Trim a Binary Search Tree

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
```

Solution 1

```
class Solution {
    public TreeNode trimBST(TreeNode root, int L, int R) {
        if (root == null) return null;

        if (root.val < L) return trimBST(root.right, L, R);
        if (root.val > R) return trimBST(root.left, L, R);

        root.left = trimBST(root.left, L, R);
        root.right = trimBST(root.right, L, R);

        return root;
    }
}
```

written by shawngao original link here

Solution 2

The code works as recursion.

```
If the root value in the range [L, R]
      we need return the root, but trim its left and right subtree;
else if the root value < L
      because of binary search tree property, the root and the left subtree are not
in range;
      we need return trimmed right subtree.
else
      similarly we need return trimmed left subtree.
```

Without freeing memory

```
class Solution {
public:
    TreeNode* trimBST(TreeNode* root, int L, int R) {
        if (root == NULL) return NULL;
        if (root->val < L) return trimBST(root->right, L, R);
        if (root->val > R) return trimBST(root->left, L, R);
        root->left = trimBST(root->left, L, R);
        root->right = trimBST(root->right, L, R);
        return root;
    }
};
```

Free the memory
As @StefanPochmann pointed out, it works well to delete only non-root nodes of the whole tree. His solution is as below. Thanks.

```
TreeNode* trimBST(TreeNode* root, int L, int R, bool top=true) {
    if (!root)
        return root;
    root->left = trimBST(root->left, L, R, false);
    root->right = trimBST(root->right, L, R, false);
    if (root->val >= L && root->val <= R)
        return root;
    auto result = root->val < L ? root->right : root->left;
    if (!top)
        delete root;
    return result;
}
```

written by zestypanda original link here

Solution 3

```
public TreeNode trimBST(TreeNode root, int L, int R) {
        if (root == null) {
            return root;
        }

        if (root.val > R) {
            return trimBST(root.left, L, R);
        }

        if (root.val < L) {
            return trimBST(root.right, L, R);
        }

        root.left = trimBST(root.left, L, R);
        root.right = trimBST(root.right, L, R);
        return root;
    }
```

Also viewable here on Github.

written by fishercoder original link here