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)

```java
/**
 * This is the interface that allows for creating nested lists.
 * You should not implement it, or speculate about its implementation
 */
public interface NestedInteger {
      // @return true if this NestedInteger holds a single integer,
      // rather than a nested list.
      public boolean isInteger();

      // @return the single integer that this NestedInteger holds,
      // if it holds a single integer
      // Return null if this NestedInteger holds a nested list
      public Integer getInteger();

      // @return the nested list that this NestedInteger holds,
      // if it holds a nested list
      // Return null if this NestedInteger holds a single integer
      public List<NestedInteger> getList();
}
```

Problem Analysis

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

Solution

```
/**
 * LeetCode 339
 * @param nestedList
 * @return the sum of all integers in the list weighted by their depth
 */
public int depthSum(List<NestedInteger> nestedList) {
  return depthSum(nestedList, 1);
}

private int depthSum(List<NestedInteger> nestedList, int depth) {
  int sum = 0;
  for (NestedInteger subList : nestedList) {
      if (subList == null) {
          continue;
      }
      if (subList.isInteger()) {
          sum += depth * subList.getInteger();
      } else {
          sum += depthSum(subList.getList(), depth + 1);
      }
  }
  return sum;
}
```

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

```
  /**
   * LeetCode 364
   * @param nestedList
   * @return  the sum of all integers in the list weighted by their depth
   *          now the weight is defined from bottom up
   */
public int depthSumInverse1(List<NestedInteger> nestedList) {
    int depth = getDepth(nestedList);
    return depthSumInverse1(nestedList, depth);
}

private int getDepth(List<NestedInteger> nestedList) {
    int depth = 0;
    for (NestedInteger subList : nestedList) {
        if (subList == null) {
            continue;
        }
        if (subList.isInteger()) {
            depth = depth > 1 ? depth : 1;
        } else {
            int newDepth = getDepth(subList.getList()) + 1;
            depth = depth > newDepth ? depth : newDepth;
        }
    }
    return depth;
}

private int depthSumInverse1(List<NestedInteger> nestedList, int depth) {
    int sum = 0;
    for (NestedInteger subList : nestedList) {
        if (subList == null) {
            continue;
        }
        if (subList.isInteger()) {
            sum += subList.getInteger() * depth;
        } else {
            sum += depthSumInverse1(subList.getList(), depth - 1);
        }
    }
    return sum;
}
```

Solution 2

```
  /**
   * LeetCode 364
   * @param nestedList
   * @return  the sum of all integers in the list weighted by their depth
   *          now the weight is defined from bottom up
   */
public int depthSumInverse2(List<NestedInteger> nestedList) {
    int weighted = 0;
    int unweighted = 0;
    while (!nestedList.isEmpty()) {
        List<NestedInteger> allSubLists = new ArrayList<NestedInteger>();
        for (NestedInteger subList : nestedList) {
            if (subList == null) {
                continue;
            }
            if (subList.isInteger()) {
                unweighted += subList.getInteger();
            } else {
                allSubLists.addAll(subList.getList());
            }
        }
        weighted += unweighted;
        nestedList = allSubLists;
    }
    return weighted;
}
```