107 Binary Tree Level Order Traversal II

2018年3月30日 19:34

Question:

Given a binary tree, return the *bottom-up level order* traversal of its nodes' values. (ie, from left to right, level by level from leaf to root).
For example:
Given binary tree [3,9,20,null,null,15,7],

```
   3
  / \
 9  20
   / \
  15  7
```

return its bottom-up level order traversal as:

```
[
  [15,7],
  [9,20],
  [3]
]
```

来自 <https://leetcode.com/problems/binary-tree-level-order-traversal-ii/description/>

给定一个二叉树，返回其节点值自底向上的层次遍历。 （即按从叶节点所在层到根节点所在的层，逐层从左向右遍历）

Solution for Python3:

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
# Iterative Version
class Solution1:
    def levelOrderBottom(self, root):
        """
        :type root: TreeNode
        :rtype: List[List[int]]
        """
        if not root:
            return []
        from collections import deque
        d = deque()
        d.append(root)
        res = []
        while d:
            L = []
            for i in range(len(d)):
                node = d.popleft()
                L += [node.val]
                if node.left:
                    d.append(node.left)
                if node.right:
```

```
                    d.append(node.right)
            res.append(L)
        res.reverse()
        return res

# Recursive Version
class Solution2:
    def levelOrderBottom(self, root):
        """
        :type root: TreeNode
        :rtype: List[List[int]]
        """
        res = []
        self.levelOrder(root, res, 0)
        res.reverse()
        return res
    def levelOrder(self, root, res, curLevel):
        if not root:
            return
        if curLevel > len(res) - 1:
            res.append([])
        res[curLevel].append(root.val)
        self.levelOrder(root.left, res, curLevel + 1)
        self.levelOrder(root.right, res, curLevel + 1)
```

Solution for C++:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
// Iterative Version
class Solution1 {
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        if (!root) {
           return vector<vector<int> > ();
        }
        queue<TreeNode*> q;
        q.push(root);
        vector<std::vector<int> > res;
        vector<int> v;
        while (!q.empty()) {
           v.clear();
           for (int i = 0, n = q.size(); i < n; i++) {
               TreeNode *node = q.front();
               q.pop();
               v.push_back(node->val);
```

```
                if (node->left) {
                    q.push(node->left);
                }
                if (node->right) {
                    q.push(node->right);
                }
            }
            res.push_back(v);
        }
        reverse(res.begin(), res.end());
        return res;
    }
};

// Recursive Version
class Solution2 {
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        vector<vector<int> > res;
        levelOrder(root, res, 0);
        reverse(res.begin(), res.end());
        return res;
    }

    void levelOrder(TreeNode* root, vector<vector<int> > &v, int curLevel) {
        if (!root) {
            return;
        }
        if (v.empty() ||curLevel > (v.size() - 1)) {
            v.push_back(vector<int> ());
        }
        v[curLevel].push_back(root->val);
        levelOrder(root->left, v, curLevel + 1);
        levelOrder(root->right, v, curLevel + 1);
    }
};
```

Appendix:

Python：list.reverse():

在list上反转list,不返回。

Python的传值和传址:

1) python不允许程序员选择传值还是传址。python参数传递采用的是传对象引用的方式。这种方式相当于传值和传址的一种综合。如果函数收到一个可变对象（比如字典或者列表）的引用，就能修改对象的原始值——相当于传址。如果函数收到的是一个不可变对象（比如数字、字符或者元组）的引用，就不能直接修改原始对象——相当于传值。
2) 所以python的传值和传址是根据传入参数的类型来选择的。
   a. 传值参数类型：数字，字符串，元祖。
   b. 传址参数类型：列表，字典。

Python的copy和deepcopy

1) 想新建一个与当前变量a相等的变量b，同时b的值与a没有关联就需要用到copy。

```
import copy

a=[1,2,3]
b=a
a.append(4)
print a,b
#输出：[1, 2, 3, 4] [1, 2, 3, 4]

a=[1,2,3]
b=copy.copy(a)
a.append(4)
print a,b
#输出：[1, 2, 3, 4] [1, 2, 3]
```

这里用了copy来让b与a相等，后面如果修改了a的值，b的值并不会改变。看来copy已经可以实现我们上面的提到的需求了，那么deepcopy又有什么用?

2) 如果我们遇到这种情况，copy就解决不了了

```
a=[1,[1,2],3]
b=copy.copy(a)
a[1].append(4)
print a,b
#输出：[1, [1, 2, 4], 3]     [1, [1, 2, 4], 3]
```

b还是被修改了。当列表或字典参数里面的值是列表或字典时，copy并不会复制参数里面的列表或字典，这时就要用到deepcopy了。

```
a=[1,[1,2],3]
b=copy.deepcopy(a)
a[1].append(4)
print a,b
#输出：[1, [1, 2, 4], 3]   [1, [1, 2], 3]
```