Average of Levels in Binary Tree

Given a non-empty binary tree, return the average value of the nodes on each level in the form of an array.

Example 1:

```
Input:
    3
   / \
  9  20
    /  \
   15   7
Output: [3, 14.5, 11]
Explanation:
The average value of nodes on level 0 is 3,  on level 1 is 14.5, and on level 2 is 11
. Hence return [3, 14.5, 11].
```

Note:

1. The range of node's value is in the range of 32-bit signed integer.

Solution 1

Classic bfs problem. At each level, compute the average since you already know the size of the level.

```java
public List<Double> averageOfLevels(TreeNode root) {
    List<Double> result = new ArrayList<>();
    Queue<TreeNode> q = new LinkedList<>();

    if(root == null) return result;
    q.add(root);
    while(!q.isEmpty()) {
        int n = q.size();
        double sum = 0.0;
        for(int i = 0; i < n; i++) {
            TreeNode node = q.poll();
            sum += node.val;
            if(node.left != null) q.offer(node.left);
            if(node.right != null) q.offer(node.right);
        }
        result.add(sum / n);
    }
    return result;
}
```

written by jaqenhgar original link here

Solution 2

Let's visit every node of the tree once, keeping track of what depth we are on. We can do this with a simple DFS.

When we visit a node, info[depth] will be a two element list, keeping track of the sum of the nodes we have seen at this depth, and the number of nodes we have seen. This is necessary and sufficient to be able to compute the average value at this depth.

At the end of our traversal, we can simply read off the answer.

```python
def averageOfLevels(self, root):
    info = []
    def dfs(node, depth = 0):
        if node:
            if len(info) <= depth:
                info.append([0, 0])
            info[depth][0] += node.val
            info[depth][1] += 1
            dfs(node.left, depth + 1)
            dfs(node.right, depth + 1)
    dfs(root)

    return [s/float(c) for s, c in info]
```

written by awice original link here

Solution 3

sumLs: store the sum of each level.
cntLs: store each level node number.
index in the list means level in the tree.

```
public class Solution {
    public List<Double> averageOfLevels(TreeNode root) {
        List<Double> sumLs = new ArrayList<Double>();
        List<Integer> cntLs = new ArrayList<Integer>();
        if (root == null) return sumLs;
        helper(root, 0, sumLs, cntLs);
        for (int i = 0; i < sumLs.size(); i++)
            sumLs.set(i, sumLs.get(i) / cntLs.get(i));
        return sumLs;
    }

    private void helper(TreeNode root, int lv, List<Double> sumLs, List<Integer> cnt
Ls) {
        if (root != null) {
         if (sumLs.size() <= lv) {
          sumLs.add((double) root.val);
          cntLs.add(1);
         } else {
          sumLs.set(lv, sumLs.get(lv) + root.val);
          cntLs.set(lv, cntLs.get(lv) + 1);
         }
            helper(root.left, lv + 1, sumLs, cntLs);
            helper(root.right, lv + 1, sumLs, cntLs);
        }
    }
}
```

written by BryanBo.Cao original link here