Nested List Weight Sum II

Given a nested list of integers, return the sum of all integers in the list weighted by their depth.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Different from the previous question where weight is increasing from root to leaf, now the weight is defined from bottom up. i.e., the leaf level integers have weight 1, and the root level integers have the largest weight.

Example 1:
Given the list `[[1,1],2,[1,1]]`, return **8**. (four 1's at depth 1, one 2 at depth 2)

Example 2:
Given the list `[1,[4,[6]]]`, return **17**. (one 1 at depth 3, one 4 at depth 2, and one 6 at depth 1; 1*3 + 4*2 + 6*1 = 17)

Solution 1

Inspired by lzb700m's solution and one of mine. Instead of multiplying by depth, add integers multiple times (by going level by level and adding the unweighted sum to the weighted sum after each level).

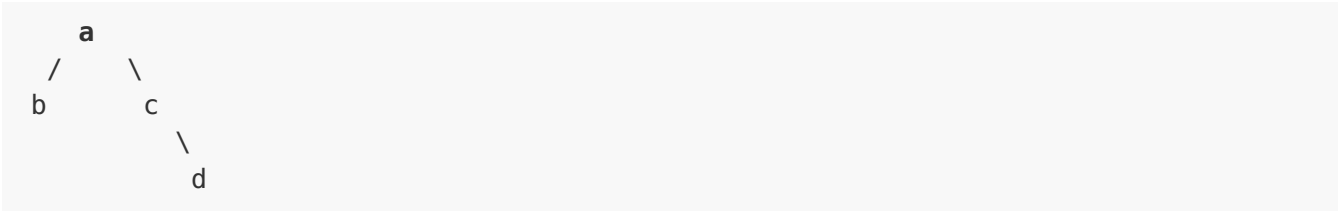
```
public int depthSumInverse(List<NestedInteger> nestedList) {
    int unweighted = 0, weighted = 0;
    while (!nestedList.isEmpty()) {
        List<NestedInteger> nextLevel = new ArrayList<>();
        for (NestedInteger ni : nestedList) {
            if (ni.isInteger())
                unweighted += ni.getInteger();
            else
                nextLevel.addAll(ni.getList());
        }
        weighted += unweighted;
        nestedList = nextLevel;
    }
    return weighted;
}
```

written by StefanPochmann original link here

Solution 2

The weight increases from the leaf to the root.

However, the following situation is not clearly defined. I will illustrate it using a tree structure.



What is the weight of a? Is it 2 to 3?

written by hyj143 original link here

Solution 3

The idea is to pass the current found integer sum into the next level of recursion, and return it back again. So that we don't have to count the number of levels in the nested list.

I think code itself is quite self explanatory.

```
public class Solution {
    public int depthSumInverse(List<NestedInteger> nestedList) {
        return helper(nestedList, 0);
    }

    private int helper(List<NestedInteger> niList, int prev) {
        int intSum = prev;
        List<NestedInteger> levelBreak = new ArrayList<>();

        for (NestedInteger ni : niList) {
            if (ni.isInteger()) {
                intSum += ni.getInteger();
            } else {
                levelBreak.addAll(ni.getList());
            }
        }

        int listSum = levelBreak.isEmpty()? 0 : helper(levelBreak, intSum);

        return listSum + intSum;
    }
}
```

written by lzb700m original link here