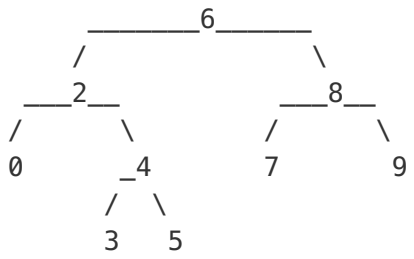
Lowest Common Ancestor of a Binary Search Tree

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 v and w as the lowest node in T that has both v and w as descendants (where we allow **a node to be a descendant of itself**).”



For example, the lowest common ancestor (LCA) of nodes `2` and `8` is `6`. Another example is LCA of nodes `2` and `4` is `2`, since a node can be a descendant of itself according to the LCA definition.

Solution 1

Just walk down from the whole tree's root as long as both p and q are in the same subtree (meaning their values are both smaller or both larger than root's). This walks straight from the root to the LCA, not looking at the rest of the tree, so it's pretty much as fast as it gets. A few ways to do it:

Iterative, O(1) space

Python

```
def lowestCommonAncestor(self, root, p, q):
    while (root.val - p.val) * (root.val - q.val) > 0:
        root = (root.left, root.right)[p.val > root.val]
    return root
```

Java

```
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
    while ((root.val - p.val) * (root.val - q.val) > 0)
        root = p.val < root.val ? root.left : root.right;
    return root;
}
```

(in case of overflow, I'd do `(root.val - (long)p.val) * (root.val - (long)q.val)`)

Different Python

```
def lowestCommonAncestor(self, root, p, q):
    a, b = sorted([p.val, q.val])
    while not a <= root.val <= b:
        root = (root.left, root.right)[a > root.val]
    return root
```

"Long" Python, maybe easiest to understand

```
def lowestCommonAncestor(self, root, p, q):
    while root:
        if p.val < root.val > q.val:
            root = root.left
        elif p.val > root.val < q.val:
            root = root.right
        else:
            return root
```

Recursive

Python

```
def lowestCommonAncestor(self, root, p, q):
    next = p.val < root.val > q.val and root.left or \
           p.val > root.val < q.val and root.right
    return self.lowestCommonAncestor(next, p, q) if next else root
```

Python One-Liner

```
def lowestCommonAncestor(self, root, p, q):
    return root if (root.val - p.val) * (root.val - q.val) < 1 else \
           self.lowestCommonAncestor((root.left, root.right)[p.val > root.val], p
, q)
```

Java One-Liner

```
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
    return (root.val - p.val) * (root.val - q.val) < 1 ? root :
           lowestCommonAncestor(p.val < root.val ? root.left : root.right, p, q);
}
```

"Long" Python, maybe easiest to understand

```
def lowestCommonAncestor(self, root, p, q):
    if p.val < root.val > q.val:
        return self.lowestCommonAncestor(root.left, p, q)
    if p.val > root.val < q.val:
        return self.lowestCommonAncestor(root.right, p, q)
    return root
```

written by StefanPochmann original link here

Solution 2

```
public class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        if(root.val > p.val && root.val > q.val){
            return lowestCommonAncestor(root.left, p, q);
        }else if(root.val < p.val && root.val < q.val){
            return lowestCommonAncestor(root.right, p, q);
        }else{
            return root;
        }
    }
}
```

written by jingzhetian original link here

Solution 3

Well, remember to take advantage of the property of binary search trees, which is, `node -> left -> val < node -> val < node -> right -> val`. Moreover, both `p` and `q` will be the descendants of the `root` of the subtree that contains both of them. And the `root` with the largest depth is just the lowest common ancestor. This idea can be turned into the following simple recursive code.

```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if (p -> val < root -> val && q -> val < root -> val)
            return lowestCommonAncestor(root -> left, p, q);
        if (p -> val > root -> val && q -> val > root -> val)
            return lowestCommonAncestor(root -> right, p, q);
        return root;
    }
};
```

Of course, we can also solve it iteratively.

```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        TreeNode* cur = root;
        while (true) {
            if (p -> val < cur -> val && q -> val < cur -> val)
                cur = cur -> left;
            else if (p -> val > cur -> val && q -> val > cur -> val)
                cur = cur -> right;
            else return cur;
        }
    }
};
```

written by jianchao.li.fighter original link here