Binary Tree Level Order Traversal

Given a binary tree, return the *level order* traversal of its nodes' values. (ie, from left to right, level by level).

For example:
Given binary tree `{3,9,20,#,#,15,7}`,

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

return its level order traversal as:

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

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

```java
public class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        List<List<Integer>> wrapList = new LinkedList<List<Integer>>();

        if(root == null) return wrapList;

        queue.offer(root);
        while(!queue.isEmpty()){
            int levelNum = queue.size();
            List<Integer> subList = new LinkedList<Integer>();
            for(int i=0; i<levelNum; i++) {
                if(queue.peek().left != null) queue.offer(queue.peek().left);
                if(queue.peek().right != null) queue.offer(queue.peek().right);
                subList.add(queue.poll().val);
            }
            wrapList.add(subList);
        }
        return wrapList;
    }
}
```

written by SOY original link here

Solution 2

```
vector<vector<int>> ret;

void buildVector(TreeNode *root, int depth)
{
    if(root == NULL) return;
    if(ret.size() == depth)
        ret.push_back(vector<int>());

    ret[depth].push_back(root->val);
    buildVector(root->left, depth + 1);
    buildVector(root->right, depth + 1);
}

vector<vector<int> > levelOrder(TreeNode *root) {
    buildVector(root, 0);
    return ret;
}
```

written by nilath original link here

Solution 3

```
class Solution {
public:
    vector<vector<int> > levelOrder(TreeNode *root) {
        vector<vector<int> >  result;
        if (!root) return result;
        queue<TreeNode*> q;
        q.push(root);
        q.push(NULL);
        vector<int> cur_vec;
        while(!q.empty()) {
            TreeNode* t = q.front();
            q.pop();
            if (t==NULL) {
                result.push_back(cur_vec);
                cur_vec.resize(0);
                if (q.size() > 0) {
                    q.push(NULL);
                }
            } else {
                cur_vec.push_back(t->val);
                if (t->left) q.push(t->left);
                if (t->right) q.push(t->right);
            }
        }
        return result;
    }
};
```

written by pankit original link here