Leetcode(python) 深度优先搜索 中等题

109有序链表转换二叉搜索树

题目：

给定一个单链表，其中的元素按升序排序，将其转换为高度平衡的二叉搜索树。

本题中，一个高度平衡二叉树是指一个二叉树每个节点 的左右两个子树的高度差的绝对值不超过 1。

示例:

给定的有序链表： [-10, -3, 0, 5, 9],

一个可能的答案是：[0, -3, 9, -10, null, 5], 它可以表示下面这个高度平衡二叉搜索树：

0

/ \

-3 9

/ /

-10 5

思路：

首先看到题目就能想到递归。

递归的子问题： 找到一个数组的中间的位置，其中间往左的数组为其左边的节点，中间往右的数组为其右边的节点。

再从中间往左的数组中找出找出中间位置，这个中间位置的左边为左边的节点，右边为右边的节点，以此类推。

代码：

1. # Definition for singly-linked list.
2. # class ListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.next = None
7. # Definition for a binary tree node.
8. # class TreeNode(object):
9. #     def \_\_init\_\_(self, x):
10. #         self.val = x
11. #         self.left = None
12. #         self.right = None
14. **class** Solution(object):
15. def sortedListToBST(self, head):
16. """
17. :type head: ListNode
18. :rtype: TreeNode
19. """
20. **if** not head:
21. **return** None
22. nums = []
23. **while** head:
24. nums.append(head.val)
25. head = head.next
26. **return** self.sortedArrayToBST(nums)
28. def sortedArrayToBST(self,nums):
29. **if** not nums:
30. **return** None
31. mid=len(nums)//2
32. root=TreeNode(nums[mid])
33. root.left=self.sortedArrayToBST(nums[:mid])
34. root.right=self.sortedArrayToBST(nums[mid+1:])
35. **return** root

199二叉树的右视图

题目：

给定一棵二叉树，想象自己站在它的右侧，按照从顶部到底部的顺序，返回从右侧所能看到的节点值。

示例:

输入: [1,2,3,null,5,null,4]

输出: [1, 3, 4]

解释:

1 <---

/ \

2 3 <---

\ \

5 4 <---

思路：使用队列的技巧在于，queue其实也可以用[-1]直接找到这个层的最后一个元素。

每次进行while循环，都是开始了新的一层，for循环的巧妙在于，直接遍历队列中已有的元素，也就是上层的元素。这样的话就直接把上层的遍历完了，新的层也加入了队列。

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** rightSideView(self, root):
10. """
11. :type root: TreeNode
12. :rtype: List[int]
13. """
14. res=[]
15. **if** **not** root:
16. **return** res
17. queue=collections.deque()
18. queue.append(root)
19. **while** queue:
20. res.append(queue[-1].val)
21. **for** i **in** range(len(queue)):
22. node=queue.popleft()
23. **if** node.left:
24. queue.append(node.left)
25. **if** node.right:
26. queue.append(node.right)
27. **return** res

473火柴拼正方形

题目：

还记得童话《卖火柴的小女孩》吗？现在，你知道小女孩有多少根火柴，请找出一种能使用所有火柴拼成一个正方形的方法。不能折断火柴，可以把火柴连接起来，并且每根火柴都要用到。

输入为小女孩拥有火柴的数目，每根火柴用其长度表示。输出即为是否能用所有的火柴拼成正方形。

示例 1:

输入: [1,1,2,2,2]

输出: true

解释: 能拼成一个边长为2的正方形，每边两根火柴。

示例 2:

输入: [3,3,3,3,4]

输出: false

解释: 不能用所有火柴拼成一个正方形。

注意:

给定的火柴长度和在 0 到 10^9之间。

火柴数组的长度不超过15。

思路：

题目可以理解为：能不能把一组数字分成4组，每组的和是相同的。

我们注意到题目给出的数组长度最多只有15个，基本上可以使用O(N!)的时间复杂度去解决。所以我们可以直接使用回溯法。回溯的思路是先设置好4条边，然后把每一个火柴看看能不能放到4条边中的一个去，如果可以的话就继续向后扫描，直到所有的火柴全部用上为止。

在使用记录四组的和的方式，进行遍历的时候保存各个组的和，如果不能满足就把这个数字再加上，相当于跳过这个数字的方式。最后结束的条件就是所有的数字全部都用完了，因为如果用完了，说明我们把所有的火柴都放到了4条边中的一个，所以得到每组都满足我们条件的结论。

代码：

1. **class** Solution(object):
2. **def** makesquare(self, nums):
3. """
4. :type nums: List[int]
5. :rtype: bool
6. """
7. **if** **not** nums **or** len(nums) < 4: **return** False
8. \_sum = sum(nums)
9. div, mod = divmod(\_sum, 4)
10. **if** mod != 0 **or** max(nums) > \_sum / 4: **return** False
11. nums.sort(reverse = True)
12. target = [div] \* 4
13. **return** self.dfs(nums, 0, target)
15. **def** dfs(self, nums, index, target):
16. **if** index == len(nums): **return** True
17. num = nums[index]
18. **for** i **in** range(4):
19. **if** target[i] >= num:
20. target[i] -= num
21. **if** self.dfs(nums, index + 1, target): **return** True
22. target[i] += num
23. **return** False

491递增子序列

题目：

给定一个整型数组, 你的任务是找到所有该数组的递增子序列，递增子序列的长度至少是2。

示例:

输入: [4, 6, 7, 7]

输出: [[4, 6], [4, 7], [4, 6, 7], [4, 6, 7, 7], [6, 7], [6, 7, 7], [7,7], [4,7,7]]

说明:

给定数组的长度不会超过15。

数组中的整数范围是 [-100,100]。

给定数组中可能包含重复数字，相等的数字应该被视为递增的一种情况。

思路：

代码：

1. **class** Solution(object):
2. **def** findSubsequences(self, nums):
3. """
4. :type nums: List[int]
5. :rtype: List[List[int]]
6. """
7. res = []
8. self.dfs(nums, 0, res, [])
9. **return** res
11. **def** dfs(self, nums, index, res, path):
12. **if** len(path) >= 2 **and** path **not** **in** res:
13. res.append(path)
14. **for** i **in** range(index, len(nums)):
15. **if** **not** path **or** path[-1] <= nums[i]:
16. path += [nums[i]]
17. self.dfs(nums, i + 1, res, path)
18. path.pop()