Leetcode(python版) 树（困难）

99恢复二叉搜索树

题目：

二叉搜索树中的两个节点被错误地交换。

请在不改变其结构的情况下，恢复这棵树。

**示例 1:**

**输入:** [1,3,null,null,2]

  1

  /

 3

  \

  2

**输出:** [3,1,null,null,2]

  3

  /

 1

  \

  2

**示例 2:**

**输入:** [3,1,4,null,null,2]

3

/ \

1 4

  /

  2

**输出:** [2,1,4,null,null,3]

2

/ \

1 4

  /

 3

**进阶:**

* 使用 O(*n*) 空间复杂度的解法很容易实现。
* 你能想出一个只使用常数空间的解决方案吗？

思路：看到BST就想什么？对，中序遍历是有序的。

那么，如果其中两个被交换了，那么中序遍历的结果一定也就不对了。比如：

[1, 2, 3, 4, 5, 6] ==> [1, 5, 3, 4, 2, 6]

那么，可以看出5这个数字比后面的3大，说明他被打乱了；另外2这个数字，比前面的数字4小，所以他也被打乱了。

所以，可以通过先进行中序遍历得到所有的，然后再查找哪些乱了，再复原，时间复杂度O(n)。

但是，中序遍历的操作不需要完全完成。在中序遍历的过程中，用一个指针保存上个节点，那么当前节点值应该小于前一个节点的值。否则就存在乱序。

第一个乱序的数字是pre，第二个乱序的数字是root，所以用两个指针分别保存

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** recoverTree(self, root):
10. """
11. :type root: TreeNode
12. :rtype: void Do not return anything, modify root in-place instead.
13. """
14. self.first,self.second,self.pre=None, None,None
15. self.inorder(root)
16. self.first.val,self.second.val=self.second.val,self.first.val
18. **def** inorder(self,root):
19. **if** **not** root:
20. **return**
21. self.inorder(root.left)
22. **if** self.pre **and** self.pre.val>root.val:
23. **if** **not** self.first:
24. self.first=self.pre
25. self.second=root
26. self.pre=root
27. self.inorder(root.right)

124二叉树中的最大路径和

题目：

给定一个**非空**二叉树，返回其最大路径和。

本题中，路径被定义为一条从树中任意节点出发，达到任意节点的序列。该路径**至少包含一个**节点，且不一定经过根节点。

**示例 1:**

**输入:** [1,2,3]

**1**

**/ \**

**2** **3**

**输出:** 6

**示例 2:**

**输入:** [-10,9,20,null,null,15,7]

  -10

   / \

  9  **20**

**/  \**

**15   7**

**输出:** 42

思路：我们可以先找路径的最大mr,ml，那么最大值是max（solve(root),solve(left),solve(right), max(mr + root.val + ml, root.val))。

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** maxPathSum(self, root):
10. """
11. :type root: TreeNode
12. :rtype: int
13. """
14. **if** root==None:
15. **return** 0
16. Solution.maxsum=root.val
17. **def** solve(root):
18. **if** root==None:
19. **return** 0
20. sum,l,r=root.val,0,0
21. **if** root.left:
22. l=solve(root.left)
23. **if** l>0:sum+=l
24. **if** root.right:
25. r=solve(root.right)
26. **if** r>0:sum+=r
27. Solution.maxsum=max(Solution.maxsum,sum)
28. **return** max(root.val,max(root.val+l,root.val+r))
29. solve(root)
30. **return** Solution.maxsum

145二叉树的后序遍历

297二叉树的序列化与反序列化

题目：

序列化是将一个数据结构或者对象转换为连续的比特位的操作，进而可以将转换后的数据存储在一个文件或者内存中，同时也可以通过网络传输到另一个计算机环境，采取相反方式重构得到原数据。

请设计一个算法来实现二叉树的序列化与反序列化。这里不限定你的序列 / 反序列化算法执行逻辑，你只需要保证一个二叉树可以被序列化为一个字符串并且将这个字符串反序列化为原始的树结构。

**示例:**

你可以将以下二叉树：

1

/ \

2 3

/ \

4 5

序列化为 "[1,2,3,null,null,4,5]"

**说明:**不要使用类的成员 / 全局 / 静态变量来存储状态，你的序列化和反序列化算法应该是无状态的。

思路：BST的中序遍历是有序的。但我们又知道，只知道树的一种遍历方式，是没法确定这个树的，BST也不例外。如果前序遍历的过程中记录下哪些位置是空节点的话，就是可以确定这棵树的。LeetCode的官方树的构建就是这样的。

因此，这个题采用前序遍历的方式，这样，遍历得到的第一个数组就是BST的根节点，数组后面的这些数中比根节点的值小的是根节点的左子树，比根节点值大的是根节点的右子树（BST的最重要性质）。

因此，重要结论：BST的前序遍历能唯一的确定一颗BST

解编码过程是通过一个队列进行操作。其实也可以是list，不过队列的效率更高。只不过需要把空节点记录下来。然后在反序列化时把它再变成空节点即可。

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Codec:
10. **def** serialize(self, root):
11. """Encodes a tree to a single string.
13. :type root: TreeNode
14. :rtype: str
15. """
16. vals = []
17. **def** preOrder(root):
18. **if** **not** root:
19. vals.append('#')
20. **else**:
21. vals.append(str(root.val))
22. preOrder(root.left)
23. preOrder(root.right)
24. preOrder(root)
25. **return** ' '.join(vals)

28. **def** deserialize(self, data):
29. """Decodes your encoded data to tree.
31. :type data: str
32. :rtype: TreeNode
33. """
34. vals = collections.deque(val **for** val **in** data.split())
35. **def** build():
36. **if** vals:
37. val = vals.popleft()
38. **if** val == '#':
39. **return** None
40. root = TreeNode(int(val))
41. root.left = build()
42. root.right = build()
43. **return** root
44. **return** build()

47. # Your Codec object will be instantiated and called as such:
48. # codec = Codec()
49. # codec.deserialize(codec.serialize(root))

685冗余连接II

题目：

在本问题中，有根树指满足以下条件的**有向**图。该树只有一个根节点，所有其他节点都是该根节点的后继。每一个节点只有一个父节点，除了根节点没有父节点。

输入一个有向图，该图由一个有着N个节点 (节点值不重复1, 2, ..., N) 的树及一条附加的边构成。附加的边的两个顶点包含在1到N中间，这条附加的边不属于树中已存在的边。

结果图是一个以边组成的二维数组。 每一个边 的元素是一对 [u, v]，用以表示**有向**图中连接顶点 u and v和顶点的边，其中父节点u是子节点v的一个父节点。

返回一条能删除的边，使得剩下的图是有N个节点的有根树。若有多个答案，返回最后出现在给定二维数组的答案。

**示例 1:**

**输入:** [[1,2], [1,3], [2,3]]

**输出:** [2,3]

**解释:** 给定的有向图如下:

1

/ \

v v

2-->3

**示例 2:**

**输入:** [[1,2], [2,3], [3,4], [4,1], [1,5]]

**输出:** [4,1]

**解释:** 给定的有向图如下:

5 <- 1 -> 2

^ |

| v

4 <- 3

**注意:**

* 二维数组大小的在3到1000范围内。
* 二维数组中的每个整数在1到N之间，其中 N 是二维数组的大小。

思路：下面的代码实现了并查集查找根节点的代码，并且做了路径压缩，防止树太高导致查找根节点缓慢。

具体到这个题，虽然说是返回最后一个边，但我们知道只需要去除一条边就够了，之前的边不会构成环，直至多余的那条边出现。

另外要注意，当一条边的左右节点的根节点不同时，要把他们设置相同，这样等下次判断某条边的左右节点相同的情况时，说明是多余的那条边了

代码：

834树中距离之和

题目：

给定一个无向、连通的树。树中有 N 个标记为 0...N-1 的节点以及 N-1 条边 。

第 i 条边连接节点 edges[i][0] 和 edges[i][1] 。

返回一个表示节点 i 与其他所有节点距离之和的列表 ans。

**示例 1:**

**输入:** N = 6, edges = [[0,1],[0,2],[2,3],[2,4],[2,5]]

**输出:** [8,12,6,10,10,10]

**解释:**

如下为给定的树的示意图：

0

/ \

1 2

/|\

3 4 5

我们可以计算出 dist(0,1) + dist(0,2) + dist(0,3) + dist(0,4) + dist(0,5)

也就是 1 + 1 + 2 + 2 + 2 = 8。 因此，answer[0] = 8，以此类推。

**说明:** 1 <= N <= 10000

思路：

代码：

968监控二叉树

题目：

思路：

代码：