Author : Xiaoqiang Yan

LeetCode 235 Lowest Common Ancestor of a Binary Search Tree

Given a binary search tree (BST), find the lowest common ancestor (LCA) of two given nodes in the BST.

According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow a node to be a descendant of itself).”

Given binary search tree: root = [6,2,8,0,4,7,9,null,null,3,5]

```
        _______6______
       /              \
    ___2__          ___8__
   /      \        /      \
   0      _4       7       9
         /  \
         3   5
```

Example 1:

```
Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8
Output: 6
Explanation: The LCA of nodes 2 and 8 is 6.
```

Example 2:

```
Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4
Output: 2
Explanation: The LCA of nodes 2 and 4 is 2, since a node can
be a descendant of itself according to the LCA definition.
```

Note:

All of the nodes’ values will be unique.

p and q are different and both values will exist in the BST.

Solution

The two nodes which have LCA, have 2 cases: 1. two nodes are in the LCA’s left subtree and right subtree seperately; 2. the LCA is one of these two. Because of the note that each element has unique value and the features of BST, we can code in 2 cases.

Code

```
struct TreeNode {
    int val;
    TreeNode* left, right;
    TreeNode(int x): val(x), left(NULL), right(NULL) {}
};
TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, T
reeNode* q)
{
    if(p->val < q->val) return BS(root, p, q);
    else return BS(root, q, p);
}
TreeNode* BS(treeNode* root, TreeNode* p, TreeNode* q) {
    if(root==NULL) return NULL;
    if(root->val==p->val || root->val==q->val || (roo->val > 
p->val &&  root->val < q->val)
        return root;
    else if (root->val < p)
        return BS(root->right, p, q);
    else
        return BS(root->left, p,q);
}
```

Note

We need to check the value of TreeNode* p and *q firstly to make sure p->value < q->value.

Summary

Binary Search/ Recursion