递归

2022年8月12日 9:36

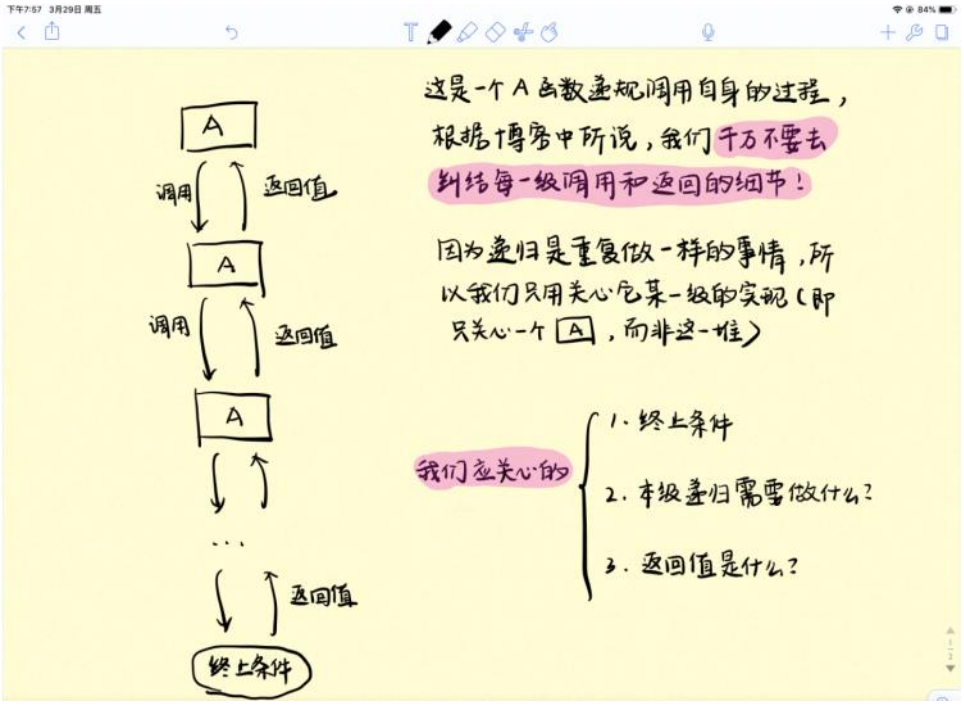
三道题套路解决递归问题

Jan 25, 2020 | leetcode
150 条评论
文章目录

2020-01-25更新：
说来惭愧，这是19年初写的文章了，那会的我还是不到50题的水平。当时是学了点后端的东西写了个博客网站，给它折腾上线后，就写了个文章放上去，顺便丢leetcode-cn上引流。没想到一年下来有好几万访问量，还有不少同学邮件联系我。
这一年来一直没有更新，最近过年回家没事，把博客还是给转到hexo上了(方便)。现在的我大概是poj+leetcode 600题水平了，对递归的看法也和去年写这篇博客的时候不太相同了。年后我一定会抽时间更新我的理解！
尽管这篇博客可能略浅，但我相信一定是适用于初学者的，如果是完全写不出递归代码的同学，还是欢迎您阅读的，它可能很适合您入门～
以下是正文：

递归解题三部曲

何为递归？程序反复调用自身即是递归。
我自己在刚开始解决递归问题的时候，总是会去纠结这一层函数做了什么，它调用自身后的下一层函数又做了什么…然后就会觉得实现一个递归解法十分复杂，根本就无从下手。
相信很多初学者和我一样，这是一个思维误区，一定要走出来。既然递归是一个反复调用自身的过程，这就说明它每一级的功能都是一样的，**因此我们只需要关注一级递归的解决过程即可。**



如上图所示，我们需要关心的主要是以下三点：

1. 整个递归的终止条件。
2. 一级递归需要做什么？
3. 应该返回给上一级的返回值是什么？

因此，也就有了我们解递归题的三部曲：

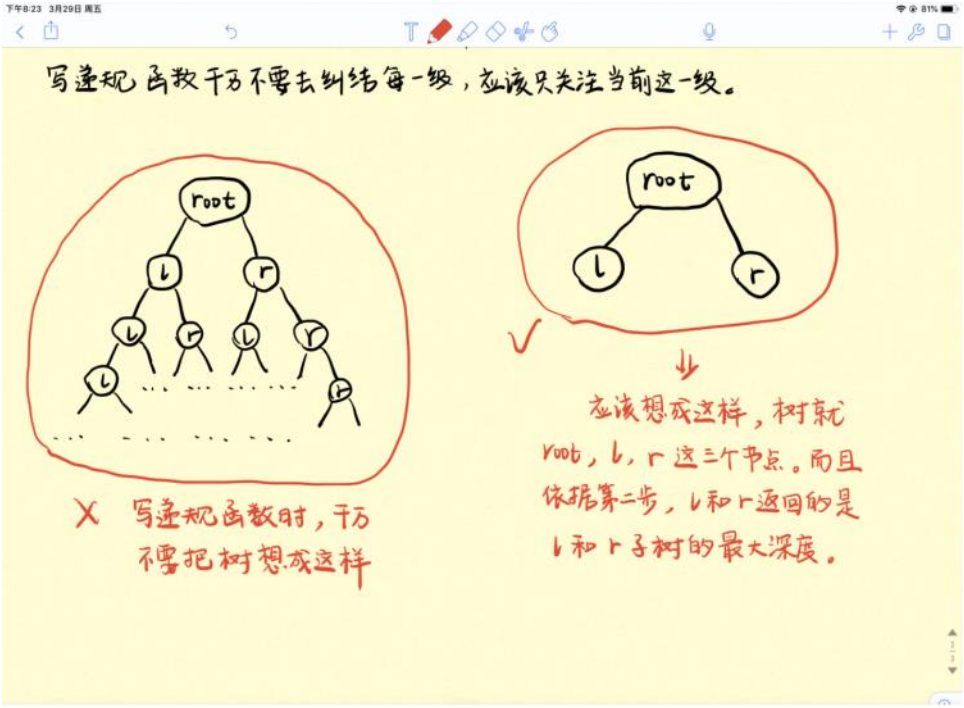
1. **找整个递归的终止条件：递归应该在什么时候结束？**
2. **找返回值：应该给上一级返回什么信息？**
3. **本级递归应该做什么：在这一级递归中，应该完成什么任务？**

一定要理解这3步，这就是以后递归秒杀算法题的依据和思路。
但这么说好像很空，我们来以题目作为例子，看看怎么套这个模版，相信3道题下来，你就能慢慢理解这个模版。之后再解这种套路递归题都能直接秒了。

例1：求二叉树的最大深度

先看一道简单的Leetcode题目： Leetcode 104. 二叉树的最大深度
题目很简单，求二叉树的最大深度，那么直接套递归解题三部曲模版：

1. **找终止条件。** 什么情况下递归结束？当然是树为空的时候，此时树的深度为0，递归就结束了。
2. **找返回值。** 应该返回什么？题目求的是树的最大深度，我们需要从每一级得到的信息自然是当前这一级对应的树的最大深度，因此我们的返回值应该是当前树的最大深度，这一步可以结合第三步来看。
3. **本级递归应该做什么。** 首先，还是强调要走出之前的思维误区，递归后我们眼里的树一定是这个样子的，看下图。此时就三个节点：root、root.left、root.right，其中根据第二步，root.left和root.right分别记录的是root的左右子树的最大深度。那么本级递归应该做什么就很明确了，自然就是在root的左右子树中选择较大的一个，再加上1就是以root为根的子树的最大深度了，然后再返回这个深度即可。



具体Java代码如下：

```
classSolution{
    publicintmaxDepth(TreeNode root){
        //终止条件：当树为空时结束递归，并返回当前深度0
        if(root == null){
            return0;
        }
        //root的左、右子树的最大深度
        intleftDepth = maxDepth(root.left);
        intrightDepth = maxDepth(root.right);
        //返回的是左右子树的最大深度+1
        returnMath.max(leftDepth, rightDepth) + 1;
    }
}
```

当足够熟练后，也可以和Leetcode评论区一样，很骚的几行代码搞定问题，让之后的新手看的一脸懵逼(这道题也是我第一次一行代码搞定一道Leetcode题)：

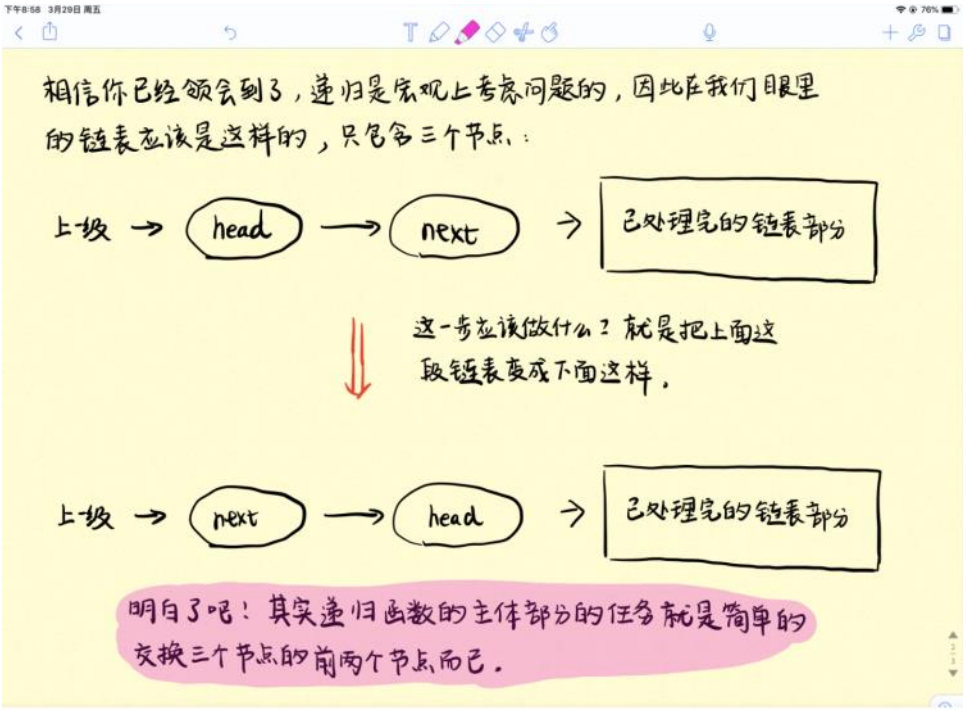
```
classSolution{
    publicintmaxDepth(TreeNode root){
        returnroot == null? 0: Math.max(maxDepth(root.left), maxDepth(root.right))
+ 1;
    }
}
```

例2：两两交换链表中的节点

看了一道递归套路解决二叉树的问题后，有点套路搞定递归的感觉了吗？我们再来看一道Leetcode中等难度的链表的问题，掌握套路后这种中等难度的问题真的就是秒：[Leetcode 24. 两两交换链表中的节点]

直接上三部曲模版：

1. **找终止条件。** 什么情况下递归终止？没得交换的时候，递归就终止了呗。因此当链表只剩一个节点或者没有节点的时候，自然递归就终止了。
2. **找返回值。** 我们希望向上一级递归返回什么信息？由于我们的目的是两两交换链表中相邻的节点，因此自然希望交换给上一级递归的是已经完成交换处理，即已经处理好的链表。
3. **本级递归应该做什么。** 结合第二步，看下图！由于只考虑本级递归，所以这个链表在我们眼里其实也就三个节点：head、head.next、已处理完的链表部分。而本级递归的任务也就是交换这3个节点中的前两个节点，就很easy了。



附上Java代码：

```
classSolution{
    publicListNode swapPairs(ListNode head){
        //终止条件：链表只剩一个节点或者没节点了，没得交换了。返回的是已经处理好的链表
        if(head == null|| head.next == null){
            returnhead;
        }
        //一共三个节点:head, next, swapPairs(next.next)
        //下面的任务便是交换这3个节点中的前两个节点
        ListNode next = head.next;
        head.next = swapPairs(next.next);
        next.next = head;
        //根据第二步：返回给上一级的是当前已经完成交换后，即处理好了的链表部分
        returnnext;
    }
}
```

例3：平衡二叉树

相信经过以上2道题，你已经大概理解了这个模版的解题流程了。

那么请你先不看以下部分，尝试解决一下这道easy难度的Leetcode题（个人觉得此题比上面的medium难度要难）：Leetcode 110. 平衡二叉树

我觉得这个题真的是集合了模版的精髓所在，下面套三部曲模版：

1. **找终止条件。** 什么情况下递归应该终止？自然是子树为空的时候，空树自然是平衡二叉树了。
2. **应该返回什么信息：**

为什么我说这个题是集合了模版精髓？正是因为此题的返回值。要知道我们搞这么多花里胡哨的，都是为了能写出正确的递归函数，因此在解这个题的时候，我们就需要思考，我们到底希望返回什么值？

何为平衡二叉树？平衡二叉树即左右两棵子树高度差不大于1的二叉树。而对于一颗树，它是一个平衡二叉树需要满足三个条件：**它的左子树是平衡二叉树，它的右子树是平衡二叉树，它的左右子树的高度差不大于1**。换句话说：如果它的左子树或右子树不是平衡二叉树，或者它的左右子树高度差大于1，那么它就不是平衡二叉树。

而在我们眼里，这颗二叉树就3个节点：root、left、right。那么我们应该返回什么呢？如果返回一个当前树是否是平衡二叉树的boolean类型的值，那么我只知道left和right这两棵树是否是平衡二叉树，无法得出left和right的高度差是否不大于1，自然也就无法得出root

这棵树是否是平衡二叉树了。而如果我返回的是一个平衡二叉树的高度的int类型的值，那么我就只知道两棵树的高度，但无法知道这两棵树是不是平衡二叉树，自然也就没法判断root这棵树是不是平衡二叉树了。
因此，这里我们返回的信息应该是既包含子树的深度的int类型的值，又包含子树是否是平衡二叉树的boolean类型的值。可以单独定义一个ReturnNode类，如下：

```
classReturnNode{
  booleanisB;
  intdepth;
  //构造方法
  publicReturnNode(booleanisB, intdepth){
    this.isB = isB;
    this.depth = depth;
  }
}
```

3. **本级递归应该做什么。** 知道了第二步的返回值后，这一步就很简单了。目前树有三个节点：root，left，right。我们首先判断left子树和right子树是否是平衡二叉树，如果不是则直接返回false。再判断两树高度差是否不大于1，如果大于1也直接返回false。否则说明以root为节点的子树是平衡二叉树，那么就返回true和它的高度。
具体的Java代码如下：

```
classSolution{
    //这个ReturnNode是参考我描述的递归套路的第二步：思考返回值是什么
    //一棵树是BST等价于它的左、右俩子树都是BST且俩子树高度差不超过1
    //因此我认为返回值应该包含当前树是否是BST和当前树的高度这两个信息
    privateclassReturnNode{
        booleanisB;
        intdepth;
        publicReturnNode(intdepth, booleanisB){
            this.isB = isB;
            this.depth = depth;
        }
    }
    //主函数
    publicbooleanisBalanced(TreeNode root){
        returnisBST(root).isB;
    }
    //参考递归套路的第三部：描述单次执行过程是什么样的
    //这里的单次执行过程具体如下：
    //是否终止?->没终止的话，判断是否满足不平衡的三个条件->返回值
    publicReturnNode isBST(TreeNode root){
        if(root == null){
            returnnewReturnNode(0, true);
        }
        //不平衡的情况有3种：左树不平衡、右树不平衡、左树和右树差的绝对值大于1
        ReturnNode left = isBST(root.left);
        ReturnNode right = isBST(root.right);
        if(left.isB == false|| right.isB == false){
            returnnewReturnNode(0, false);
        }
        if(Math.abs(left.depth - right.depth) > 1){
            returnnewReturnNode(0, false);
        }
        //不满足上面3种情况，说明平衡了，树的深度为左右俩子树最大深度+1
        returnnewReturnNode(Math.max(left.depth, right.depth) + 1, true);
    }
}
```

一些可以用这个套路解决的题

暂时就写这么多啦，作为一个高考语文及格分，大学又学了工科的人，表述能力实在差因此啰啰嗦嗦写了一大堆，希望大家能理解这个很好用的套路。
下面我再列举几道我在刷题过程中遇到的也是用这个套路秒的题，真的太多了，大部分链表和树的递归题都能这么秒，因为树和链表天生就是适合递归的结构。
我会随时补充，正好大家可以看了上面三个题后可以拿这些题来练练手，看看自己是否能独立快速准确的写出递归解法了。
Leetcode 101. 对称二叉树
Leetcode 111. 二叉树的最小深度
Leetcode 226. 翻转二叉树：这个题的备注是最骚的。Mac OS下载神器homebrew的大佬作者去面试谷歌，没做出来这道算法题，然后被谷歌面试官怼了：”我们90％的工程师使用您编写的软件(Homebrew)，但是您却无法在面试时在白板上写出翻转二叉树这道题，这太糟糕了。”
Leetcode 617. 合并二叉树
Leetcode 654. 最大二叉树
Leetcode 83. 删除排序链表中的重复元素

来自 <https://lyl0724.github.io/2020/01/25/1/>