var searchBST = function(root, val) {
  function traversal (root) {
    if (root !== null) {
      if (root.val === val) {
        return root
      } else if (root.val < val) {
        return traversal(root.right)
      } else {
        return traversal(root.left)
      }
    } else {
      return root
    }
  }
  return traversal(root)
}
```

559 N叉树的最大深度

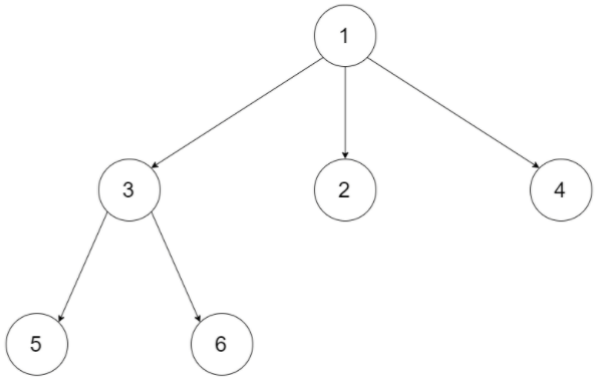
559. N叉树的最大深度

难度 简单 84 收藏 分享 切换为英文 关注 反馈

给定一个 N 叉树，找到其最大深度。

最大深度是指从根节点到最远叶子节点的最长路径上的节点总数。

例如，给定一个 `3叉树` ：



我们应返回其最大深度，3。

说明:

1. 树的深度不会超过 `1000` 。
2. 树的节点总不会超过 `5000` 。

```
var maxDepth = function(root) {
  if (root === null) {
    return 0
  } else {
    let depth = 1
    function traversal (root, curDepth) {
      if (root !== null) {
        if (curDepth > depth) {
          depth = curDepth
        }
        root.children.forEach(child => traversal(child, curDepth + 1))
      }
    }
    traversal(root, 1)
    return depth
  }
}
```

589 N叉树的前序遍历

力扣 探索 题库 圈子 竞赛 力扣杯 面试 职位 商店

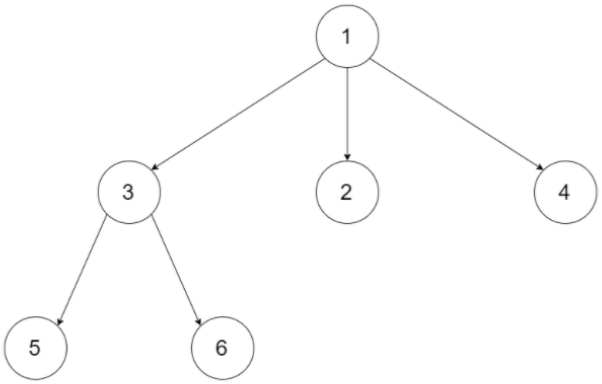
题目描述 | 评论 (160) | 题解(126) | 提交记录

589. N叉树的前序遍历

难度 简单 75 收藏 分享 切换为英文 关注 反馈

给定一个 N 叉树，返回其节点值的*前序遍历*。

例如，给定一个 `3叉树` ：



返回其前序遍历: `[1,3,5,6,2,4]` 。

说明: 递归法很简单，你可以使用迭代法完成此题吗?

```
var preorder = function(root) {
  const res = []
  function traversal (root) {
    if (root !== null) {
      res.push(root.val)
      root.children.forEach(child => traversal(child))
    }
  }
  traversal(root)
  return res
}
```

897 递增顺序查找树

897. 递增顺序查找树

难度 简单 65 收藏 分享 切换为英文 关注 反馈

给你一个树，请你 **按中序遍历** 重新排列树，使树中最左边的结点现在是树的根，并且每个结点没有左子结点，只有一个右子结点。

示例：

输入： [5,3,6,2,4,null,8,1,null,null,null,7,9]

输出： [1,null,2,null,3,null,4,null,5,null,6,null,7,null,8,null,9]

```
var increasingBST = function(root) {
  const arr = []
  function traversal (root) {
    if (root !== null) {
      traversal(root.left)
      arr.push(root.val)
      traversal(root.right)
    }
  }
  traversal(root)
  const res = new TreeNode(arr[0])
  let currentNode = res
  for (let i = 0; i < arr.length - 1; i++) {
    currentNode.left = null
    currentNode.right = new TreeNode(arr[i + 1])
    currentNode = currentNode.right
  }
  return res
}
```

原文在掘金: https://juejin.im/post/5e1c4e46f265da3e140fa54d

欢迎点赞👍、关注和来撩三连😎