

东哥带你刷二叉搜索树（特性篇）

 Stars 107k  B站 @labuladong 配套PDF和插件 下载 打卡挑战 报名 精品课程 查看






微信搜一搜

Q labuladong公众号

通知：数据结构精品课持续更新中，[详情见这里](#)。

读完本文，你不仅学会了算法套路，还可以顺便解决如下题目：

| 牛客 | LeetCode | 力扣 | 难度 |
|----|--|--------------------|---|
| - | 230. Kth Smallest Element in a BST | 230. 二叉搜索树中第K小的元素 |  |
| - | 538. Convert BST to Greater Tree | 538. 把二叉搜索树转换为累加树 |  |
| - | 1038. Binary Search Tree to Greater Sum Tree | 1038. 把二叉搜索树转换为累加树 |  |

PS：刷题插件集成了手把手刷二叉树功能，按照公式和套路讲解了 150 道二叉树题目，可手把手带你刷完二叉树分类的题目，迅速掌握递归思维。

前文手把手带你刷二叉树已经写了 [第一期](#)，[第二期](#) 和 [第三期](#)，今天写一篇二叉搜索树（Binary Search Tree，后文简写 BST）相关的文章，手把手带你刷 BST。

首先，BST 的特性大家应该都很熟悉了：

1、对于 BST 的每一个节点 `node`，左子树节点的值都比 `node` 的值要小，右子树节点的值都比 `node` 的值大。

2、对于 BST 的每一个节点 `node`，它的左侧子树和右侧子树都是 BST。

二叉搜索树并不复杂，但我觉得它可以算是数据结构领域的半壁江山，直接基于 BST 的数据结构有 AVL 树，红黑树等等，拥有了自平衡性质，可以提供 $\log N$ 级别的增删查改效率；还有 B+ 树，线段树等结构都是基于 BST 的思想来设计的。

从做算法题的角度来看 BST，除了它的定义，还有一个重要的性质：BST 的中序遍历结果是有序的（升序）。

也就是说，如果输入一棵 BST，以下代码可以将 BST 中每个节点的值升序打印出来：

```
void traverse(TreeNode root) {  
    if (root == null) return;  
    traverse(root.left);  
    // 中序遍历代码位置  
    print(root.val);  
    traverse(root.right);  
}
```

那么根据这个性质，我们来做两道算法题。

寻找第 K 小的元素

这是力扣第 230 题「[二叉搜索树中第 K 小的元素](#)」，看下题目：

230. 二叉搜索树中第K小的元素

labuladong 题解

思路

难度 中等

👍 302

☆ 收藏

🔗 分享

🌐 切换为英文

🔔 接收动态

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。

你可以假设 `k` 总是有效的， $1 \leq k \leq$ 二叉搜索树元素个数。

示例：

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

```
      /\n     3 6\n    /\n   2  4\n  /\n 1\n输出: 3
```

这个需求很常见吧，一个直接的思路就是升序排序，然后找第 `k` 个元素呗。BST 的中序遍历其实就是升序排序的结果，找第 `k` 个元素肯定不是什么难事。

按照这个思路，可以直接写出代码：

```
int kthSmallest(TreeNode root, int k) {  
    // 利用 BST 的中序遍历特性  
    traverse(root, k);  
    return res;  
}  
  
// 记录结果  
int res = 0;  
// 记录当前元素的排名  
int rank = 0;  
void traverse(TreeNode root, int k) {  
    if (root == null) {  
        return;  
    }  
    traverse(root.left, k);  
    /* 中序遍历代码位置 */  
    rank++;  
    if (k == rank) {  
        // 找到第 k 小的元素  
        res = root.val;  
        return;  
    }  
    /******/  
    traverse(root.right, k);  
}
```

这道题就做完了，不过呢，还是要多说几句，因为这个解法并不是最高效的解法，而是仅仅适用于这道题。

我们后文 [高效计算数据流的中位数](#) 中就提过今天的这个问题：

如果让你实现一个在二叉搜索树中通过排名计算对应元素的方法 `select(int k)`，你会怎么设计？

如果按照我们刚才说的方法，利用「BST 中序遍历就是升序排序结果」这个性质，每次寻找第 `k` 小的元素都要中序遍历一次，最坏的时间复杂度是 $O(N)$ ，`N` 是 BST 的节点个数。

要知道 BST 性质是非常牛逼的，像红黑树这种改良的自平衡 BST，增删查改都是 $O(\log N)$ 的复杂度，让你算一个第 `k` 小元素，时间复杂度竟然要 $O(N)$ ，有点低效了。

所以说，计算第 `k` 小元素，最好的算法肯定也是对数级别的复杂度，不过这个依赖于 BST 节点记录的信息有多少。

我们想一下 BST 的操作为什么这么高效？就拿搜索某一个元素来说，BST 能够在对数时间找到该元素的根本原因还是在 BST 的定义里，左子树小右子树大嘛，所以每个节点都可以通过对比自身的值判断去左子树还是右子树搜索目标值，从而避免了全树遍历，达到对数级复杂度。

那么回到这个问题，想找到第 `k` 小的元素，或者说找到排名为 `k` 的元素，如果想达到对数级复杂度，关键也在于每个节点得知道自己排第几。

比如说你让我查找排名为 `k` 的元素，当前节点知道自己排名第 `m`，那么我可以比较 `m` 和 `k` 的大小：

- 1、如果 `m == k`，显然就是找到了第 `k` 个元素，返回当前节点就行了。
- 2、如果 `k < m`，那说明排名第 `k` 的元素在左子树，所以可以去左子树搜索第 `k` 个元素。
- 3、如果 `k > m`，那说明排名第 `k` 的元素在右子树，所以可以去右子树搜索第 `k - m - 1` 个元素。

这样就可以将时间复杂度降到 $O(\log N)$ 了。

那么，如何让每一个节点知道自己的排名呢？

这就是我们之前说的，需要在二叉树节点中维护额外信息。**每个节点需要记录，以自己为根的这棵二叉树有多少个节点。**

也就是说，我们 `TreeNode` 中的字段应该如下：

```
class TreeNode {
    int val;
    // 以该节点为根的树的节点总数
    int size;
    TreeNode left;
    TreeNode right;
}
```

有了 `size` 字段，外加 BST 节点左小右大的性质，对于每个节点 `node` 就可以通过 `node.left` 推导出 `node` 的排名，从而做到我们刚才说到的对数级算法。

当然，`size` 字段需要在增删元素的时候需要被正确维护，力扣提供的 `TreeNode` 是没有 `size` 这个字段的，所以我们这道题就只能利用 BST 中序遍历的特性实现了，但是我们上面说到的优化思路是 BST 的常见操作，还是有必要理解的。

BST 转化累加树

力扣第 538 题和 1038 题都是这道题，完全一样，你可以把它们一块做掉。看下题目：

538. 把二叉搜索树转换为累加树

labuladong 题解

思路

难度 中等

👍 420

☆ 收藏

🔗 分享

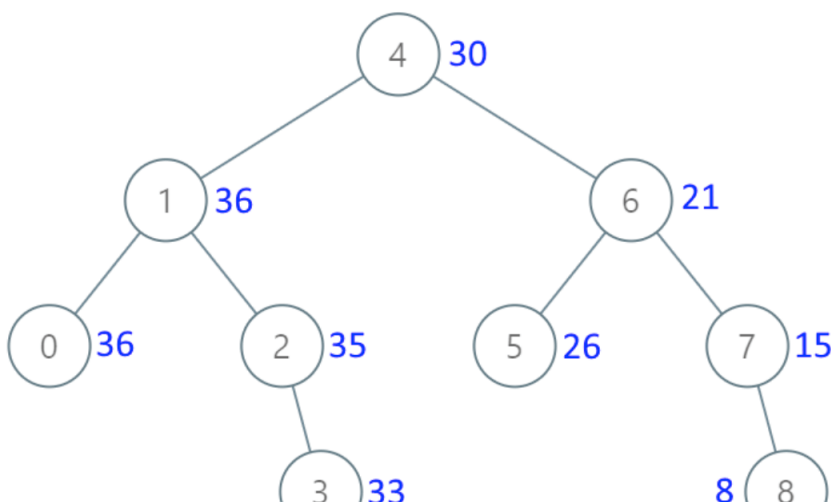
🌐 切换为英文

🔔 接收动态

💡 反馈

给出二叉搜索树的根节点，该树的节点值各不相同，请你将其转换为累加树（Greater Sum Tree），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。

示例 1：



题目应该不难理解，比如图中的节点 5，转化成累加树的话，比 5 大的节点有 6, 7, 8，加上 5 本身，所以累加树上这个节点的值应该是 $5+6+7+8=26$ 。

我们需要把 BST 转化成累加树，函数签名如下：

```
TreeNode convertBST(TreeNode root)
```

按照二叉树的通用思路，需要思考每个节点应该做什么，但是这道题上很难想到什么思路。

BST 的每个节点左小右大，这似乎是一个有用的信息，既然累加和是计算大于等于当前值的所有元素之和，那么每个节点都去计算右子树的和，不就行了吗？

这是不行的。对于一个节点来说，确实右子树都是比它大的元素，但问题是它的父节点也可能是比它大的元素呀？这个没法确定的，我们又没有触达父节点的指针，所以二叉树的通用思路在这里用不了。

其实，正确的解法很简单，还是利用 BST 的中序遍历特性。

刚才我们说了 BST 的中序遍历代码可以升序打印节点的值：

```
void traverse(TreeNode root) {  
    if (root == null) return;  
    traverse(root.left);  
    // 中序遍历代码位置  
    print(root.val);  
    traverse(root.right);  
}
```

那如果我想降序打印节点的值怎么办？

很简单，只要把递归顺序改一下就行了：

```

void traverse(TreeNode root) {
    if (root == null) return;
    // 先递归遍历右子树
    traverse(root.right);
    // 中序遍历代码位置
    print(root.val);
    // 后递归遍历左子树
    traverse(root.left);
}

```

这段代码可以降序打印 BST 节点的值，如果维护一个外部累加变量 `sum`，然后把 `sum` 赋值给 BST 中的每一个节点，不就将 BST 转化成累加树了吗？

看下代码就明白了：

```

TreeNode convertBST(TreeNode root) {
    traverse(root);
    return root;
}

// 记录累加和
int sum = 0;
void traverse(TreeNode root) {
    if (root == null) {
        return;
    }
    traverse(root.right);
    // 维护累加和
    sum += root.val;
    // 将 BST 转化成累加树
    root.val = sum;
    traverse(root.left);
}

```

这道题就解决了，核心还是 BST 的中序遍历特性，只不过我们修改了递归顺序，降序遍历 BST 的元素值，从而契合题目累加树的要求。

简单总结下吧，BST 相关的问题，要么利用 BST 左小右大的特性提升算法效率，要么利用中序遍历的特性满足题目的要求，也就这么些事儿吧。

最后调查下，经过这几篇二叉树相关的系列文章，大家刷题有没有点感觉了？可以留言和我交流。
本文对你有帮助的话，请三连~

► 引用本文的文章

《labuladong 的算法小抄》已经出版，关注公众号查看详情；后台回复关键词「进群」可加入算法群；回复「PDF」可获取精华文章 PDF：



微信搜一搜

Q labuladong 公众号

共同维护高质量学习环境，评论礼仪[见这里](#)，违者直接拉黑不解释

6 Comments - powered by [utteranc.es](#)

JingweiZuo commented on Mar 15, 2022

如何让每一个节点知道自己的排名？每个节点需要记录，以自己为根的这棵二叉树有多少个节点。有了 size 字段，外加 BST 节点左小右大的性质，对于每个节点 node 就可以通过 node.left 推导出 node 的排名

这一段表述不太理解，知道以自己为根的这棵二叉树有多少个节点后 对计算此节点的排名有什么帮助？

labuladong commented on Mar 15, 2022

Owner

BST 左小右大，所以节点 x 的左子树的节点个数就是 x 的排名

👍 5

zxc948406613 commented 3 months ago

那算第k小的节点的时候，每次对比当前节点的时候，都得去计算出左子树节点的个数和总子树个数么？