Nested List Weighted Sum I (LeetCode 339 - Easy)

Problem Description

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.

**Example 1:**  
Given the list [[1, 1], 2, [1, 1]], return **10**. (four 1's at depth 2, one 2 at depth 1)

**Example 2:**  
Given the list [1, [4, [6]]], return **27**. (one 1 at depth 1, one 4 at depth 2, and one 6 at depth 3; 1 + 4\*2 + 6\*3 = 27)



Problem Analysis

DFS, pass an argument **depth** to each level and use multiplication.

Solution



Nested List Weighted Sum II (LeetCode 364 - Medium)

Problem Description

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)

Problem Analysis

Solution 1

We can always use recursion to get the **max depth** of the Nested List. Use the solution of the previous question with a little modification on the argument of **depth**.

With this solution, we have to DFS the Nested List twice, first time to get the depth of the whole list, second time for calculating the sum.

Solution 2

Only DFS once.

Maintain

Two variables: **weighted** and **unweighted** and one list: **allSubLists**.

When iterating through each level, add all the integers to **unweighted**, and add all the sub lists into **allSubLists**. After iterating through the current level, add **unweighted** to **weighted.** By doing this we do not need to use any multiplications.

The variable **allSubLists** is now the next level so we assign **allSubLists** to the original list then keep iterating until the list is empty.

Solution 1



Solution 2

