互联网经典算法面试题-验证二叉搜索树

Github喵 3天前

以下文章来源于程序员小熊，作者Dine

程序员小熊

华为程序员，公众号采用“动图”的方式剖析各大厂笔试和面试中的高频算法题，公众号长期分享各种编程语言、数据结构与算法和后台开发...

前言

大家好，我是来自于**华为**的**程序员小熊**。今天给大家带来一道与**二叉树**相关的面试高频题，这道题在半年内被谷歌、字节、微软和亚马逊等大厂作为面试题，即力扣上的第98题-验证二叉搜索树。

本文主要介绍**递归**和**深度优先搜索**两种方法来解答此题，供大家参考，希望对大家有所帮助。

验证二叉搜索树

给你一个二叉树的根节点 root ，判断其是否是一个有效的二叉搜索树。

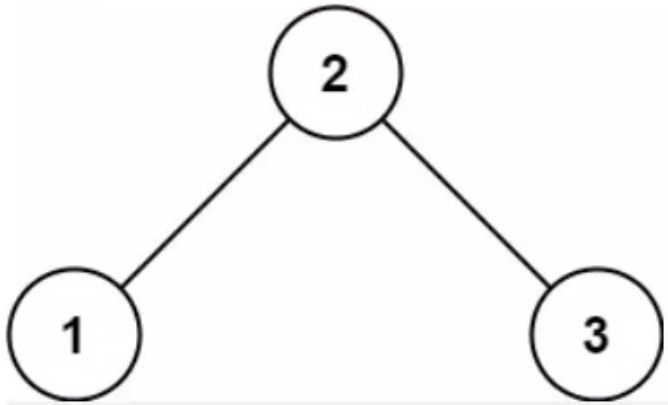
有效二叉搜索树定义如下：

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：

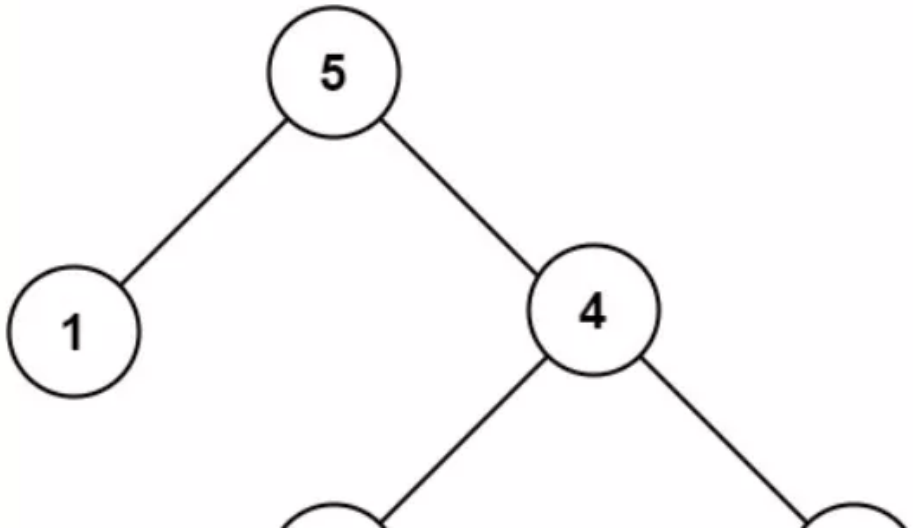



```
输入：root = [2,1,3]
输出：true
```

程序员小熊

示例 1

示例 2：



输入： root = [5,1,4,null,null,3,6]
输出： false
解释： 根节点的值是 5 ，但是右子节点的值是 4 。

程序员小熊

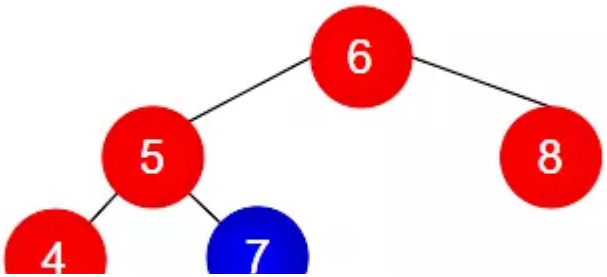
示例 2 及提示

二叉搜索树

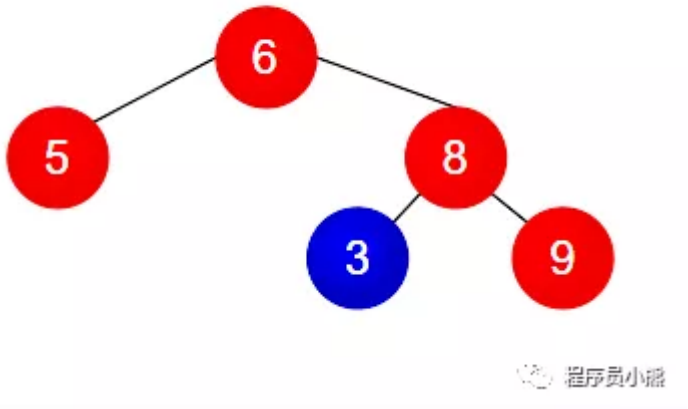
题目已提示**有效**二叉搜索树的定义如下：

- 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。

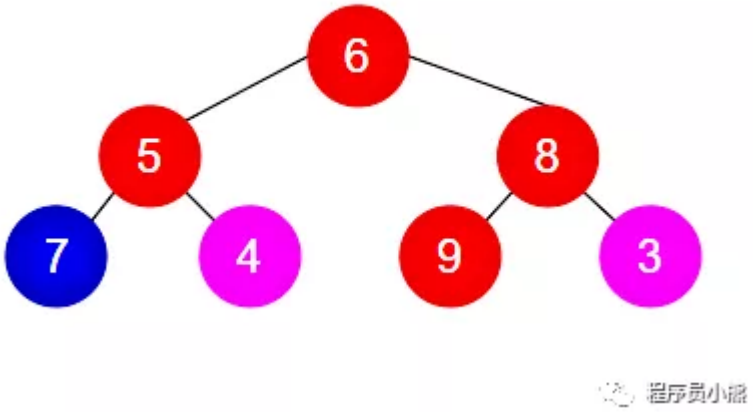
举例



例 1



例 2



例 3

判断二叉搜索树

针对上面的**举例**，根据**二叉搜索树**的判断方法，对上面的例子是否是**二叉搜索树**进行如下判断：

1. 例 1 **不是** 二叉搜索树。原因：根节点（值为 6）的左子树中有节点（值为 7）的数大于根节点的数。

2. 例 2 **不是** 二叉搜索树。原因：根节点（值为 6）的右子树中有节点（值为 3）的数小于根节点的数。

3. 例 3 **不是** 二叉搜索树。原因：根节点的左子树不是二叉搜索树，左子树的根节点的值 5 不仅小于左子节点的值 7 还大于右子节点的值 4，并且根节点的值 6 小于左子树中节点的值 7；根节点的右子树也不是二叉搜索树，右子树的根节点的值 8 不仅大于右子节点的值 3 还小于左子节点的值 9，并且根节点的值 6 大于右子树中节点的值 3。

解题思路

根据**二叉搜索树**的定义，判断一棵树是否是二叉搜索树，需要判断**每个节点**是否符合二叉树的性质，而且**判断的依据又是一样**的，因此可采用**递归法**去解答此题。

递归

上述提到的**判断的依据**（假设当前节点存在左右子节点）是指：

1. 当前节点的值**大于其左子节点**的值；

2. 当前节点的值**小于其右子节点**的值；

3. 如果当前节点存在左右子树，则其左右子树上的节点还要满足：**左子树**上的节点值**小于当前节点**的值，**右子树**上的节点值**大于当前节点**的值；

根据以上的思路，可以通过设置**上下界**，来判断节点是否符合**二叉搜索树**的性质。

如果存在上下界，则判断节点是否在上下界内，如**不在**，则**不是**二叉搜索树；否则以该节点的值作为**上界**，对其左子树进行**递归**判断，以该节点的值作为**下界**，对其右子树进行**递归**判断。

注意

空树属于二叉搜索树。

Show me the Code

C

```
bool isValidBST_Helper(struct TreeNode* root, double min, double max) {
    /* 特殊判断 */
    if (root == NULL) {
        return true;
    }

    /* 当前节点不在上下界内，不是二叉搜索树 */
    if (root->val <= min || root->val >= max) {
        return false;
    }
```

```
    /* 判断左右子树是否是二叉搜索树 */
    return isValidBST_Helper(root->left, min, root->val) && isValidBST_Helper(root->right, root->val, max);
}

bool isValidBST(struct TreeNode* root) {
    return isValidBST_Helper(root, LONG_MIN, LONG_MAX);
}
```

C++

```
bool isValidBST_Helper(TreeNode* root, double min, double max) {
    if (root == nullptr) {
        return true;
    }

    if (root->val <= min || root->val >= max) {
        return false;
    }

    return isValidBST_Helper(root->left, min, root->val) && isValidBST_Helper(root->right, root->val, max);
}

bool isValidBST(TreeNode* root) {
    return isValidBST_Helper(root, LONG_MIN, LONG_MAX);
}
```

Java

```
boolean isValidBST_Helper(TreeNode root, double min, double max) {
```

```
    if (root == null) {
        return true;
    }

    if (root.val <= min || root.val >= max) {
        return false;
    }

    return isValidBST_Helper(root.left, min, root.val) && isValidBST_Helper(root.right, root.val, max);
}

boolean isValidBST(TreeNode root) {
    return isValidBST_Helper(root, Long.MIN_VALUE, Long.MAX_VALUE);
}
```

Python3

```
def isValidBST(self, root: TreeNode) -> bool:
    def isValidBST_Helper(root, min, right):
        if root is None:
            return True

        if root.val <= min or root.val >= right:
            return False

        return isValidBST_Helper(root.left, min, root.val) and isValidBST_Helper(root.right, root.val, right)

    return isValidBST_Helper(root, -float('inf'), float('inf'))
```

Golang

```
func isValidBST(root *TreeNode) bool {
  return isValidBST_Helper(root, math.MinInt64, math.MaxInt64)
}

func isValidBST_Helper(root *TreeNode, min, max int) bool {
  if root == nil {
    return true
  }

  if min >= root.Val || max <= root.Val {
    return false
  }

  return isValidBST_Helper(root.Left, min, root.Val) && isValidBST_Helper(root.Right, root.Val, max)
}
```

复杂度分析

时间复杂度：**O(n)**，其中 n 为二叉树节点的个数。

空间复杂度：**O(n)**。

深度优先搜索

根据**二叉搜索树**的性质，对其进行**中序遍历**，得到的数组一定是**升序排列**的。因此可以根据这个特性，判断一棵树是否是二叉搜索树。

如果采用中序遍历，将二叉树的所有节点的值存放在数组中，再去判断该数组是否是**升序的**，步骤有点繁琐。

由于判断数组是否是升序排列，只需要判断数组的**后一个元素是否大于前一个元素**即可，因此本题可以设置一个**变量**，用于保存**中序遍历前一个节点的值**，再判断**当前节点的值**是否**大于**该变量**保存的值**。

如果**不大于**，则代表该树**不是**二叉搜索树；否则继续遍历并判断。

Show me the Code

C++

```
long pre = LONG_MIN;
bool isValidBST(TreeNode* root) {
    if (root == nullptr) {
        return true;
    }

    if (!isValidBST(root->left)) {
        return false;
    }

    if (root->val <= pre) {
        return false;
    }

    pre = root->val;
```

```
    return isValidBST(root->right);
}
```

Java

```
long temp = Long.MIN_VALUE;
boolean isValidBST(TreeNode root) {
    if (root == null) {
        return true;
    }

    if(!isValidBST(root.left)) {
        return false;
    }

    if (root.val <= temp) {
        return false;
    }

    temp = root.val;
    return isValidBST(root.right);
}
```

复杂度分析

时间复杂度：**O(n)**，其中 n 为二叉树节点的个数。

空间复杂度：**O(n)**。

今天，你学会了了吗？

往期精彩内容：

看一遍就理解：MVCC原理详解

2021 编程语言排行榜

源码面前没有秘密，推荐 9 个带你阅读源码的开源项目

假如苹果没有发明 iPhone

...

喜欢此内容的人还喜欢

网易面试官：MySQL的自增 ID 用完了，怎么办?

面试专栏

ElasticSearch 面试 4 连问，你顶得住么?

面试专栏

面试官：聊聊生产者和消费者模式的工作原理

面试专栏