Leetcode(Python) 二叉搜索树 中等

220存在重复元素III

题目：

给定一个整数数组，判断数组中是否有两个不同的索引 *i* 和 *j*，使得 **nums [i]** 和 **nums [j]** 的差的绝对值最大为 *t*，并且 *i* 和 *j* 之间的差的绝对值最大为 *ķ*。

**示例 1:**

**输入:** nums = [1,2,3,1], k= 3, t = 0

**输出:** true

**示例 2:**

**输入:** nums = [1,0,1,1], k=1, t = 2

**输出:** true

**示例 3:**

**输入:** nums = [1,5,9,1,5,9], k = 2, t = 3

**输出:** false

思路：因为没有重复，所以不能用hash。坐标之差在一个范围内，我们可以每次只在这个范围内寻找数，比如：从左往右移动，当长度大于k，那么就把搜索区间的最左边删除

代码：

1. **class** Solution(object):
2. **def** containsNearbyAlmostDuplicate(self, nums, k, t):
3. """
4. :type nums: List[int]
5. :type k: int
6. :type t: int
7. :rtype: bool
8. """
9. lenth=len(nums)
10. a=set()
11. **for** i **in** range(lenth):
12. **if** t==0:
13. **if** nums[i] **in** a:
14. **return** True
15. **else**:
16. **for** atem **in** a:
17. **if** abs(nums[i]-atem)<=t:
18. **return** True
19. a.add(nums[i])
21. **if** len(a)==k+1:
22. a.remove(nums[i-k])
23. **return** False

731我的日程安排表II

题目：

实现一个 MyCalendar 类来存放你的日程安排。如果要添加的时间内不会导致三重预订时，则可以存储这个新的日程安排。

MyCalendar 有一个 book(int start, int end)方法。它意味着在start到end时间内增加一个日程安排，注意，这里的时间是半开区间，即 [start, end), 实数 x 的范围为，  start <= x < end。

当三个日程安排有一些时间上的交叉时（例如三个日程安排都在同一时间内），就会产生三重预订。

每次调用 MyCalendar.book方法时，如果可以将日程安排成功添加到日历中而不会导致三重预订，返回 true。否则，返回 false 并且不要将该日程安排添加到日历中。

请按照以下步骤调用MyCalendar 类: MyCalendar cal = new MyCalendar(); MyCalendar.book(start, end)

**示例 1:**

MyCalendar();

MyCalendar.book(10, 20); // returns true

MyCalendar.book(50, 60); // returns true

MyCalendar.book(10, 40); // returns true

MyCalendar.book(5, 15); // returns false

MyCalendar.book(5, 10); // returns true

MyCalendar.book(25, 55); // returns true

**解释:**

前两个日程安排可以添加至日历中。 第三个日程安排会导致双重预订，但可以添加至日历中。

第四个日程安排活动（5,15）不能添加至日历中，因为它会导致三重预订。

第五个日程安排（5,10）可以添加至日历中，因为它未使用已经双重预订的时间10。

第六个日程安排（25,55）可以添加至日历中，因为时间 [25,40] 将和第三个日程安排双重预订；

时间 [40,50] 将单独预订，时间 [50,55）将和第二个日程安排双重预订。

**说明:**

* 每个测试用例，调用 MyCalendar.book 函数最多不超过 100次。
* 调用函数 MyCalendar.book(start, end)时， start 和 end 的取值范围为 [0, 10^9]。

思路：使用两个list分别保存已经预定了的区间和已经重叠了的区间。（深刻理解你定义的东西是什么，只有理解了才能做题）

判断区间重叠的方式是两个区间的起点的最大值 < 两个区间的终点的最小值。

那么，当一个新的区间到达的时候，先要对已经重叠的区间进行一个遍历，如果发现重叠，那么直接失败。

否则代表能添加成功，添加成功的含义是，当前区间不和任何overlaped里区间有交集。所以，更新已经重叠的区间的方式是找到和当前区间重叠的区间，然后添加到overlaped里。最后把当前区间放到booked里。

注意，在寻找新的重叠区间的时候，需要对所有的已经预定的区间进行遍历，也就是说我们当前的这个区间有可能和多个已经预定的区间重叠，要把多个重叠区间都放到overlaped里去。

时间复杂度是O(N^2)，空间复杂度是O(N).

代码：

1. **class** MyCalendarTwo(object):
3. **def** \_\_init\_\_(self):
4. self.booked=list()
5. self.overlaped=list()
7. **def** book(self, start, end):
8. """
9. :type start: int
10. :type end: int
11. :rtype: bool
12. """
13. **for** os, oe **in** self.overlaped:
14. **if** max(os, start) < min(oe, end):
15. **return** False
16. **for** bs, be **in** self.booked:
17. ss = max(bs, start)
18. ee = min(be, end)
19. **if** ss < ee:
20. self.overlaped.append((ss, ee))
21. self.booked.append((start, end))
22. **return** True


26. # Your MyCalendarTwo object will be instantiated and called as such:
27. # obj = MyCalendarTwo()
28. # param\_1 = obj.book(start,end)

938二叉搜索树的范围和

题目：

给定二叉搜索树的根结点 root，返回 L 和 R（含）之间的所有结点的值的和。

二叉搜索树保证具有唯一的值。

**示例 1：**

**输入：**root = [10,5,15,3,7,null,18], L = 7, R = 15

**输出：**32

**示例 2：**

**输入：**root = [10,5,15,3,7,13,18,1,null,6], L = 6, R = 10

**输出：**23

**提示：**

1. 树中的结点数量最多为 10000 个。
2. 最终的答案保证小于 2^31。

思路：看见BST，就想起来它特殊的性质。所以这个题肯定能用上性质。

如果root不存在，返回0。如果root节点在[L,R]内，那么把结果加上root的值，然后再分别加上左右子树的值。为什么？因为这个时候左右子树都可能存在满足[L,R]区间，所以必须都加上。

如果root的值比L还小，说明左子树一定不会满足[L,R]区间，那么直接向右边找就行。

如果root的值比R还大，说明右子树一定不会满足[L,R]区间，那么直接向左边找就行。

时间复杂度是O(N)，空间复杂度是O(1)。

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** rangeSumBST(self, root, L, R):
10. """
11. :type root: TreeNode
12. :type L: int
13. :type R: int
14. :rtype: int
15. """
16. res=[0]
17. self.dfs(root,L,R,res)
18. **return** res[0]
19. **def** dfs(self,root,L,R,res):
20. **if** **not** root:
21. **return**
22. **if** L<=root.val<=R:
23. res[0]+=root.val
24. **if** root.val<R:
25. self.dfs(root.right,L,R,res)
26. **if** root.val>L:
27. self.dfs(root.left,L,R,res)