Find Mode in Binary Search Tree

Given a binary search tree (BST) with duplicates, find all themode(s) (the most frequently occurred element) in the given BST.

Assume a BST is defined as follows:

- The left subtree of a node contains only nodes with keys**less than or equal to** the node's key.
- The right subtree of a node contains only nodes with keys**greater than or equal to** the node's key.
- Both the left and right subtrees must also be binary search trees.

For example:
Given BST `[1,null,2,2]`,

```
   1
    \
     2
    /
   2
```

return `[2]`.

Note: If a tree has more than one mode, you can return them in any order.

Follow up: Could you do that without using any extra space? (Assume that the implicit stack space incurred due to recursion does not count).

Solution 1

I've seen several solutions claimed to be O(1) space, but I disagree. They traverse the tree in in-order and keep track of the current set of modes (among other things). But that's not O(1) space, not even when disregarding recursion stack space (as explicitly allowed) and result space (not mentioned but reasonable). The set's contents aren't on stack space, so it can't be disregarded that way. And if the values are for example 1,2,3,4,...,n-2,n-1,n-1 (unique values followed by one double value), the set grows to Ω(n) and it can't be disregarded because the result only has size 1.

I think the way to do it properly is to do two passes. One to find the highest number of occurrences of any value, and then a second pass to collect all values occurring that often. Any other ideas?

Here's a (two-pass) solution that I think can rightfully be called O(1) space. Both passes keep track of the current value etc, and the second pass additionally collects the modes in the result array. I took the value handling out of the in-order traversal into its own function for clarity. Also, this way you could very easily replace my recursive in-order traversal with for example Morris traversal. Then you wouldn't even need to disregard the recursion stack space in order to claim O(1) extra space usage.

```java
public class Solution {

    public int[] findMode(TreeNode root) {
        inorder(root);
        modes = new int[modeCount];
        modeCount = 0;
        currCount = 0;
        inorder(root);
        return modes;
    }

    private int currVal;
    private int currCount = 0;
    private int maxCount = 0;
    private int modeCount = 0;

    private int[] modes;

    private void handleValue(int val) {
        if (val != currVal) {
            currVal = val;
            currCount = 0;
        }
        currCount++;
        if (currCount > maxCount) {
            maxCount = currCount;
            modeCount = 1;
        } else if (currCount == maxCount) {
            if (modes != null)
                modes[modeCount] = currVal;
            modeCount++;
        }
    }

    private void inorder(TreeNode root) {
        if (root == null) return;
        inorder(root.left);
        handleValue(root.val);
        inorder(root.right);
    }
}
```

Edit: Here's Morris traversal, just replace my previous `inorder` function with this. I hadn't realized it earlier, but having my separate `handleValue` function doesn't just nicely separate the traversal logic from this problem's logic, but it's also beneficial for Morris traversal because I'm calling `handleValue` from two different places in the code!

```
    private void inorder(TreeNode root) {
        TreeNode node = root;
        while (node != null) {
            if (node.left == null) {
                handleValue(node.val);
                node = node.right;
            } else {
                TreeNode prev = node.left;
                while (prev.right != null && prev.right != node)
                    prev = prev.right;
                if (prev.right == null) {
                    prev.right = node;
                    node = node.left;
                } else {
                    prev.right = null;
                    handleValue(node.val);
                    node = node.right;
                }
            }
        }
    }
```

written by StefanPochmann original link here

Solution 2

What does "mode" mean?

written by ycc0602 original link here

Solution 3

O(n) time O(n) space

```
public class Solution {
    Map<Integer, Integer> map;
    int max = 0;
    public int[] findMode(TreeNode root) {
        if(root==null) return new int[0];
        this.map = new HashMap<>();

        inorder(root);

        List<Integer> list = new LinkedList<>();
        for(int key: map.keySet()){
            if(map.get(key) == max) list.add(key);
        }

        int[] res = new int[list.size()];
        for(int i = 0; i<res.length; i++) res[i] = list.get(i);
        return res;
    }

    private void inorder(TreeNode node){
        if(node.left!=null) inorder(node.left);
        map.put(node.val, map.getOrDefault(node.val, 0)+1);
        max = Math.max(max, map.get(node.val));
        if(node.right!=null) inorder(node.right);
    }
}
```

Just travel the tree and count, find the those with max counts. Nothing much. Spent 10min on figuring out what is mode....

If using this method (hashmap), inorder/preorder/postorder gives the same result. Because essentially you just travel the entire nodes and count. And BST is not necessary. This method works for any tree.

written by Chidong original link here