669 Trim a Binary Search Tree

2018年4月22日 16:28

Given a binary search tree and the lowest and highest boundaries as L and R, trim the tree so that all its elements lies in [L, R] (R >= L). You might need to change the root of the tree, so the result should return the new root of the trimmed binary search tree.

Example 1:

Input:

```
    1
   / \
  0   2
L = 1
  R = 2
```

Output:

```
    1
     \
      2
```

Example 2:

Input:

```
    3
   / \
  0   4
   \
    2
   /
  1
L = 1
  R = 3
```

Output:

```
      3
     /
   2
  /
 1
```

来自 <https://leetcode.com/problems/trim-a-binary-search-tree/description/>

给定一个二叉搜索树，同时给定最小边界L 和最大边界 R。通过修剪二叉搜索树，使得所有节点的值在[L, R]中 (R>=L) 。你可能需要改变树的根节点，所以结果应当返回修剪好的二叉搜索树的新的根节点。

Solution for Python3：

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution1:
    def trimBST(self, root, L, R):
        """
        :type root: TreeNode
        :type L: int
        :type R: int
        :rtype: TreeNode
        """
        if not root:
            return root
        if root.val < L:
            return self.trimBST(root.right, L, R)
        if root.val > R:
            return self.trimBST(root.left, L, R)
        root.left = self.trimBST(root.left, L, R)
        root.right = self.trimBST(root.right, L, R)
        return root

class Solution2:
    def trimBST(self, root, L, R):
        """
        :type root: TreeNode
        :type L: int
        :type R: int
        :rtype: TreeNode
        """
        while root.val < L or root.val > R:
            if root.val < L:
                root = root.right
            else:
                root = root.left
        Lnode, Rnode = root, root
        while Lnode.left:
            if Lnode.left.val < L:
                Lnode.left = Lnode.left.right
            else:
                Lnode = Lnode.left
```

```
45          while Rnode.right:
46              if Rnode.right.val > R:
47                  Rnode.right = Rnode.right.left
48              else:
49                  Rnode = Rnode.right
50          return root
```

Solution for C++:

```
1   /**
2    * Definition for a binary tree node.
3    * struct TreeNode {
4    *     int val;
5    *     TreeNode *left;
6    *     TreeNode *right;
7    *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
8    * };
9    */
10  class Solution1 {
11  public:
12      TreeNode* trimBST(TreeNode* root, int L, int R) {
13          if (!root)
14              return root;
15          if (root->val < L)
16              return trimBST(root->right, L, R);
17          if (root->val > R)
18              return trimBST(root->left, L, R);
19          root->left = trimBST(root->left, L, R);
20          root->right = trimBST(root->right, L, R);
21          return root;
22      }
23  };
24
25  class Solution2 {
26  public:
27      TreeNode* trimBST(TreeNode* root, int L, int R) {
28          // find the true root
29          while (root->val < L || root->val > R) {
30              if (root->val < L)
31                  root = root->right;
32              else
33                  root = root->left;
34          }
35          TreeNode* Ltmp = root;
```

```
36          TreeNode* Rtmp = root;
37          // remove the element lower than L
38          while (Ltmp->left) {
39              if (Ltmp->left->val < L)
40                  Ltmp->left = Ltmp->left->right;
41              else
42                  Ltmp = Ltmp->left;
43          }
44          // remove the element larger than R
45          while (Rtmp->right) {
46              if (Rtmp->right->val > R)
47                  Rtmp->right = Rtmp->right->left;
48              else
49                  Rtmp = Rtmp->right;
50          }
51          return root;
52      }
53  };
```