Construct String from Binary Tree

You need to construct a string consists of parenthesis and integers from a binary tree with the preorder traversing way.

The null node needs to be represented by empty parenthesis pair "()". And you need to omit all the empty parenthesis pairs that don't affect the one-to-one mapping relationship between the string and the original binary tree.

Example 1:

```
Input: Binary tree: [1,2,3,4]
       1
     /   \
    2     3
   /
  4

Output: "1(2(4))(3)"

Explanation: Originallay it needs to be "1(2(4)())(3()())",
but you need to omit all the unnecessary empty parenthesis pairs.
And it will be "1(2(4))(3)".
```

Example 2:

```
Input: Binary tree: [1,2,3,null,4]
       1
     /   \
    2     3
     \
      4

Output: "1(2()(4))(3)"

Explanation: Almost the same as the first example,
except we can't omit the first parenthesis pair to break the one-to-one mapping relat
ionship between the input and the output.
```

Solution 1

```
public class Solution {
    public String tree2str(TreeNode t) {
        if (t == null) return "";

        String result = t.val + "";

        String left = tree2str(t.left);
        String right = tree2str(t.right);

        if (left == "" && right == "") return result;
        if (left == "") return result + "()" + "(" + right + ")";
        if (right == "") return result + "(" + left + ")";
        return result + "(" + left + ")" + "(" + right + ")";
    }
}
```

written by shawngao original link here

Solution 2

```
public String tree2str(TreeNode t) {
       StringBuilder sb = new StringBuilder();
       helper(sb,t);
       return sb.toString();
   }
   public void helper(StringBuilder sb,TreeNode t){
       if(t!=null){
           sb.append(t.val);
           if(t.left!=null||t.right!=null){
               sb.append("(");
               helper(sb,t.left);
               sb.append(")");
               if(t.right!=null){
                   sb.append("(");
               helper(sb,t.right);
               sb.append(")");
               }
           }
       }
   }
```

written by dwyanecf original link here

Solution 3

We do this recursively.

- If the tree is empty, we return an empty string.
- We record each child as '(' + (string of child) + ')'
- If there is a right child but no left child, we still need to record '()' instead of empty string.

```
def tree2str(self, t):
    if not t: return ''
    left = '({})'.format(self.tree2str(t.left)) if (t.left or t.right) else ''
    right = '({})'.format(self.tree2str(t.right)) if t.right else ''
    return '{}{}{}'.format(t.val, left, right)
```

written by awice original link here