Merge Two Binary Trees

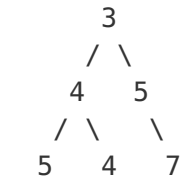
Given two binary trees and imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not.

You need to merge them into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of new tree.

Example 1:

Input:

Output:



Note: The merging process must start from the root nodes of both trees.

Solution 1

```
public class Solution {
    public TreeNode mergeTrees(TreeNode t1, TreeNode t2) {
        if (t1 == null && t2 == null) return null;

        int val = (t1 == null ? 0 : t1.val) + (t2 == null ? 0 : t2.val);
        TreeNode newNode = new TreeNode(val);

        newNode.left = mergeTrees(t1 == null ? null : t1.left, t2 == null ? null : t
2.left);
        newNode.right = mergeTrees(t1 == null ? null : t1.right, t2 == null ? null :
t2.right);

        return newNode;
    }
}
```

written by shawngao original link here

Solution 2

Create new Nodes

I like to create new nodes for newly formed tree in this type of problem, as you are literally creating nested graph otherwise.

```
class Solution {
public:
    TreeNode* mergeTrees(TreeNode* t1, TreeNode* t2) {
        if (!t1 && !t2) {
            return nullptr;
        }
        TreeNode* node = new TreeNode((t1 ? t1->val : 0) + (t2 ? t2->val : 0));
        node->left = mergeTrees((t1 ? t1->left : nullptr), (t2 ? t2->left : nullptr)
);
        node->right = mergeTrees((t1 ? t1->right : nullptr), (t2 ? t2->right : null
ptr));
        return node;
    }
};
```

Share Nodes with the nonnull TreeNode

As @zqfan point out, this problem explicitly tell you to use the NOT null node, there is no need to create new nodes. And the code would also be simpler.

```
class Solution {
public:
    TreeNode* mergeTrees(TreeNode* t1, TreeNode* t2) {
        if (!t1) return t2;
        if (!t2) return t1;

        TreeNode* node = new TreeNode(t1->val + t2->val);
        node->left = mergeTrees(t1->left, t2->left);
        node->right = mergeTrees(t1->right, t2->right);
        return node;
    }
};
```

written by alexander original link here

Solution 3

python solution

```python
class Solution(object):
    def mergeTrees(self, t1, t2):
        if t1 and t2:
            root = TreeNode(t1.val + t2.val)
            root.left = self.mergeTrees(t1.left, t2.left)
            root.right = self.mergeTrees(t1.right, t2.right)
            return root
        else:
            return t1 or t2
```

c++ solution

```cpp
class Solution {
public:
    TreeNode* mergeTrees(TreeNode* t1, TreeNode* t2) {
        if ( t1 && t2 ) {
            TreeNode * root = new TreeNode(t1->val + t2->val);
            root->left = mergeTrees(t1->left, t2->left);
            root->right = mergeTrees(t1->right, t2->right);
            return root;
        } else {
            return t1 ? t1 : t2;
        }
    }
};
```

written by zqfan original link here