Flatten Nested List Iterator

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:
Given the list `[[1,1],2,[1,1]]`,

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

Example 2:
Given the list `[1,[4,[6]]]`,

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

Solution 1

An iterator shouldn't copy the entire data but just iterate over the original data structure.

I keep the current progress in a stack. My `hasNext` tries to find an integer. My `next` returns it and moves on. I call `hasNext` in `next` because `hasNext` is optional. Some user of the iterator might call only `next` and never `hasNext`, e.g., if they know how many integers are in the structure or if they want to handle the ending with exception handling.

Python

Using a stack of [list, index] pairs.

```python
class NestedIterator(object):

    def __init__(self, nestedList):
        self.stack = [[nestedList, 0]]

    def next(self):
        self.hasNext()
        nestedList, i = self.stack[-1]
        self.stack[-1][1] += 1
        return nestedList[i].getInteger()

    def hasNext(self):
        s = self.stack
        while s:
            nestedList, i = s[-1]
            if i == len(nestedList):
                s.pop()
            else:
                x = nestedList[i]
                if x.isInteger():
                    return True
                s[-1][1] += 1
                s.append([x.getList(), 0])
        return False
```

Java

Using a stack of ListIterators.

```
public class NestedIterator implements Iterator<Integer> {

    public NestedIterator(List<NestedInteger> nestedList) {
        lists = new Stack<>();
        lists.push(nestedList.listIterator());
    }

    public Integer next() {
        hasNext();
        return lists.peek().next().getInteger();
    }

    public boolean hasNext() {
        while (!lists.empty()) {
            if (!lists.peek().hasNext()) {
                lists.pop();
            } else {
                NestedInteger x = lists.peek().next();
                if (x.isInteger())
                    return lists.peek().previous() == x;
                lists.push(x.getList().listIterator());
            }
        }
        return false;
    }

    private Stack<ListIterator<NestedInteger>> lists;
}
```

C++

Using stacks of begin and end iterators.

```
class NestedIterator {
public:
    NestedIterator(vector<NestedInteger> &nestedList) {
        begins.push(nestedList.begin());
        ends.push(nestedList.end());
    }

    int next() {
        hasNext();
        return (begins.top()++)->getInteger();
    }

    bool hasNext() {
        while (begins.size()) {
            if (begins.top() == ends.top()) {
                begins.pop();
                ends.pop();
            } else {
                auto x = begins.top();
                if (x->isInteger())
                    return true;
                begins.top()++;
                begins.push(x->getList().begin());
                ends.push(x->getList().end());
            }
        }
        return false;
    }

private:
    stack<vector<NestedInteger>::iterator> begins, ends;
};
```

written by StefanPochmann original link here

Solution 2

A question before this is the Nested List Weight Sum, and it requires recursion to solve. As it carries to this problem that we will need recursion to solve it. But since we need to access each NestedInteger at a time, we will use a stack to help.

In the constructor, we push all the nestedList into the stack from back to front, so when we pop the stack, it returns the very first element. Second, in the hasNext() function, we peek the first element in stack currently, and if it is an Integer, we will return true and pop the element. If it is a list, we will further flatten it. This is iterative version of flatting the nested list. Again, we need to iterate from the back to front of the list.

```
public class NestedIterator implements Iterator<Integer> {
    Stack<NestedInteger> stack = new Stack<>();
    public NestedIterator(List<NestedInteger> nestedList) {
        for(int i = nestedList.size() - 1; i >= 0; i--) {
            stack.push(nestedList.get(i));
        }
    }

    @Override
    public Integer next() {
        return stack.pop().getInteger();
    }

    @Override
    public boolean hasNext() {
        while(!stack.isEmpty()) {
            NestedInteger curr = stack.peek();
            if(curr.isInteger()) {
                return true;
            }
            stack.pop();
            for(int i = curr.getList().size() - 1; i >= 0; i--) {
                stack.push(curr.getList().get(i));
            }
        }
        return false;
    }
}
```

written by yfcheng original link here

Solution 3

For this problem "Run Code" button is not working and code can't be tested. Please fix it :)

written by kaidul original link here