341. Flatten Nested List Iterator

Medium 2028 775 Add to List Share

Given a nested list of integers, implement an iterator to flatten it.

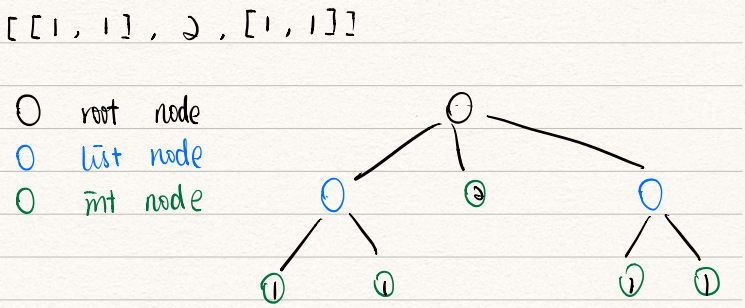
Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Example 1:

```
Input: [[1,1],2,[1,1]]
Output: [1,1,2,1,1]
Explanation: By calling next repeatedly until hasNext returns false,
             the order of elements returned by next should be: [1,1,2,1,1].
```

Example 2:

```
Input: [1,[4,[6]]]
Output: [1,4,6]
Explanation: By calling next repeatedly until hasNext returns false,
             the order of elements returned by next should be: [1,4,6].
```



```cpp
/**
 * // This is the interface that allows for creating nested lists.
 * // You should not implement it, or speculate about its implementation
 * class NestedInteger {
 *   public:
 *     // Return true if this NestedInteger holds a single integer, rather than a nested list.
 *     bool isInteger() const;
 *
 *     // Return the single integer that this NestedInteger holds, if it holds a single integer
 *     // The result is undefined if this NestedInteger holds a nested list
 *     int getInteger() const;
 *
 *     // Return the nested list that this NestedInteger holds, if it holds a nested list
 *     // The result is undefined if this NestedInteger holds a single integer
 *     const vector<NestedInteger> &getList() const;
 * };
 */

class NestedIterator {
public:
    NestedIterator(vector<NestedInteger> &nestedList) {
        index = 0;
        for (auto node : nestedList) {
            traverse(node);
        }
    }

    int next() {
        return res[index++];
    }

    bool hasNext() {
        if (index < res.size()) {
            return true;
        }
        return false;
    }

private:
    vector<int> res;
    int index;

    void traverse(NestedInteger root) {
        if (root.isInteger()) {
            // achieve leaf node
            res.push_back(root.getInteger());
            return;
        }

        // iterate subtree
        for (auto subtree : root.getList()) {
            traverse(subtree);
        }
    }
};

/**
 * Your NestedIterator object will be instantiated and called as such:
 * NestedIterator i(nestedList);
 * while (i.hasNext()) cout << i.next();
 */
```