LC 364. Nested List Weight Sum II

Question

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:

```
Input: [[1,1],2,[1,1]]
Output: 8
Explanation: Four 1's at depth 1, one 2 at depth 2.
```

Example 2:

```
Input: [1,[4,[6]]]
Output: 17
Explanation: One 1 at depth 3, one 4 at depth 2, and one 6 at depth 1; 1*3 + 4*2 + 6*1 = 17.
```

Solution

```
# """
# This is the interface that allows for creating nested lists.
# You should not implement it, or speculate about its implementation
# """
#class NestedInteger:
#    def __init__(self, value=None):
#        """
#        If value is not specified, initializes an empty list.
#        Otherwise initializes a single integer equal to value.
#        """
#
#    def isInteger(self):
#        """
#        @return True if this NestedInteger holds a single integer, rather than a
nested list.
#        :rtype bool
#        """
#
#    def add(self, elem):
#        """
#        Set this NestedInteger to hold a nested list and adds a nested integer
elem to it.
#        :rtype void
#        """
#
#    def setInteger(self, value):
#        """
#        Set this NestedInteger to hold a single integer equal to value.
#        :rtype void
#        """
#
#    def getInteger(self):
#        """
#        @return the single integer that this NestedInteger holds, if it holds a
single integer
#        Return None if this NestedInteger holds a nested list
#        :rtype int
#        """
#
#    def getList(self):
#        """
#        @return the nested list that this NestedInteger holds, if it holds a
nested list
#        Return None if this NestedInteger holds a single integer
#        :rtype List[NestedInteger]
#        """

class Solution:
    def depthSumInverse(self, nestedList):
        """
        :type nestedList: List[NestedInteger]
        :rtype: int
        """
        #Solution 2
        q = [item for item in nestedList]
        stack = []
        s = 0
        while len(q) > 0:
            n = len(q)
            for i in range(n):
                item = q.pop(0)
                if item.isInteger():
                    s += item.getInteger()
                else:
                    q += item.getList()
            stack.append(s)
        return sum(stack)


        #Solution
        #Add all the integers at level to level_sum.
        #Push all elements which are not interger
        #(and are list type) into the list for next iteration.
        #Make sure to flatten this list otherwise infinite loop.
        #Now we only initilaize level_sum once.
        #And successive level's integers are added to it.
        #Once a level finishes, we add to total_sum.
        #This naturally implements the multiplication logic - lower level
        #sums are added multiple times to total sum.
        #这个方法牛逼之处在于total_sum += level_sum, +=
        #eg:[1,[4,[6]]]最外圈的1被到total_sum 3次, 4加到2次, 6一次
        #刚好对应了6的权重1, 4的权重2, 1的权重3

        #[[1,1],2,[1,1]]
        #第一编nestedList, total_sum=2, 第二遍的时候直接在第一遍的total_sum上加的
        #相当于2, 最后被计算了两次
        total_sum, level_sum = 0, 0
        while len(nestedList):
            next_level_list = []
            for x in nestedList:
                if x.isInteger():
                    level_sum += x.getInteger()
                else:
                    for y in x.getList():
                        next_level_list.append(y)
            total_sum += level_sum
            nestedList = next_level_list
        return total_sum
```