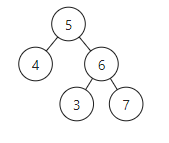
98.验证二叉搜索树

1.用先序遍历的递归法不能解决

按照之前做的，在二叉树上递归的方法，即单层逻辑为：判断左孩子的节点值是否小于当前根节点的节点值，右孩子的节点值是否大于当前根节点的节点值，是存在问题的。

因为二叉搜索树要求，左子树上的所有节点的值都小于根节点的值，右子树上的所有节点的值都大于根节点的值，上述方法将大小关系局限于子树内，遇到下面这样的二叉树就无法正确判断：



2.为了解决上述问题想到的递归法

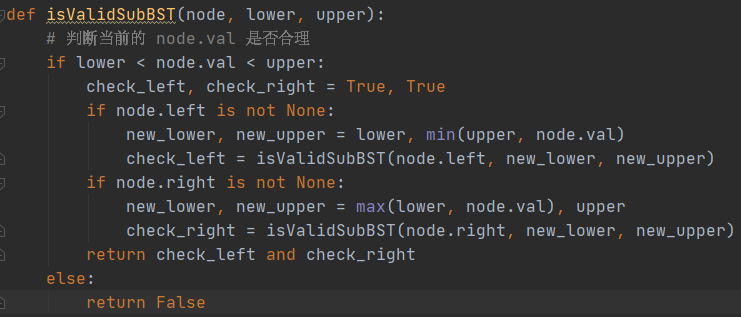
既然从根节点判断两个孩子的方法存在问题，可以将问题转换为判断当前节点的值是否满足条件。

条件是从根节点上递归传递下来的，有一个上界和一个下界。

对于左孩子来说，需要修改上界；对于右孩子来说，需要修改下界。

与其保存所有可能的上下界值，不如只保存最大的下界和最小的上界，只要满足这两个条件，剩下的一定也能满足。

注意这种方法，对于根节点，上下界应该是节点值取值范围的上下界。

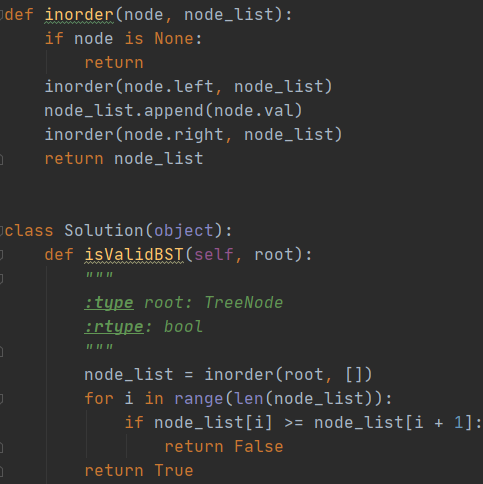


还有一个小问题是，为了防止左右子树判断时，上下界互相干扰，不要直接修改上下界的值，而是创建一个新的变量，用于保存新的上界或下界。

3. 真正解决问题的方法

二叉搜索树的性质：中序遍历得到的序列是有序的。

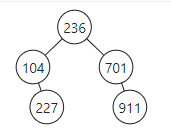
只要先对二叉搜索树做中序遍历，再对得到的序列判断是否有序即可：



530. 二叉搜索树的最小绝对差

1. 最小绝对差不一定发生在父子节点之间

因为二叉搜索树的性质，最小绝对差一定发生在中序遍历的相邻元素之间，但是中序遍历的相邻元素不一定是父子关系：



最小绝对差：236-227 = 9，但是227和236不是父子关系。

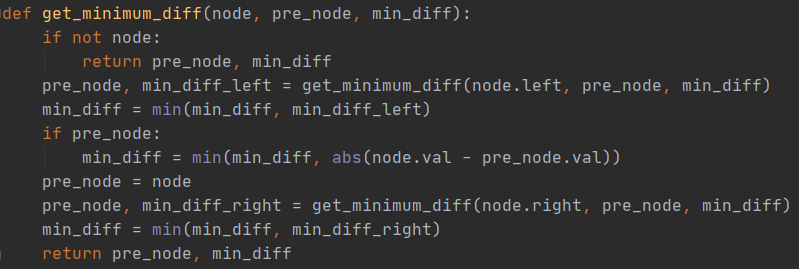
和98验证二叉树的问题一样，不能把递归局限在一棵父子关系的子树中。

2. 简易做法

先中序遍历二叉搜索树，再在有序序列中直接查找最小绝对差。

3. 对2的优化

可以在中序遍历中直接查找最小绝对差，需要记录上一个节点。



注意：在这种情况下，需要把pre\_node也作为返回值返回，否则在中序遍历中，无法更新pre\_node的值，会导致1那样的错误。

501. 二叉搜索树中的众数

目标是通过一次树上的中序遍历，就能够得到众数序列，并且不消耗多余的存储空间。

1. pre\_node初始条件判断

在这里，用于存储上一个节点值的pre\_node不是一个树节点，所以不能用if not pre\_node的方法判断初始条件，否则当pre\_node=0的时候会产生错误判断，对pre\_node进行错误的重新赋值。

从另一个角度来说，也可以把pre\_node设置为一个树节点或到树节点的指针，就像530题一样。

（Python的自动类型转换造成的问题）

1. 什么时候修改结果数组？

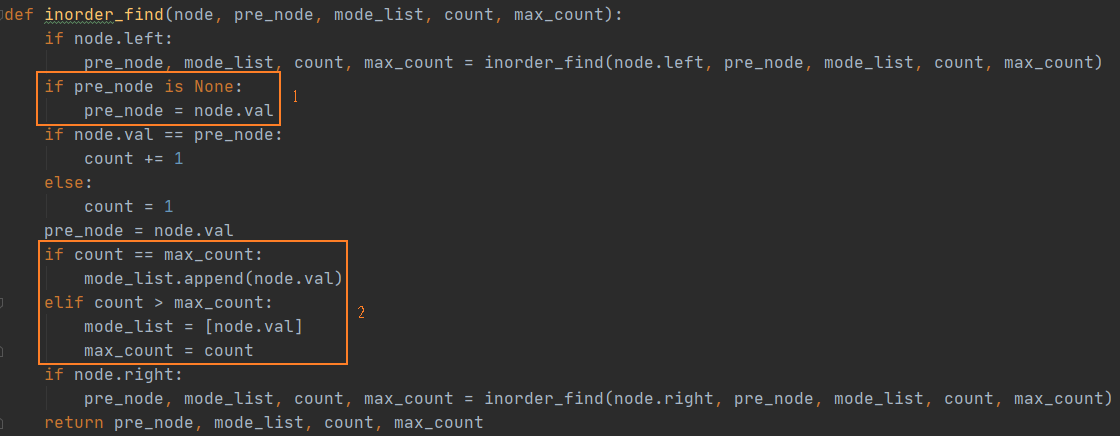
我一开始的想法是，当遍历到和pre\_node值不同的节点时，修改结果数组。

这样会产生两个问题：

① 需要添加到结果数组中的值是pre\_node的值，而不是node的值（因为此时count和max\_count的计数是pre\_node的）；

② 遍历结束之后，需要对最后遍历到的pre\_node值做一次额外的判断，否则如果最后遍历到的值也是众数的话，不能正常的加入结果数组（因为不会再有不同的节点了）。

与其考虑的这么麻烦，不如遍历到每个节点的时候，就直接判断一次是否能够加入结果数组或修改结果数组。



1. 类的成员变量

如上图所示，每次递归都需要把所有更新后的参数都返回，不仅很麻烦，而且很容易遗忘导致错误。

不如直接把这几个随着递归不断更新的变量作为类的成员变量，把递归函数也变成类的成员函数，这样就不需要每次回传这几个变量也能够实现随着递归更新了。

236.二叉树的最近公共祖先

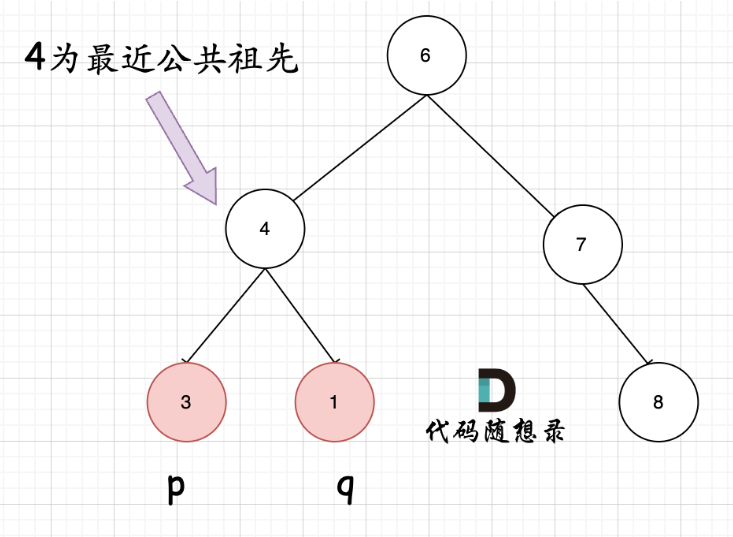
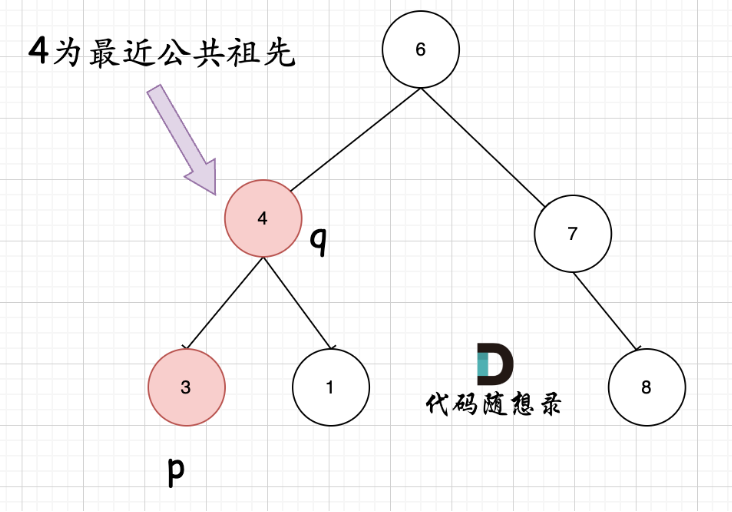
1. 最初的想法

先序遍历，从上向下传递一个存放着当前节点所有祖先的数组，因此也可以得到两个目标节点的祖先数组；对这两个祖先数组做双指针遍历，找到最后一个相等的节点，即为最近的公共祖先。

2. 仅递归的方法

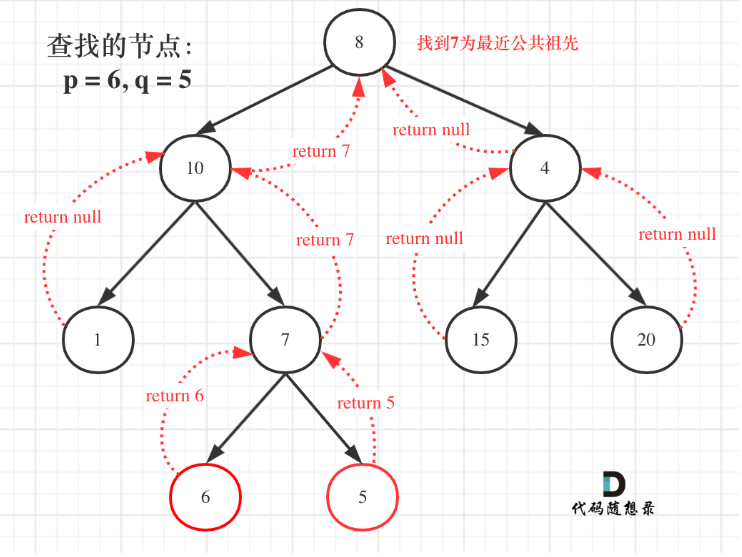
想要找到最近的祖先节点，本质上是从底层向顶层遍历 → 后序遍历。

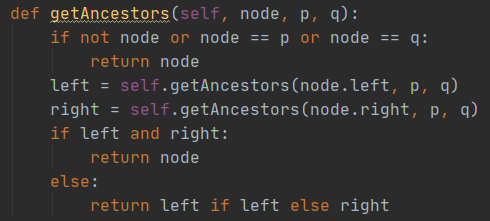
二叉树中成对的节点都可以转换成两种情况：① 两个节点在某个公共祖先的左右子树上；② 一个节点是另一个节点的子节点。

在后序遍历的过程中，如果找到了p或q节点，或遍历到空节点就返回该节点。对于当前节点，如果左右遍历的返回值均不为空，说明当前节点就是最近的祖先节点，那么返回当前节点；如果左右遍历的返回值有一个为空，则返回不为空的那一个；对于左右遍历的返回值均为空的情况，则返回任意一个均可（相当于返回空）。

具体流程如下：





需要额外注意一下，代码实际上包括了一个节点是另一个节点的子节点的情况（这时返回作为祖先的那个节点作为最近公共祖先）。

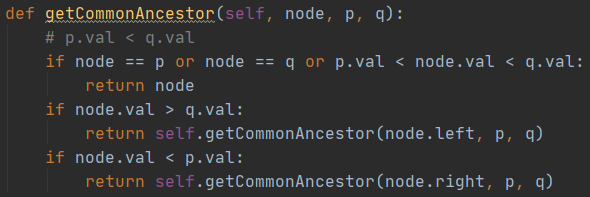
235.二叉搜索树的最近公共祖先

因为二叉搜索树的有序性，现在不需要后序遍历+整棵二叉树也可以找到最近公共祖先了！

当p和q节点分散在某个node节点的左右子树上时，这个node节点就是最近公共祖先。（因为再向下遍历，不可能找到公共祖先了）

当某个node节点为p或q时，此时另一个节点一定在这个node节点的子树上，这个node节点就是最近公共祖先。

不满足上述两种情况时，只需要遍历node节点的左子树或右子树，查找最近祖先节点即可。



注意：我这里是事先对p和q进行了排序，保证p.val一定小于q.val，这样可以减少递归中的判断。

（和普通二叉树的区别在于，在二叉搜索树上，我可以知道p和q节点具体在什么位置，但是在普通二叉树上，我必须先对整棵树进行遍历，找到p和q节点）

450.删除二叉搜索树中的节点

因为删除二叉搜索树中的节点需要重整树结构，所以要对所有可能的情况单独考虑。

1. 整体逻辑

一共有4种不同情况：

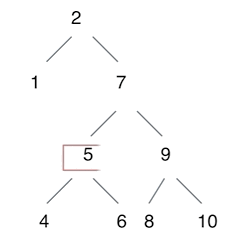
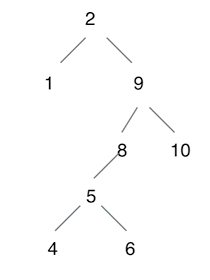
① 找不到目标节点，直接返回；

（这种情况一开始被我忽略了，但是其实题目里是有提，目标节点有可能找不到的）

② 找到目标节点，目标节点是叶子节点，直接删除该节点，返回空；

③ 找到目标节点，目标节点只有左子树或只有右子树，删除该节点，返回该节点的左子树根节点或右子树根节点；

④ 找到目标节点，目标节点既有左子树又有右子树，删除该节点，把该节点的左子树根节点放到右子树最左边节点的左孩子的位置上，返回该节点的右子树根节点。

上图展示的是第4种情况，删除节点7的步骤。因为右子树上的所有节点一定比左子树上的节点值要大，只要把左子树放在最左边，就能保证还满足二叉搜索树的条件。

2. 记录前一个节点？

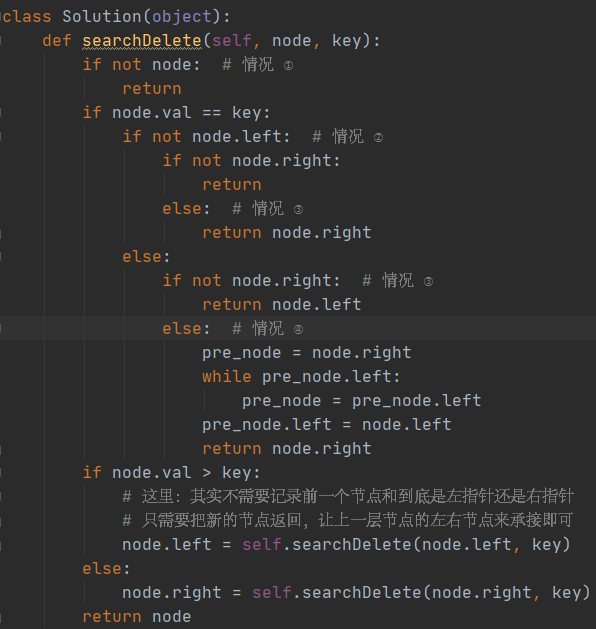
这个题目是没法记录前一个节点的，因为不知道是左孩子还是右孩子。

其实也没必要记录前一个节点，只需要返回更新后的节点即可，让上一层递归的父节点用左指针或右指针去承接这个返回值。

3. 递归当前节点还是父节点？

第一次做这道题的时候我想的是递归父节点，检查两个子节点是否是目标节点。

这种方法存在一个很大的问题：如果根节点是目标节点，是没有办法处理的。所以还是得递归当前节点，检查当前节点是否是目标节点，并返回当前节点给上层递归。



100114. 元素和最小的山形三元组II

1. 不能用三指针

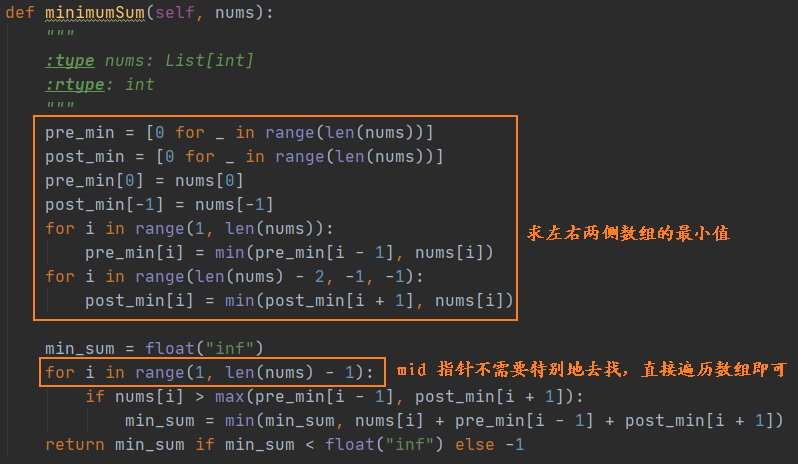
比赛的时候第一反应是用三指针解决三重for循环的问题，但是是不行的：题目中的数组是无序的，没有办法判断当条件不满足时应该移动哪个指针。

1. 解决方案

当时想到了控制中间的指针mid，检查两侧的数组中是否存在满足条件的left和right的这种做法，但是拘泥于要把mid从最大值开始一路找下来，其实直接在数组上遍历mid指针就行。

当mid指针对应的值确定之后，想要总和最小，只需要找到两侧数组中的最小值即可。

不需要每次更新mid指针之后，在两侧的数组中重新寻找最小值，直接保存一个记录两侧数组中前缀和最小值的数组，在改变mid指针指向之后，查阅这两个数组即可。



100097. 合法分组的最少组数

1. 是不是模拟？

竞赛的时候，第一反应是模拟，先把数字的频次统计出来，然后对每个不符合条件的分组进行拆分，一直在报错，而且每次还需要排序，时间复杂度很高。

1. 解决方案

假设某组数字的数量为s，尝试划分为数量为k和k+1的小组，有两种情况：

① 先尝试划分为数量为k+1的小组，此时划分的组数为，剩余的数量为。

如果r = 0，则总组数为d；

如果r > 0，此时最多可以从每组中取1个元素，与剩下的r个元素构成一组。需要满足的条件是，总组数为d + 1。

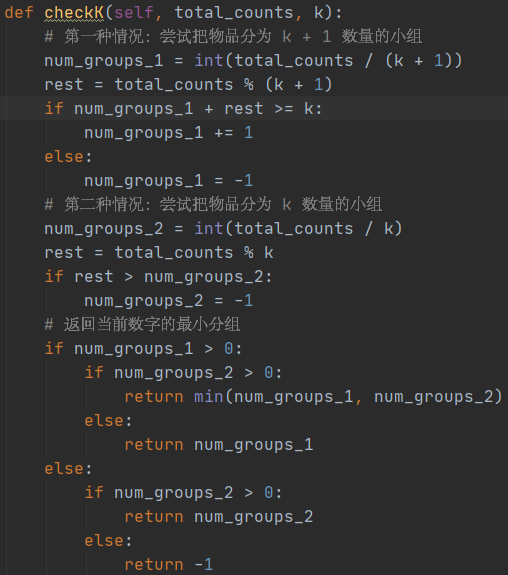
② 再尝试划分为数量为k的小组，此时划分的组数是，剩余的数量为。

如果r = 0，则总组数为d；

如果r > 0，则需要把这剩下的r个元素放入每组中，需要满足的条件是r <= d，总组数为d。

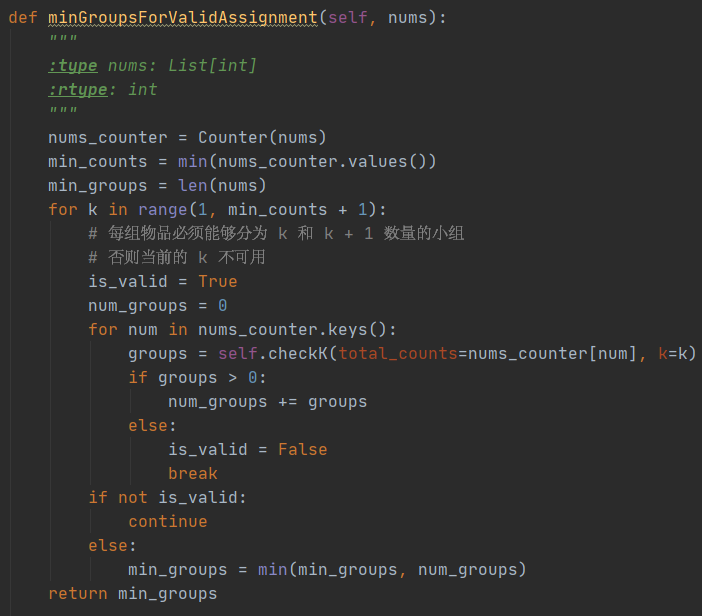
③ 如果上述两种情况都不能划分，证明当前数字组不能划分为k或k + 1的小组，则当前的k不合法，更换k值，继续判断。

否则，在上述两种情况中，选择总组数较小的返回。

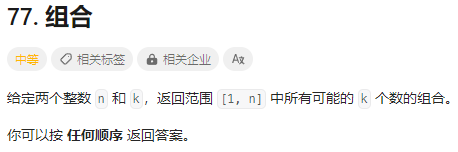


1. 主函数的代码逻辑

K的可能取值为1 ~ min\_counts，也就是最小的数字组的数量，只需要遍历这个k值，判断所有的数字组是否都能被划分，得到最小划分组数即可。



77. 组合



经典的回溯法。

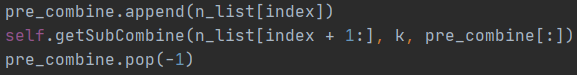
1. 整体逻辑

回溯法可以转换成一棵N叉树的形式，用树上递归的方式来考虑会比直接考虑如何回溯要简单。

注意：

① 因为是一棵N叉树，所以对于每棵子树，要遍历该子树根节点的所有孩子节点。换言之，需要一个for循环。

② 用回溯问题查找组合（也就是在树上查找路径），当把一个新的节点加入当前路径，递归完成，要回溯时，需要再把该节点从当前路径中移除，否则无法得到正确的结果：



只要有回溯，一定会有这个步骤（比如在二叉树上查找路径）。但是一般的递归可能没有回溯的过程（只是return），所以之前做过的题目不太有这个步骤。

1. 本题存在剪枝优化的逻辑

当剩余的数字个数不足以支持达成题目要求的路径时，可以直接把对应递归的树枝剪掉。

举个例子，如果n=4，k=4，唯一满足条件的组合是[1, 2, 3, 4]，那么其余的树枝都可以直接剪掉。

如果当前子树根节点的数字集合是n\_list，已经加入队列的数字是pre\_combine，那么要求

(len(n\_list) – index) + len(pre\_combine) >= k：



1. 整体代码

