Recover Binary Search Tree

Two elements of a binary search tree (BST) are swapped by mistake.

Recover the tree without changing its structure.

Note:
A solution using O(n) space is pretty straight forward. Could you devise a constant space solution?
confused what `"{1,#,2,3}"` means? > read more on how binary tree is serialized on OJ.

OJ's Binary Tree Serialization:
The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

```
   1
  / \
 2   3
    /
   4
    \
     5
```

The above binary tree is serialized as `"{1,2,3,#,#,4,#,#,5}"`.

Solution 1

This question appeared difficult to me but it is really just a simple in-order traversal! I got really frustrated when other people are showing off Morris Traversal which is totally not necessary here.

Let's start by writing the in order traversal:

```
private void traverse (TreeNode root) {
   if (root == null)
      return;
   traverse(root.left);
   // Do some business
   traverse(root.right);
}
```

So when we need to print the node values in order, we insert System.out.println(root.val) in the place of "Do some business".

What is the business we are doing here? We need to find the first and second elements that are not in order right?

How do we find these two elements? For example, we have the following tree that is printed as in order traversal:

6, 3, 4, 5, 2

We compare each node with its next one and we can find out that 6 is the first element to swap because 6 > 3 and 2 is the second element to swap because 2 < 5.

Really, what we are comparing is the current node and its previous node in the "in order traversal".

Let us define three variables, firstElement, secondElement, and prevElement. Now we just need to build the "do some business" logic as finding the two elements. See the code below:

```
public class Solution {

    TreeNode firstElement = null;
    TreeNode secondElement = null;
    // The reason for this initialization is to avoid null pointer exception in th
e first comparison when prevElement has not been initialized
    TreeNode prevElement = new TreeNode(Integer.MIN_VALUE);

    public void recoverTree(TreeNode root) {

        // In order traversal to find the two elements
        traverse(root);

        // Swap the values of the two nodes
        int temp = firstElement.val;
        firstElement.val = secondElement.val;
        secondElement.val = temp;
    }

    private void traverse(TreeNode root) {

        if (root == null)
            return;

        traverse(root.left);

        // Start of "do some business",
        // If first element has not been found, assign it to prevElement (refer t
o 6 in the example above)
        if (firstElement == null && prevElement.val >= root.val) {
            firstElement = prevElement;
        }

        // If first element is found, assign the second element to the root (refe
r to 2 in the example above)
        if (firstElement != null && prevElement.val >= root.val) {
            secondElement = root;
        }
        prevElement = root;

        // End of "do some business"

        traverse(root.right);
}
```

And we are done, it is just that easy!

written by qwl5004 original link here

Solution 2

To understand this, you need to first understand Morris Traversal or Morris Threading Traversal. It take use of leaf nodes' right/left pointer to achieve O(1) space Traversal on a Binary Tree. Below is a standard Inorder Morris Traversal, referred from http://www.cnblogs.com/AnnieKim/archive/2013/06/15/morristraversal.html (a Chinese Blog, while the graphs are great for illustration)

```java
public void morrisTraversal(TreeNode root){
        TreeNode temp = null;
        while(root!=null){
            if(root.left!=null){
                // connect threading for root
                temp = root.left;
                while(temp.right!=null && temp.right != root)
                    temp = temp.right;
                // the threading already exists
                if(temp.right!=null){
                    temp.right = null;
                    System.out.println(root.val);
                    root = root.right;
                }else{
                    // construct the threading
                    temp.right = root;
                    root = root.left;
                }
            }else{
                System.out.println(root.val);
                root = root.right;
            }
        }
    }
```

In the above code, `System.out.println(root.val);` appear twice, which functions as outputing the Node in ascending order (BST). Since these places are in order, replace them with

```java
    if(pre!=null && pre.val > root.val){
        if(first==null){first = pre;second = root;}
        else{second = root;}
  }
pre = root;
```

each time, the pre node and root are in order as `System.out.println(root.val);` outputs them in order.

Then, come to how to specify the first wrong node and second wrong node.

When they are not consecutive, the first time we meet `pre.val > root.val` ensure us the first node is the pre node, since root should be traversal ahead of pre, pre should be at least at small as root. The second time we meet `pre.val > root.val` ensure us the second node is the root node, since we are now looking for a node to replace with out first node, which is found before.

When they are consecutive, which means the case `pre.val > cur.val` will appear only once. We need to take case this case without destroy the previous analysis. So the first node will still be pre, and the second will be just set to root. Once we meet this case again, the first node will not be affected.

Below is the updated version on Morris Traversal.

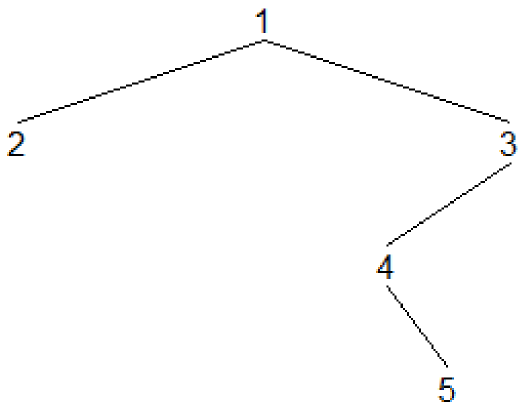
```
public void recoverTree(TreeNode root) {
        TreeNode pre = null;
        TreeNode first = null, second = null;
        // Morris Traversal
        TreeNode temp = null;
        while(root!=null){
            if(root.left!=null){
                // connect threading for root
                temp = root.left;
                while(temp.right!=null && temp.right != root)
                    temp = temp.right;
                // the threading already exists
                if(temp.right!=null){
                    if(pre!=null && pre.val > root.val){
                        if(first==null){first = pre;second = root;}
                        else{second = root;}
                    }
                    pre = root;

                    temp.right = null;
                    root = root.right;
                }else{
                    // construct the threading
                    temp.right = root;
                    root = root.left;
                }
            }else{
                if(pre!=null && pre.val > root.val){
                    if(first==null){first = pre;second = root;}
                    else{second = root;}
                }
                pre = root;
                root = root.right;
            }
        }
        // swap two node values;
        if(first!= null && second != null){
            int t = first.val;
            first.val = second.val;
            second.val = t;
        }
    }
```

written by siyang3 original link here

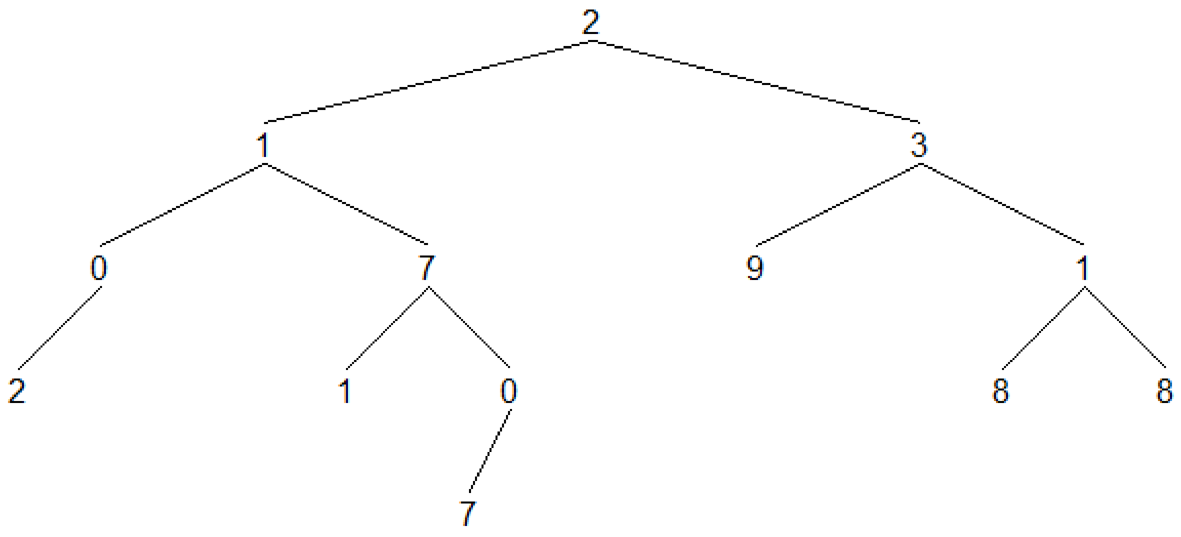
Solution 3

Wrote some tools for my own local testing. For example `deserialize('[1,2,3,null,null,4,null,null,5]')` will turn that into a tree and return the root as explained in the FAQ. I also wrote a visualizer. Two examples:

`drawtree(deserialize('[1,2,3,null,null,4,null,null,5]'))`:



`drawtree(deserialize('[2,1,3,0,7,9,1,2,null,1,0,null,null,8,8,null,null,null,null,7]'))`:



Here's the code. If you save it as a Python script and run it, it should as a demo show the above two pictures in turtle windows (one after the other). And you can of course import it from other scripts and then it will only provide the class/functions and not show the demo.

```python
class TreeNode:
    def __init__(self, val, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
    def __repr__(self):
        return 'TreeNode({})'.format(self.val)

def deserialize(string):
    if string == '{}':
        return None
    nodes = [None if val == 'null' else TreeNode(int(val))
             for val in string.strip('[]{}').split(',')]
    kids = nodes[::-1]
    root = kids.pop()
    for node in nodes:
        if node:
            if kids: node.left  = kids.pop()
            if kids: node.right = kids.pop()
    return root

def drawtree(root):
    def height(root):
        return 1 + max(height(root.left), height(root.right)) if root else -1
    def jumpto(x, y):
        t.penup()
        t.goto(x, y)
        t.pendown()
    def draw(node, x, y, dx):
        if node:
            t.goto(x, y)
            jumpto(x, y-20)
            t.write(node.val, align='center', font=('Arial', 12, 'normal'))
            draw(node.left, x-dx, y-60, dx/2)
            jumpto(x, y-20)
            draw(node.right, x+dx, y-60, dx/2)
    import turtle
    t = turtle.Turtle()
    t.speed(0); turtle.delay(0)
    h = height(root)
    jumpto(0, 30*h)
    draw(root, 0, 30*h, 40*h)
    t.hideturtle()
    turtle.mainloop()

if __name__ == '__main__':
    drawtree(deserialize('[1,2,3,null,null,4,null,null,5]'))
    drawtree(deserialize('[2,1,3,0,7,9,1,2,null,1,0,null,null,8,8,null,null,null,
null,7]'))
```

written by StefanPochmann original link here