手把手带你刷通二叉搜索树（第三期）

Original labuladong labuladong 2021-01-14 16:30

收录于合集
#手把手刷数据结构 33 #二叉搜索树 3

后台回复进群一起刷力扣😁

点击卡片可搜索关键词👇

labuladong推荐搜索

二叉树 | 动态规划 | 回溯算法 | Linux

读完本文，可以去力扣解决如下题目：

96.不同的二叉搜索树（Easy）

95.不同的二叉搜索树II（Medium）

之前写了两篇手把手刷 BST 算法题的文章，第一篇 讲了中序遍历对 BST 的重要意义，第二篇 写了 BST 的基本操作。

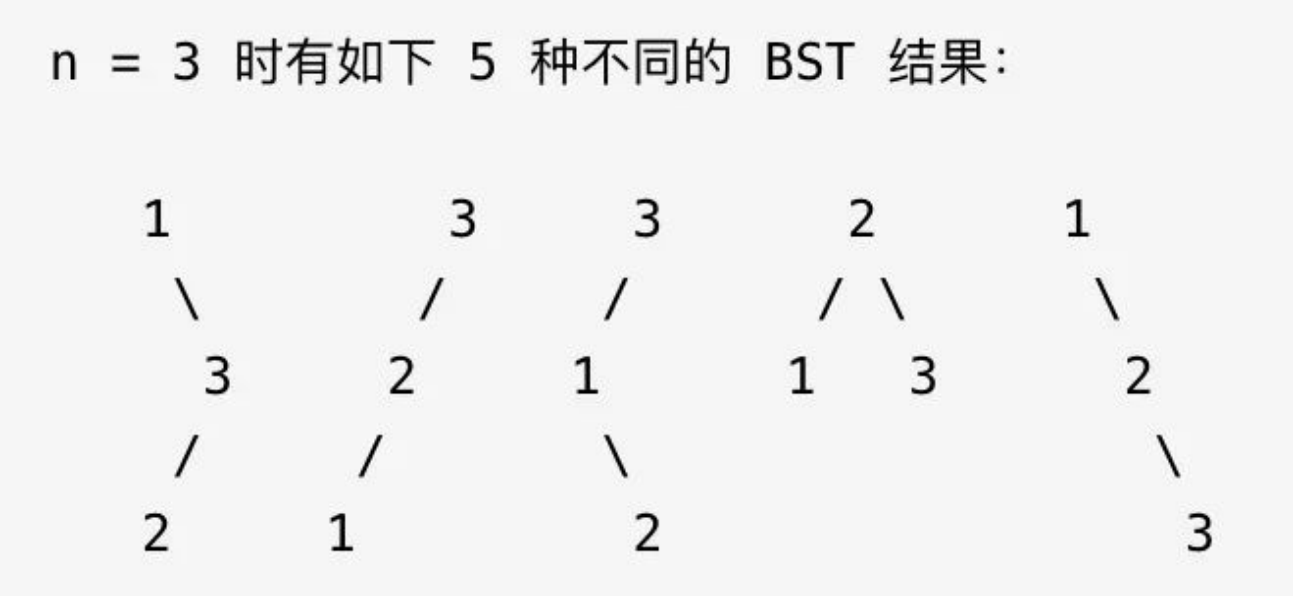
本文就来写手把手刷 BST 系列的第三篇，循序渐进地讲两道题，如何计算所有合法 BST。

第一道题是力扣第 96 题「不同的二叉搜索树」，给你输入一个正整数 `n`，请你计算，存储 `{1,2,3...,n}` 这些值共有有多少种不同的 BST 结构。

函数签名如下：

```
int numTrees(int n);
```

比如说输入 `n = 3`，算法返回 5，因为共有如下 5 种不同的 BST 结构存储 `{1,2,3}`：



这就是一个正宗的穷举问题，那么什么方式能够正确地穷举合法 BST 的数量呢？

我们前文说过，不要小看「穷举」，这是一件看起来简单但是比较有技术含量的事情，问题的关键就是不能数漏，也不能数多，你咋整？

之前 手把手刷二叉树第一期 说过，二叉树算法的关键就在于明确根节点需要做什么，其实 BST 作为一种特殊的二叉树，核心思路也是一样的。

举个例子，比如给算法输入 `n = 5` ，也就是说用 `{1,2,3,4,5}` 这些数字去构造 BST。

首先，这棵 BST 的根节点总共有几种情况？

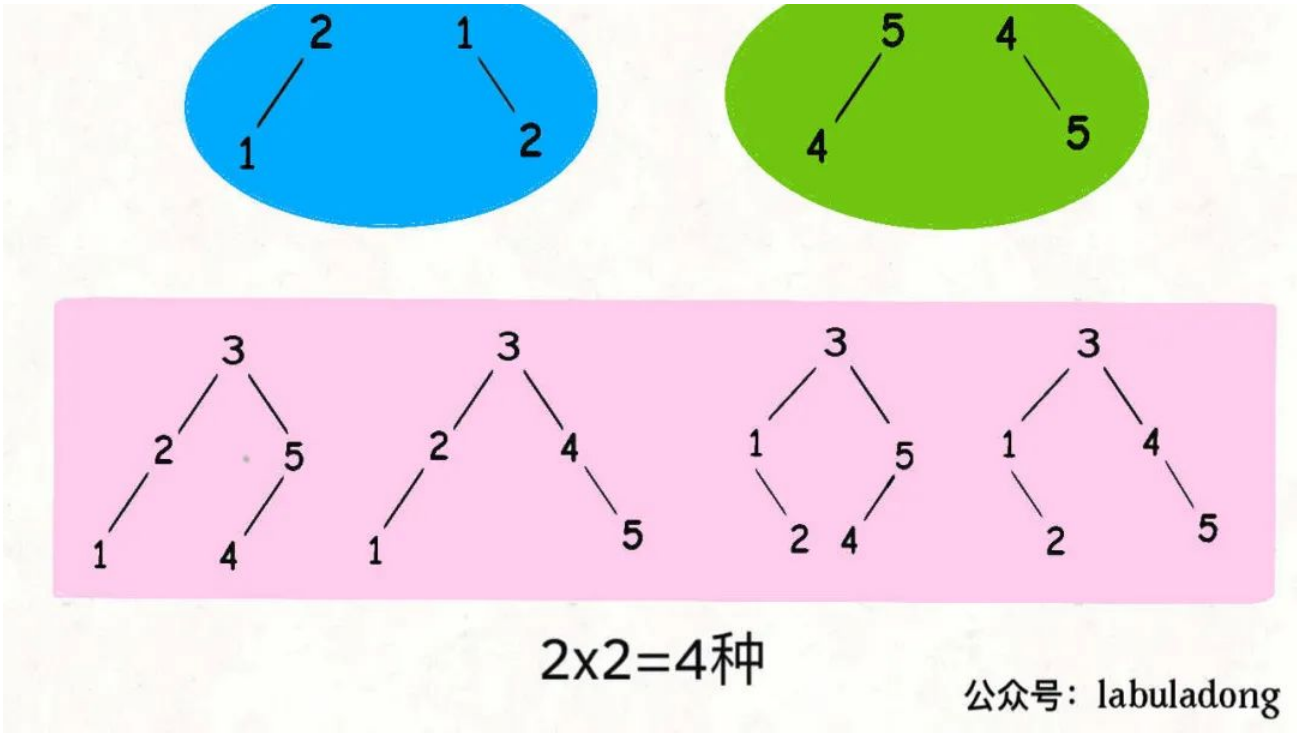
显然有 5 种情况对吧，因为每个数字都可以作为根节点。

比如说我们固定 `3` 作为根节点，这个前提下能有几种不同的 BST 呢？

根据 BST 的特性，根节点的左子树都比根节点的值小，右子树的值都比根节点的值大。

所以如果固定 `3` 作为根节点，左子树节点就是 `{1,2}` 的组合，右子树就是 `{4,5}` 的组合。

左子树的组合数和右子树的组合数乘积就是 `3` 作为根节点时的 BST 个数。



我们这是说了 3 为根节点这一种特殊情况，其实其他的节点也是一样的。

那你可能会问，我们可以一眼看出 {1,2} 和 {4,5} 有几种组合，但是怎么让算法进行计算呢？

其实很简单，只需要递归就行了，我们可以写这样一个函数：

```
// 定义：闭区间 [lo, hi] 的数字能组成 count(lo, hi) 种 BST
int count(int lo, int hi);
```

根据这个函数的定义，结合刚才的分析，可以写出代码：

```
/* 主函数 */
int numTrees(int n) {
    // 计算闭区间 [1, n] 组成的 BST 个数
    return count(1, n);
}

/* 计算闭区间 [lo, hi] 组成的 BST 个数 */
int count(int lo, int hi) {
    // base case
    if (lo > hi) return 1;

    int res = 0;
    for (int i = lo; i <= hi; i++) {
        // i 的值作为根节点 root
        int left = count(lo, i - 1);
        int right = count(i + 1, hi);
        // 左右子树的组合数乘积是 BST 的总数
        res += left * right;
    }

    return res;
}
```

注意 base case，显然当 `lo > hi` 闭区间 `[lo, hi]` 肯定是个空区间，也就对应着空节点 null，虽然是空节点，但是也是一种情况，所以要返回 1 而不能返回 0。

这样，题目的要求已经实现了，但是时间复杂度非常高，肯定存在重叠子问题。

前文动态规划相关的问题多次讲过消除重叠子问题的方法，无非就是加一个备忘录：

```
// 备忘录
int[][] memo;

int numTrees(int n) {
    // 备忘录的值初始化为 0
    memo = new int[n + 1][n + 1];
    return count(1, n);
}

int count(int lo, int hi) {
    if (lo > hi) return 1;
    // 查备忘录
    if (memo[lo][hi] != 0) {
        return memo[lo][hi];
    }

    int res = 0;
    for (int mid = lo; mid <= hi; mid++) {
        int left = count(lo, mid - 1);
        int right = count(mid + 1, hi);
        res += left * right;
    }
    // 将结果存入备忘录
    memo[lo][hi] = res;

    return res;
}
```

这样，这道题就完全解决了。

那么，如果给一个进阶题目，不止让你计算有几个不同的 BST，而是要你构建出所有合法的 BST，如何实现这个算法呢？

这道题就是力扣第 95 题「不同的二叉搜索树 II」，让你构建所有 BST，函数签名如下：

```
List<TreeNode> generateTrees(int n);
```

比如说输入 `n = 3`，算法返回一个列表，列表中存储着如下五棵 BST 的根节点：

明白了上道题构造合法 BST 的方法，这道题的思路也是一样的：

1、穷举 `root` 节点的所有可能。

2、递归构造出左右子树的所有合法 BST。

3、给 `root` 节点穷举所有左右子树的组合。

我们可以直接看代码：

```java
/* 主函数 */
public List<TreeNode> generateTrees(int n) {
    if (n == 0) return new LinkedList<>();
    // 构造闭区间 [1, n] 组成的 BST
    return build(1, n);
}

/* 构造闭区间 [lo, hi] 组成的 BST */
List<TreeNode> build(int lo, int hi) {
    List<TreeNode> res = new LinkedList<>();
    // base case
    if (lo > hi) {
        res.add(null);
        return res;
    }

    // 1、穷举 root 节点的所有可能。
    for (int i = lo; i <= hi; i++) {
        // 2、递归构造出左右子树的所有合法 BST。
        List<TreeNode> leftTree = build(lo, i - 1);
        List<TreeNode> rightTree = build(i + 1, hi);
        // 3、给 root 节点穷举所有左右子树的组合。
        for (TreeNode left : leftTree) {
            for (TreeNode right : rightTree) {
                // i 作为根节点 root 的值
                TreeNode root = new TreeNode(i);
                root.left = left;
                root.right = right;
                res.add(root);
            }
        }
    }

    return res;
}
```

这样，两道题都解决了。

希望二叉树和 BST 的系列文章对你有帮助。

精华文章目录点这里

收录于合集 #手把手刷数据结构 33

上一篇
完全二叉树的节点数，你真的会算吗？

下一篇
原创 | 手把手刷二叉搜索树（第二期）

Read more

People who liked this content also liked

高频面试系列：单词拆分问题
labuladong