Inorder Successor in BST

Given a binary search tree and a node in it, find the in-order successor of that node in the BST.

Note: If the given node has no in-order successor in the tree, return `null`.

Solution 1

Just want to share my recursive solution for both getting the successor and predecessor for a given node in BST.

Successor

```
public TreeNode successor(TreeNode root, TreeNode p) {
  if (root == null)
    return null;

  if (root.val <= p.val) {
    return successor(root.right, p);
  } else {
    TreeNode left = successor(root.left, p);
    return (left != null) ? left : root;
  }
}
```

Predecessor

```
public TreeNode predecessor(TreeNode root, TreeNode p) {
  if (root == null)
    return null;

  if (root.val >= p.val) {
    return predecessor(root.left, p);
  } else {
    TreeNode right = predecessor(root.right, p);
    return (right != null) ? right : root;
  }
}
```

written by jeantimex original link here

Solution 2

The inorder traversal of a BST is the nodes in ascending order. To find a successor, you just need to find the smallest one that is larger than the given value since there are no duplicate values in a BST. It just like the binary search in a sorted list. The time complexity should be `O(h)` where h is the depth of the result node. `succ` is a pointer that keeps the possible successor. Whenever you go left the current root is the new possible successor, otherwise the it remains the same.

Only in a balanced BST `O(h) = O(log n)`. In the worst case `h` can be as large as `n`.

Java

```java
public TreeNode inorderSuccessor(TreeNode root, TreeNode p) {
    TreeNode succ = null;
    while (root != null) {
        if (p.val < root.val) {
            succ = root;
            root = root.left;
        }
        else
            root = root.right;
    }
    return succ;
}

// 29 / 29 test cases passed.
// Status: Accepted
// Runtime: 5 ms
```

Python

```python
def inorderSuccessor(self, root, p):
    succ = None
    while root:
        if p.val < root.val:
            succ = root
            root = root.left
        else:
            root = root.right
    return succ

# 29 / 29 test cases passed.
# Status: Accepted
# Runtime: 112 ms
```

written by dietpepsi original link here

Solution 3

Update: Ugh, turns out I didn't think it through and the big case distinction is unnecessary. Just search from root to bottom, trying to find the smallest node larger than p and return the last one that was larger. D'oh. Props to smileyogurt.966 for doing that first, I think. I'll just write it in my ternary style for C++:

```
TreeNode* inorderSuccessor(TreeNode* root, TreeNode* p) {
    TreeNode* candidate = NULL;
    while (root)
        root = (root->val > p->val) ? (candidate = root)->left : root->right;
    return candidate;
}
```

Old: If `p` has a right subtree, then get its successor from there. Otherwise do a regular search from `root` to `p` but remember the node of the last left-turn and return that. Same solution as everyone, I guess, just written a bit shorter. Runtime O(h), where h is the height of the tree.

C++

```
TreeNode* inorderSuccessor(TreeNode* root, TreeNode* p) {
    if (p->right) {
        p = p->right;
        while (p->left)
            p = p->left;
        return p;
    }
    TreeNode* candidate = NULL;
    while (root != p)
        root = (p->val > root->val) ? root->right : (candidate = root)->left;
    return candidate;
}
```

Java

```
public TreeNode inorderSuccessor(TreeNode root, TreeNode p) {
    if (p.right != null) {
        p = p.right;
        while (p.left != null)
            p = p.left;
        return p;
    }
    TreeNode candidate = null;
    while (root != p)
        root = (p.val > root.val) ? root.right : (candidate = root).left;
    return candidate;
}
```

written by StefanPochmann original link here