【第五课】二叉树与经典问题

112. 路径总和

从根节点开始，每经过一个节点就让targetSum的值减去当前节点的值，然后再将targetSum传递给当前节点的左右子节，如果到某个叶子节点targetSum的值为0，那么就存在符合题意的路径。

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val, left, right) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.left = (left===undefined ? null : left)
 *     this.right = (right===undefined ? null : right)
 * }
 */
/**
 * @param {TreeNode} root
 * @param {number} targetSum
 * @return {boolean}
 */
var hasPathSum = function (root, targetSum) {
    if (!root) return false;
    if (!root.left && !root.right) return root.val == targetSum;
    targetSum -= root.val;
    return hasPathSum(root.left, targetSum) || hasPathSum(root.right, targetSum);
};
```

105. 从前序与中序遍历序列构造二叉树

根据前序遍历的顺序，我们可以很轻松的获取树的根节点，根据中序遍历，我们可以获取到根节点的左右子树。

可以先构建对应的根节点，然后去判断这个节点的左右子树部分，获取左右子树的根节点，再去构建。重复上述操作直到将整个树全部构建出来。

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val, left, right) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.left = (left===undefined ? null : left)
 *     this.right = (right===undefined ? null : right)
 * }
 */
/**
 * @param {number[]} preorder
 * @param {number[]} inorder
 * @return {TreeNode}
```

```
 */
var buildTree = function (preorder, inorder) {
    const map = new Map();
    for (let i = 0; i < inorder.length; i++) {
        map.set(inorder[i], i);
    }
    const helper = (pStart, pEnd, iStart, iEnd) => {
        if (pStart > pEnd) return null;
        let rootVal = preorder[pStart];
        let root = new TreeNode(rootVal);
        let mid = map.get(rootVal);
        let leftNum = mid - iStart;
        root.left = helper(pStart + 1, pStart + leftNum, iStart, mid - 1);
        root.right = helper(pStart + leftNum + 1, pEnd, mid + 1, iEnd);
        return root;
    }
    return helper(0, preorder.length - 1, 0, inorder.length - 1);
};
```

222. 完全二叉树的节点个数

我们可以自底向上的进行计数，先去找到叶子节点，然后向上每经过一个节点就加1，最后当我们到最顶层，再加上一个根节点的个数1 就是整个树的节点个数。

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val, left, right) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.left = (left===undefined ? null : left)
 *     this.right = (right===undefined ? null : right)
 * }
 */
/**
 * @param {TreeNode} root
 * @return {number}
 */
var countNodes = function (root) {
    if (!root) return 0;
    return countNodes(root.left) + countNodes(root.right) + 1;
};
```

剑指 Offer 54. 二叉搜索树的第k大节点

逆向进行中序遍历即可。转换为逆向中序遍历的第K个节点。

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
```

```
 * }
 */
/**
 * @param {TreeNode} root
 * @param {number} k
 * @return {number}
 */
var kthLargest = function(root, k) {
    if(!root) return null;
    let max = 0;
    var dfs = function(root){
        if(!root) return null;
        dfs(root.right);
        if(!--k) return max = root.val;
        dfs(root.left);
    }
    dfs(root);
    return max;
};
```

剑指 Offer 26. 树的子结构

我们需要判断A树是否包含B树，首先我们就需要现在A树中找到B树的根节点

找到B树的根节点后，再去判断B树根节点下面的子节点是否和A树中找到节点的子节点相等
直到整个B树都在A树中找到，或者有节点不同

```
var isSubStructure = function(A, B) {
    return (!!A && !!B) && (recur(A,B)||isSubStructure(A.left,B)||isSubStructure(A.right,B))
};
var recur = function(A,B){
        if(B == null) return true;
        if(A == null || A.val != B.val) return false;
        return recur(A.left, B.left) && recur(A.right, B.right);
}
```

968. 监控二叉树

根据题意我们可以推导出：

- 如果在root放置摄像头，那么root的子节点都会被监控到，所以只要保证子节点下面的子树被监控即可。
- 如果不在root放置摄像头，就要保证root两个子树被监控到的同时必须在root的一个子节点上安装摄像头，从而保证root被监控到。

我们就可以分析出三种状态：

- a.root在必须放置摄像头的情况下，覆盖整个树所需要的摄像头的数目
- b.不考虑root是否放置摄像头，覆盖整个树所需要的摄像头的数目
- c.不考虑root是否被监控到，覆盖两个子树所需的摄像头的数目

这三种状态下a ≥ b ≥ c我们假设当前节点的子节点分别对应的状态变量为la,lb,lc和ra,rb,rc

- a状态：左右子树均被监控且根节点有摄像头，所以 a = lc+rc+1
- b状态：整棵树均被监控，根节点不考虑是否有摄像头，所以 b = min(a,min(la+rb,lb+ra));
- c状态：要保证两棵子树被完全监控，要么在root放一个摄像头，要么root处不放置摄像头,此时两棵子树分别保证自己被监控，需要的摄像头数目为lb+rb。

如果root的子节点为空，我们就不能在这个子节点放置摄像头来监控root。我们就设定一个极大的整数，用于标识这种情况。
最后我们只需要求出状态b的结果就可以了。

```
var minCameraCover = function (root) {
    const dfs = (root) => {
        if (!root) return [Math.floor(Number.MAX_SAFE_INTEGER / 2), 0, 0];
        const [la, lb, lc] = dfs(root.left);
        const [ra, rb, rc] = dfs(root.right);
        let a = lc + rc + 1;
        let b = Math.min(a, Math.min(la + rb, lb + ra));
        let c = Math.min(a, lb + rb);
        return [a, b, c];
    }
    return dfs(root)[1];
};
```

662. 二叉树最大宽度

这个问题中的主要想法是给每个节点一个 index值，如果我们走向左子树，那么 index→ index* 2，如果我们走向右子树，那么 index→ index* 2 + 1。当我们在看同一层深度的位置值 L 和 R 的时候，宽度就是 R - L + 1。

宽度优先搜索顺序遍历每个节点的过程中，我们记录节点的 position 信息，对于每一个深度，第一个遇到的节点是最左边的节点，最后一个到达的节点是最右边的节点。

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val, left, right) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.left = (left===undefined ? null : left)
 *     this.right = (right===undefined ? null : right)
```

```
 * }
 */
/**
 * @param {TreeNode} root
 * @return {number}
 */
var widthOfBinaryTree = function (root) {
    if (!root) return 0;
    let max= 1n, que = [[0n, root]];
    while (que.length) {
        const width = que[que.length - 1][0] - que[0][0] + 1n;
        if (width > max) max= width;
        let temp= [];
        for (const [i, q] of que) {
            q.left && tmp.push([i * 2n, q.left]);
            q.right && tmp.push([i * 2n + 1n, q.right]);
        }
        que = temp;
    }
    return Number(max);
};
```