

二叉搜索树： 二叉搜索子树的最大键值和

困难/二叉搜索树、前序遍历、后序遍历

学习目标

拉勾教育

— 互联网人实战大学 —

掌握二叉搜索树的定义和特点

掌握二叉搜索树的基本操作的实现

掌握二叉树不同遍历方式的特点



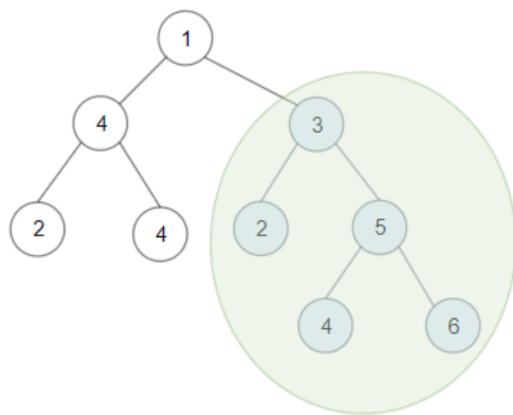
题目描述

拉勾教育

— 互联网人实战大学 —

给出一棵root为根的二叉树，请你返回任意二叉搜索子树的最大键值和

示例 1:

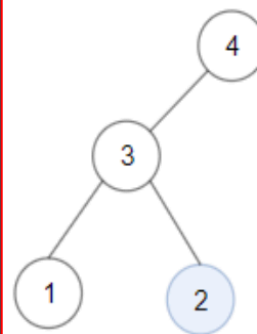


输入: root = [1,4,3,2,4,2,5,null,null,null,null,null,4,6]

输出: 20

解释: 键值为 3 的子树是和最大的二叉搜索树。

示例 2:



输入: root = [4,3,null,1,2]

输出: 2

解释: 键值为 2 的单节点子树是和最大的二叉搜索树。

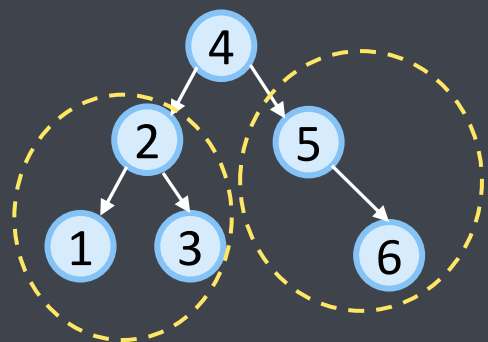
一. Comprehend 理解题意

二叉搜索树：

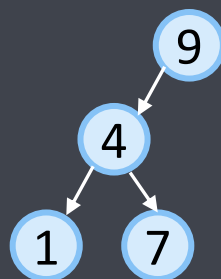
如果左子树不为空，则左子树所有结点值都小于根结点的值

如果右子树不为空，则右子树所有结点值都大于或等于根结点的值

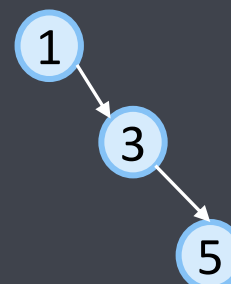
任意一棵子树也是一棵二叉搜索树



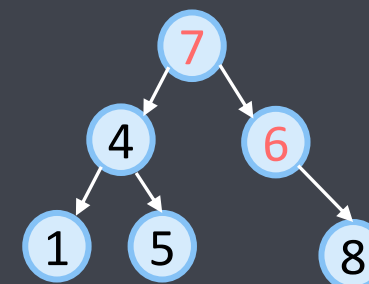
二叉搜索树



二叉搜索树



二叉搜索树



二叉树

一. Comprehend 理解题意

当你需要完成的功能是插入、删除、检索时，二叉搜索树具有极佳的性能

二叉搜索树在排序、检索、数据库管理系统及人工智能等方面广泛应用

二叉搜索树的**中序遍历序列**是一个**递增**的序列



一. Comprehend 理解题意

二叉搜索树的建立

给定一个序列，对序列的每一个元素插入到二叉搜索树：

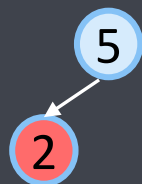
1. 如果二叉搜索树为空，则该元素即为根结点
2. 如二叉搜索树为非空，如该元素值小于根结点值，插入左子树，否则插入右子树

即可建立一棵二叉搜索树

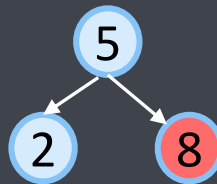
例：给定序列 $K=[5,2,8,5,9]$ 要求建立一棵二叉搜索树



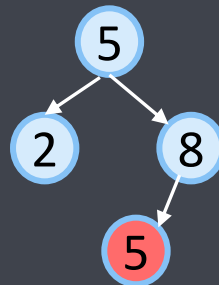
1个结点



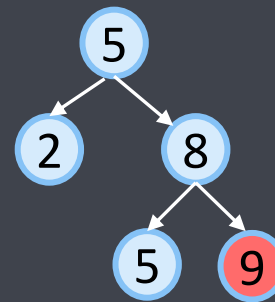
2个结点



3个结点



4个结点



5个结点

一. Comprehend 理解题意

拉勾教育

— 互联网人实战大学 —

二叉搜索树的建立递归实现



重点

```
class Solution {
    public void insert(TreeNode root, int item) {
        // 如果根结点为空，则元素为根结点
        if (root == null) {
            root = new TreeNode(item);
            return;
        } else if (item < root.val) { // 元素小于根结点值，插入左子树
            insert(root.left, item);
        } else { // 如果元素大于等于根结点值，插入右子树
            insert(root.right, item);
        }
    }
}
```

二叉搜索树插入结点的时间复杂度为 $O(h)$

一. Comprehend 理解题意

在二叉搜索树中查找给定值的元素

如果二叉搜索树为空，查找失败，返回空

如给定值与根结点比较，如相等则查找成功，若不相等

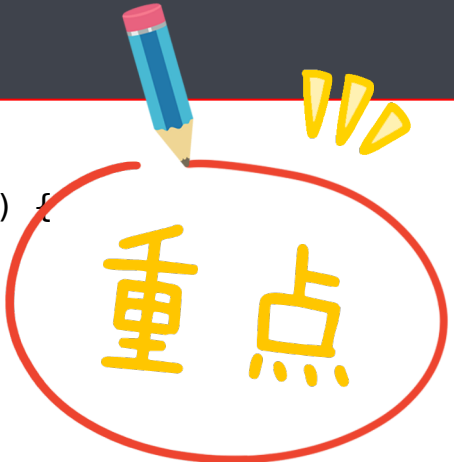
- 若给定值小于根结点，则在左子树中继续查找
- 若给定值大于等于根结点，在右子树中继续查找

请同学们根据算法实现代码，并分析时间复杂度



一. Comprehend 理解题意

在二叉搜索树的查找查找给定值的元素参考代码（递归）



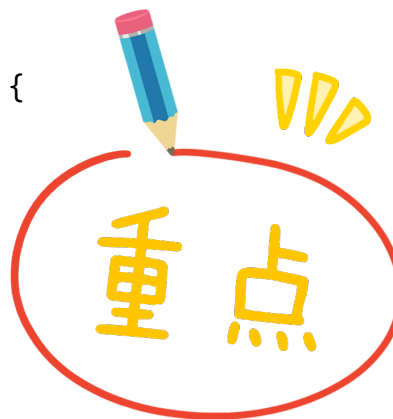
```
class Solution {  
    public boolean searchBST(TreeNode root, int value) {  
        // 递归终止条件 + 边界问题  
        if(root == null) { return false; }  
        boolean state = false;  
        // 如果查找值与当前结点相等，返回真  
        if(value == root.val) {state = true; }  
        // 如果查找值小于根结点值，在左子树中查找  
        if(value < root.val) { state = searchBST(root.left,value); }  
        // 如果查找值大于等于根结点值，在右子树中查找  
        if(value >= root.val) { state = searchBST(root.right,value); }  
        }  
        return state;  
    }  
}
```

时间复杂度为 $O(h)$ ， h 是二叉树的高度

一. Comprehend 理解题意

在二叉搜索树的查找查找给定值的元素参考代码（循环）

```
class Solution {  
    public boolean searchBST(TreeNode root, int value) {  
        while(root){  
            if(value == root.val) { return true;}  
            if(value < root.val ) { root = root.left; }  
            if(value >= root.val) { root = root.right; }  
        }  
        return false;  
    }  
}
```



时间复杂度为 $O(h)$ ， h 是二叉树的高度

一. Comprehend 理解题意

理解题意（本题要求返回二叉搜索子树的最大键值和）

1. 如何判断一棵子树是否是二叉搜索树？
2. 如何记录一棵子树的键值和？
3. 如何得到最大键值和？

细节问题

1. 对于空结点这颗子树该如何处理？
2. 选择何种遍历方式来遍历二叉树最容易得到目标值？

二. Choose 数据结构及算法思维选择

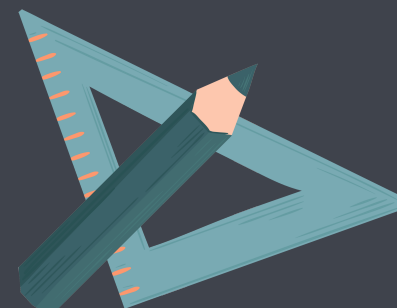
拉勾教育

— 互联网人实战大学 —

方案一：暴力搜索（暴力解法） 方案二：后序遍历（优化解法）

- 从上到下，暴力搜索
- 数据结构：二叉树，队列
- 算法思维：暴力搜索

- 从下往上，后序遍历求解
- 数组结构：二叉树
- 算法思维：后序遍历



三. Code 基本解法及编码实现

解法一: 暴力解法 (算法)

1. 从上到下，依次遍历每个结点
2. 判断以当前结点为根结点的子树是否BST，并返回这颗子树的键值和
3. 返回所有二叉搜索子树的最大键值和

请同学们根据算法实现代码，并分析时间复杂度



三. Code 基本解法及编码实现

拉勾教育

— 互联网人实战大学 —

解法一：暴力解法边界和细节问题

边界问题

- 如果树为空的时候，该如何处理

细节问题

从上往下遍历，选择前序遍历 or 层次遍历？

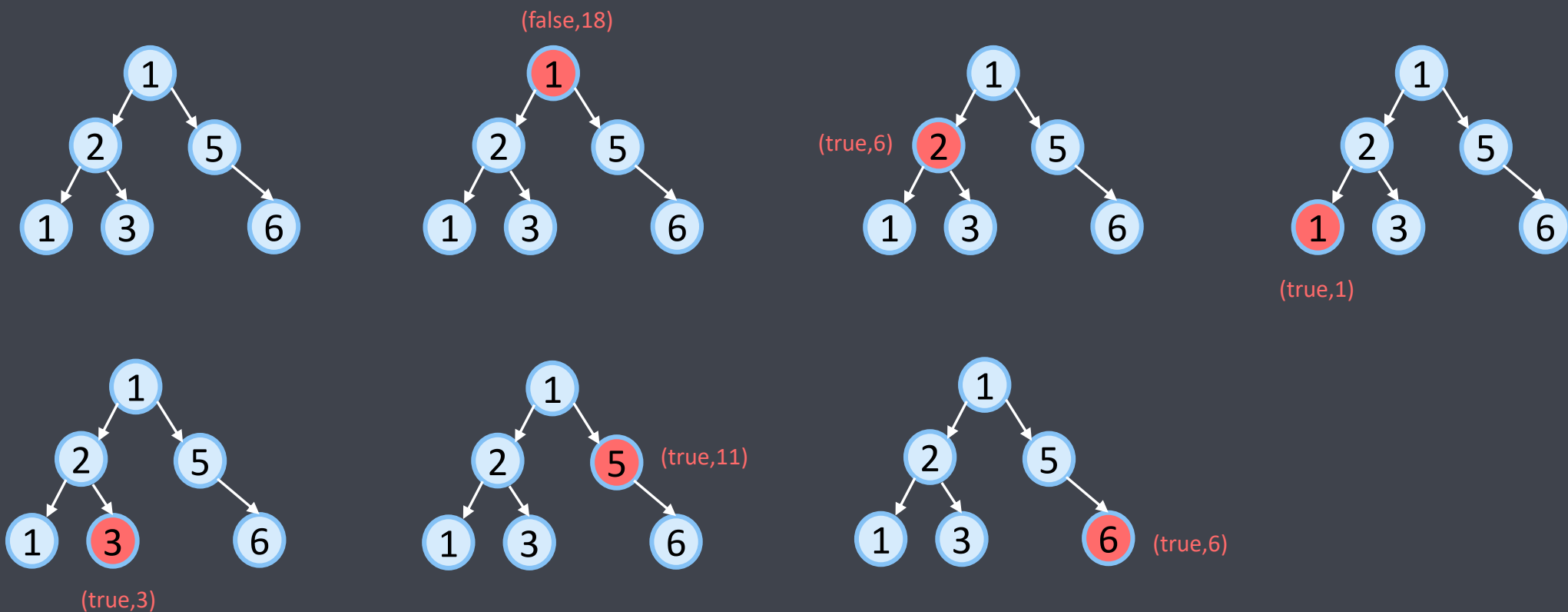
判断子树是否是BST时，如何同时返回当前子树的键值和？

在遍历过程中如何记录更新目标值？



三. Code 基本解法及编码实现

解法一：暴力解法算法演示



三. Code 基本解法及编码实现

拉勾教育

— 互联网人实战大学 —

解法一：暴力解法关键点

判断一棵二叉树是否是BST，并返回键值和

中序遍历一棵二叉树

遍历某个结点时，如果结点值 \leq 前一个结点值，则不是BST，否则将当前结点值加到键值和，

记录结点值，继续遍历

思考：函数返回值 = 是否是BST + 键值和 输入参数 = 结点 + 前一个结点值

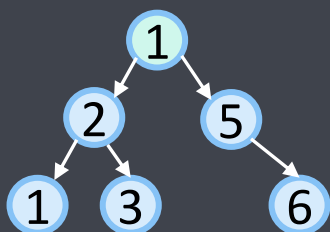
在java中如何实现

使用一个数组 `int[3]`，`int[0]=isBST`，`int[1]=preNodeValue`，`int[2]=sum`

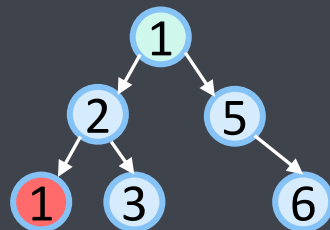
三. Code 基本解法及编码实现

解法一：暴力解法关键点

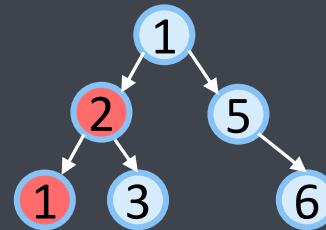
判断一棵二叉树是否是BST，并返回键值和



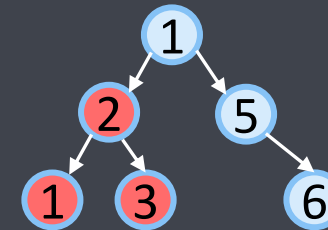
[0,MIN_VALUE,0]



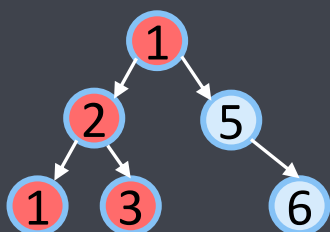
[1,MIN_VALUE,0]



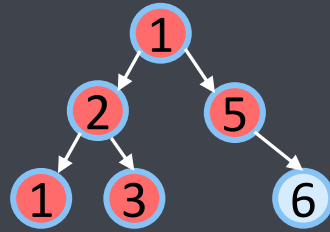
[1,1,1]



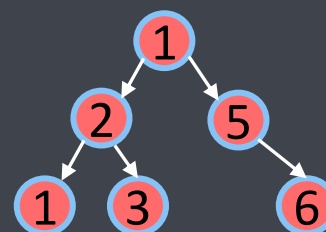
[1,2,3]



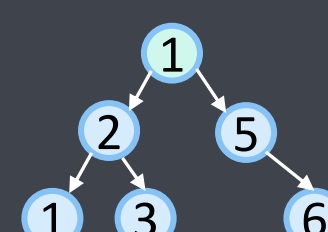
[1,3,6]



[1,1,7]



[1,5,12]



[1,6,18]

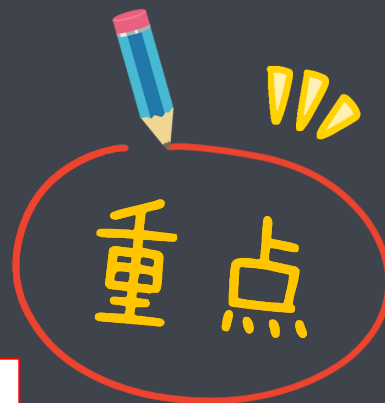
三. Code 基本解法及编码实现

解法一：暴力解法关键点

判断一棵二叉树是BST，并返回键值和参考代码

```
public int[] isValidBST(TreeNode root) {  
    int [] state=new int[3]; // 使用一个数组来记录函数返回值和状态值  
    state[0]=0; // isBST 初始值为0  
    state[1]=Integer.MIN_VALUE; // preNoeValue 初始值为整数最小值  
    state[2]=0; // 键值和 初始值为0  
    helper(root,state);  
    return state;  
}
```

```
public int isValidBST(TreeNode root) {  
    int [] state=new int[3];  
    boolean isBST = false;  
    int preNodeValue = Integer.MIN_VALUE;  
    int sum = 0  
    helper(root,isBST,preNodeValue,sum);  
    return isBST?sum:0;  
}
```



思考：

- 左边两种解法哪一种是对的？

三. Code 基本解法及编码实现

解法一: 暴力解法关键点

判断一棵二叉树是BST，并返回键值和参考代码

```
private void helper(TreeNode root,int[] state){
    if (root == null) {
        state[0]=1; // 空结点子树是BST
        return;
    }
    helper(root.left,state); // 遍历左子树
    if(root.val <= state[1] || state[0] == 0){ // 如果序列非递增 or 左子树不是BST
        state[0]=0;
        state[2]=0;
        return;
    }
    state[1]=root.val; // 遍历根结点
    state[2]=state[2]+root.val; // 将根结点值加入到键值和中
    helper(root.right,state); // 继续遍历右子树
}
```

三. Code 基本解法及编码实现

解法一：暴力解法代码实现

```
class Solution {
    int res=Integer.MIN_VALUE;
    public int maxSumBST(TreeNode root) {
        if(root == null){
            res = res>0?res:0; // 注意空子树的键值和为0
            return 0;
        }
        int[] state=isValidBST(root);// 遍历根结点
        if(state[0]==1){
            res = res > state[2] ? res : state[2];// 更新目标值
        }
        maxSumBST(root.left);// 遍历左子树
        maxSumBST(root.right);// 遍历右子树
        return res;
    }
}
```

执行结果： 超出时间限制 显示详情 >

最后执行的输入：

[8594,null,2,null,-100,null,-99,null,-98,null,-97,null,-96,null,-95,... 查看全部

已完成 执行用时：0 ms

输入 [1,4,3,2,4,2,5,null,null,null,null,null,4,6]

输出 20

预期结果 20

三. Code 基本解法及编码实现

解法一：暴力解法复杂度分析

时间复杂度： $O(n^2)$

- n 为二叉树结点数目
- 最差情况下最下层节点会被遍历 n 次，依次往上被遍历 $n-1, \dots, 1$ 次
- 最差时间复杂度是 $O(n^2)$

空间复杂度： $O(h)$

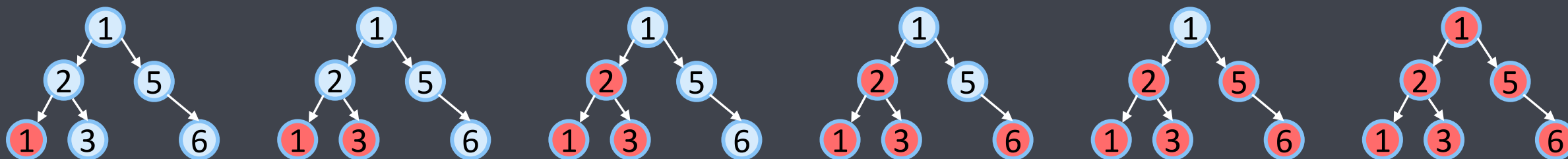
- h 是二叉树的高度
- 递归需要是用栈，栈的深度取决于二叉树的高度



四. Consider 思考更优解

思考：

- 本题关键点在于判断一棵子树是否是BST
- $BST = \text{左子树是BST} + \text{右子树是BST} + \text{左子树最大值} < \text{根结点值} + \text{右子树最小值} \geq \text{根结点值}$
- 何种遍历方式更容易解题???



采用后序遍历的方式，左 -> 右 -> 根，所有节点只需遍历一次

五. Code 最优解思路及编码实现

拉勾教育

— 互联网人实战大学 —

解法二：优化解法边界和细节问题

边界问题

- 二叉树为空的情况

细节问题

- 遍历结点的递归函数返回值是什么？
- 使用一个变量来记录最大子树的键值和，如何设置该变量的初值？



五. Code 最优解思路及编码实现

解法二：优化解法递归三要素

1. 确定函数等价关系式（参数，返回值）

- 参数是根节点+目标值，返回值 = （当前子树的键值和，最小结点值，最大结点值，是否是BST）

2. 确定结束条件

- 当前结点为空

3. 函数主功能

- 先遍历左右子树，根据其返回值与当前节点值，更新函数返回值

五. Code 最优解思路及编码实现

拉勾教育

— 互联网人实战大学 —

解法二：优化解法代码模板



// 使用一个类来同时返回多个结果

```
class Result {  
    int min;  
    int max;  
    int sum;  
    boolean isBST;  
  
    public Result(int min, int max,  
                  int sum, boolean isBST) {  
        this.min = min;  
        this.max = max;  
        this.sum = sum;  
        this.isBST = isBST;  
    }  
}
```

// 递归三要素

// 1) 确定函数等价关系式 (参数, 返回值)

```
private Result traverse(?, ?) {  
    if ( ? ) { return ? } // 2) 递归终止条件
```

Result leftResult = traverse(?, ?); // 遍历左子树

Result rightResult = traverse(?, ?); // 遍历右子树

// 3) 函数主功能

```
int val = root.val;
```

```
}
```

五. Code 最优解思路及编码实现

解法二：优化解法参考代码

```
class Solution {
    public int maxSumBST(TreeNode root) {
        int[] max = new int[1];
        traverse(root, max);
        return max[0];
    }
    private Result traverse(TreeNode root, int[] max) {
        if (root == null) { return null; } // 递归终止条件
        Result leftResult = traverse(root.left, max); // 遍历左子树
        Result rightResult = traverse(root.right, max); // 遍历右子树
        // 函数主功能
        if (leftResult != null && (!leftResult.isBST || leftResult.max >= root.val)) {
            return new Result(Integer.MIN_VALUE, Integer.MAX_VALUE, 0, false);
        }
        if (rightResult != null && (!rightResult.isBST || rightResult.min <= root.val)) {
            return new Result(Integer.MIN_VALUE, Integer.MAX_VALUE, 0, false);
        }
        int nsum = leftResult == null ? 0 : leftResult.sum;
        nsum += rightResult == null ? 0 : rightResult.sum;
        nsum += root.val;

        int minval = leftResult == null ? root.val : leftResult.min;
        int maxval = rightResult == null ? root.val : rightResult.max;
        max[0] = Math.max(max[0], nsum);
        return new Result(minval, maxval, nsum, true);
    }
}
```

五. Code 最优解思路及编码实现

解法二：优化解法复杂度分析

时间复杂度： $O(n)$

- n 是二叉树结点数目
- 每个结点只需被遍历一次

空间复杂度： $O(h)$

- h 是二叉树的高度
- 递归需要是用栈，栈的深度取决于二叉树的高度



六. Change 变形延伸

题目变形

- （练习）排序数组转成二叉搜索树

延伸扩展

- 二叉搜索树因其优良特性应用广泛
- 在面试中也是重点考察知识点
- 请大家务必掌握各种遍历方式的特点，根据题目灵活应用

本题来源：

- leetcode 1373 <https://leetcode.com/problems/maximum-sum-bst-in-binary-tree/>

总结

拉勾教育

— 互联网人实战大学 —

掌握二叉搜索树的定义和特点

掌握二叉搜索树的基本操作的实现

掌握二叉树不同遍历方式的特点



课后练习

1. 二叉搜索树转换成累加树([Leetcode 538](#) / 简单)
2. 删除二叉搜索树的结点 ([Leetcode 450](#) / 中等)
3. 二叉搜索树的编码与解码 ([Leetcode 449](#) / 中等)
4. 计算右侧小于当前元素的个数 ([Leetcode 315](#) / 困难)

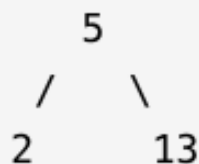


课后练习

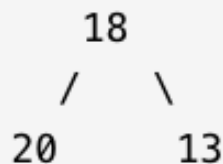
1. 二叉搜索树转换成累加树([Leetcode 538](#) / 简单)

提示：给定一棵二叉搜索树，请把它转换成为一棵累加树，使得每个结点的值是原来结点的值加上所有大于它的结点值之和

输入：原始二叉搜索树：



输出：转换为累加树：



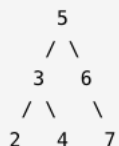
课后练习

2. 删除二叉搜索树的结点 ([Leetcode 450](#)/中等)

提示：给定一棵二叉搜索树的root和一个值key，删除二叉搜索树中key对应的结点，并保证二叉搜索树的性质不变，返回新的二叉搜索树根结点。

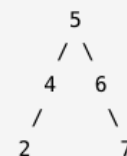
说明：要求算法的时间复杂度为 $O(h)$ ， h 是二叉搜索树的树高

```
root = [5,3,6,2,4,null,7]
key = 3
```

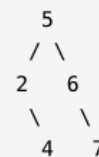


给定需要删除的节点值是 3，所以我们首先找到 3 这个节点，然后删除它。

一个正确的答案是 [5,4,6,2,null,null,7]，如下图所示。



另一个正确答案是 [5,2,6,null,4,null,7]。



课后练习

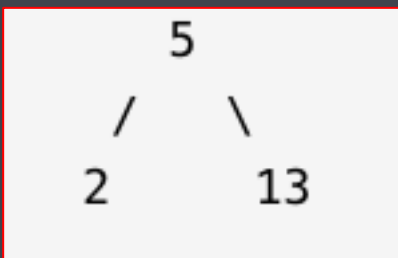
拉勾教育

— 互联网人实战大学 —

3. 二叉搜索树的编码与解码 ([Leetcode 449](#) / 中等)

提示：请设计一个算法来序列化和反序列化二叉搜索树。保证BST可以序列化为字符串，并且可以将该字符串反序列化为最初的BST。

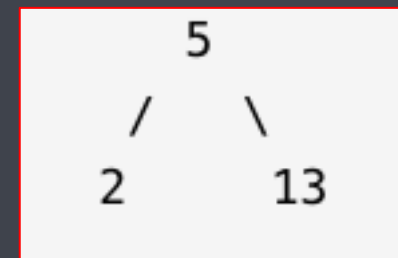
说明：BST序列化方式可以自行决定



输入

前序遍历序列：5,2,13

序列化后字符串



反序列化

4. 计算右侧小于当前元素的个数 ([Leetcode 315](#)/困难)

提示：给定一个整数数组nums，按要求返回一个新的数组counts。数组counts有如下性质：
counts[i]的值是nums[i]右侧小于nums[i]的元素个数

输入：nums = [5,2,6,1]

输出：[2,1,1,0]

解释：

5 的右侧有 2 个更小的元素 (2 和 1)

2 的右侧仅有 1 个更小的元素 (1)

6 的右侧有 1 个更小的元素 (1)

1 的右侧有 0 个更小的元素

拉勾教育

— 互联网人实战大学 —



下载「拉勾教育App」
获取更多内容