标签

- 剪枝

难度

- 中等

入选理由

1. 剪枝通常都是对递归树剪，最典型的就是回溯。而这道题就是树，让你形象化认识剪枝

题目地址（814 二叉树剪枝）

https://leetcode-cn.com/problems/binary-tree-pruning

题目描述

```
给定二叉树根结点 root ，此外树的每个结点的值要么是 0，要么是 1。

返回移除了所有不包含 1 的子树的原二叉树。

( 节点 X 的子树为 X 本身，以及所有 X 的后代。)

示例1:
输入: [1,null,0,0,1]
输出: [1,null,0,null,1]

示例2:
输入: [1,0,1,0,0,0,1]
输出: [1,null,1,null,1]

示例3:
输入: [1,1,0,1,1,0,1,0]
输出: [1,1,0,1,1,null,1]

说明:

给定的二叉树最多有 100 个节点。
每个节点的值只会为 0 或 1
```

前置知识

- 二叉树
- 递归

思路

这个题可是算真正意义的“剪枝”了，出这个题的主要原因是想让大家理解，其实我们日常使用的各种搜索算法其实和这颗二叉树很像，这个题里让我们剪掉全 0 的子树，这就和我们剪掉重复解或者不可行解非常类似，因此这个题用来了解搜索空间和剪枝很合适。

说了半天看这道题吧，一般树的题是跑不了递归的，我说一下我做树这种题的初使递归的考虑过程：

- 首先只考虑只有一个根结点的树桩：是 0 返回 null 不是 0 返回这个节点
- 再考虑只有一个根结点和左右两个叶子节点的树：先去看左叶子节点是否是 0，是剪掉，否则留下来，右叶子节点同理，如果左右节点都剪掉了就又回到了第一种情况。
- 泛化上述过程：首先我们去对根结点的左子树修剪，再对右子树修剪，如果左右子树都被剪没了，那就判断根结点是不是也要被剪掉。

上述分析过程很容易抽象出如下递归的代码。

代码

代码支持：Java，Python，JS

Java Code：

```java
public TreeNode pruneTree(TreeNode root) {

    if (root == null)
        return null;

    root.left = pruneTree(root.left);
    root.right = pruneTree(root.right);

    return root.val == 0 && root.left == null && root.right == null ? null : root;
}
```

Python Code：

```python
class Solution(object):
    def pruneTree(self, root):
        def containsOne(node):
            if not node: return False
```

```
            left = containsOne(node.left)
            right = containsOne(node.right)
            if not left: node.left = None
            if not right: node.right = None
            return node.val == 1 or left or right

        return root if containsOne(root) else None
```

JS Code：

```
/**
 * 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 {TreeNode}
 */
// [0,null,0,0,0]
var pruneTree = function (root) {
  function dfs(root) {
    if (!root) return 0;
    const l = dfs(root.left);
    const r = dfs(root.right);
    if (l == 0) root.left = null;
    if (r == 0) root.right = null;
    return root.val + l + r;
  }
  ans = new TreeNode(-1);
  ans.left = root;
  dfs(ans);
  return ans.left;
};
```

复杂度分析

- 空间复杂度：没有额外空间使用，因此空间复杂度就是递归栈的最大深度$O(H)$，其中 H 是树高。

- 时间复杂度：最坏情况就是所有节点都剪掉了，因此时间复杂度是$O(N)$，其中 N 是树节点的个数。

上一页 下一页