<https://leetcode.com/problems/flatten-nested-list-iterator/description/>

You are given a nested list of integers nestedList. Each element is either an integer or a list whose elements may also be integers or other lists. Implement an iterator to flatten it.

Implement the NestedIterator class:

NestedIterator(List<NestedInteger> nestedList) Initializes the iterator with the nested list nestedList.

int next() Returns the next integer in the nested list.

boolean hasNext() Returns true if there are still some integers in the nested list and false otherwise.

Your code will be tested with the following pseudocode:

initialize iterator with nestedList

res = []

while iterator.hasNext()

append iterator.next() to the end of res

return res

If res matches the expected flattened list, then your code will be judged as correct.

**Example 1:**

**Input:** nestedList = [[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:** nestedList = [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].

**Constraints:**

1 <= nestedList.length <= 500

The values of the integers in the nested list is in the range [-106, 106].

**Attempt 1: 2025-02-18**

**Solution 1: Stack (180 min)**

/\*\*

 \* // 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 empty list if this NestedInteger holds a single integer

 \*     public List<NestedInteger> getList();

 \* }

 \*/

public class NestedIterator implements Iterator<Integer> {

    Stack<NestedInteger> stack = new Stack<>();

    public NestedIterator(List<NestedInteger> nestedList) {

        prepareStack(nestedList);

    }

    // Returns the next integer in the nested list.

    @Override

    public Integer next() {

        if(!hasNext()) {

            return null;

        }

        return stack.pop().getInteger();

    }

    // Returns true if there are still some integers in the nested list and false otherwise.

    @Override

    public boolean hasNext() {

        // 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.

        while(!stack.isEmpty() && !stack.peek().isInteger()) {

            List<NestedInteger> list = stack.pop().getList();

            prepareStack(list);

        }

        return !stack.isEmpty();

    }

    private void prepareStack(List<NestedInteger> nestedList) {

        for(int i = nestedList.size() - 1; i >= 0; i--) {

            stack.push(nestedList.get(i));

        }

    }

}

/\*\*

 \* Your NestedIterator object will be instantiated and called as such:

 \* NestedIterator i = new NestedIterator(nestedList);

 \* while (i.hasNext()) v[f()] = i.next();

 \*/

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to Deepseek**

Your solution for the LeetCode problem 341, "Flatten Nested List Iterator," is well-structured and follows the correct approach using a stack to handle the nested structure.

<https://leetcode.com/problems/flatten-nested-list-iterator/solutions/80147/simple-java-solution-using-a-stack-with-explanation/>

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.

Note: first version of the solution has some mistakes as mentioned in some of the comments below, updated it and apologize if it was misleading.

Below is a detailed explanation of the solution along with the code:

**Problem Explanation**

The problem requires you to implement an iterator that flattens a nested list of integers. The nested list is represented by the NestedInteger interface, which can either be a single integer or a list of NestedInteger objects.

**Approach**

Stack Usage: Use a stack to keep track of the nested lists. By pushing elements onto the stack in reverse order, you ensure that the first element of the nested list is on top of the stack.

Constructor: Initialize the stack by pushing all elements from the nested list onto the stack in reverse order.

hasNext() Method: This method checks if there are any integers left to iterate over. If the top element of the stack is a list, it flattens it by pushing its elements onto the stack.

next() Method: This method returns the next integer in the flattened list. It assumes that hasNext() has been called to ensure the top element is an integer.

**Java Solution Code**

import java.util.\*;

public class NestedIterator implements Iterator<Integer> {

Deque<NestedInteger> stack = new ArrayDeque<>();

public NestedIterator(List<NestedInteger> nestedList) {

prepareStack(nestedList);

}

@Override

public Integer next() {

if (!hasNext()) {

return null;

}

return stack.pop().getInteger();

}

@Override

public boolean hasNext() {

while (!stack.isEmpty() && !stack.peek().isInteger()) {

List<NestedInteger> list = stack.pop().getList();

prepareStack(list);

}

return !stack.isEmpty();

}

private void prepareStack(List<NestedInteger> nestedList) {

for (int i = nestedList.size() - 1; i >= 0; i--) {

stack.push(nestedList.get(i));

}

}

}

// This is the interface that allows for creating nested lists.

// You should not implement it, or speculate about its implementation

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();

}

**Explanation of the Code**

Constructor: The constructor initializes the stack by pushing all elements from the nested list onto the stack in reverse order using the prepareStack method.

prepareStack Method: This method takes a list of NestedInteger and pushes each element onto the stack in reverse order.

hasNext Method: This method checks if the stack is not empty and if the top element is not an integer. If the top element is a list, it flattens it by pushing its elements onto the stack. This continues until the top element is an integer or the stack is empty.

next Method: This method returns the next integer in the flattened list. It assumes that hasNext() has been called to ensure the top element is an integer.

**Complexity Analysis**

Time Complexity:

Constructor: O(N), where N is the total number of nested elements.

next(): O(1).

hasNext(): O(1) on average, but can be O(N) in the worst case when flattening a deeply nested list.

Space Complexity: O(N), where N is the total number of nested elements.

This solution efficiently flattens the nested list and provides an iterator to access each integer in sequence.

**If give you input as nestedList = [[1,2],2,[1,1]], can you simulate how it process ?**

Sure! Let's simulate how the NestedIterator processes the input nestedList = [[1,2],2,[1,1]]. We'll go step by step through the constructor, hasNext(), and next() calls.

**Input**

nestedList = [[1,2], 2, [1,1]]

**Step 1: Constructor**

When the NestedIterator is initialized, the constructor calls prepareStack(nestedList).

**prepareStack(nestedList):**

The input nestedList is [[1,2], 2, [1,1]].

The stack is initially empty.

We iterate through nestedList in reverse order and push each element onto the stack.

|  |  |  |
| --- | --- | --- |
| Step | Element | Stack After Push (top -> bottom) |
| 1 | [1,1] | [[1,1]] |
| 2 | 2 | [2, [1,1]] |
| 3 | [1,2] | [[1,2], 2, [1,1]] |

Final Stack After Constructor:

Stack: [[1,2], 2, [1,1]]

**Step 2: First hasNext() Call**

The hasNext() method checks if there is an integer available to return.

**hasNext() Logic:**

The stack is not empty, and the top element is [1,2] (a list, not an integer).

Pop [1,2] from the stack and call prepareStack([1,2]).

**prepareStack([1,2]):**

Iterate through [1,2] in reverse order and push elements onto the stack.

|  |  |  |
| --- | --- | --- |
| Step | Element | Stack After Push (top -> bottom) |
| 1 | 2 | [2, 2, [1,1]] |
| 2 | 1 | [1, 2, 2, [1,1]] |

Stack After Flattening [1,2]:

Stack: [1, 2, 2, [1,1]]

The top element is now 1 (an integer), so hasNext() returns true.

**Step 3: First next() Call**

The next() method returns the top element of the stack, which is 1.

Pop 1 from the stack.

Return 1.

Stack After next():

Stack: [2, 2, [1,1]]

**Step 4: Second hasNext() Call**

The hasNext() method checks if there is another integer available.

The stack is not empty, and the top element is 2 (an integer).

No need to flatten anything. hasNext() returns true.

**Step 5: Second next() Call**

The next() method returns the top element of the stack, which is 2.

Pop 2 from the stack.

Return 2.

Stack After next():

Stack: [2, [1,1]]

**Step 6: Third hasNext() Call**

The hasNext() method checks if there is another integer available.

The stack is not empty, and the top element is 2 (an integer).

No need to flatten anything. hasNext() returns true.

**Step 7: Third next() Call**

The next() method returns the top element of the stack, which is 2.

Pop 2 from the stack.

Return 2.

Stack After next():

Stack: [[1,1]]

**Step 8: Fourth hasNext() Call**

The hasNext() method checks if there is another integer available.

The stack is not empty, and the top element is [1,1] (a list, not an integer).

Pop [1,1] from the stack and call prepareStack([1,1]).

**prepareStack([1,1]):**

Iterate through [1,1] in reverse order and push elements onto the stack.

|  |  |  |
| --- | --- | --- |
| Step | Element | Stack After Push (top -> bottom) |
| 1 | 1 | [1] |
| 2 | 1 | [1, 1] |

Stack After Flattening [1,1]:

Stack: [1, 1]

The top element is now 1 (an integer), so hasNext() returns true.

**Step 9: Fourth next() Call**

The next() method returns the top element of the stack, which is 1.

Pop 1 from the stack.

Return 1.

Stack After next():

Stack: [1]

**Step 10: Fifth hasNext() Call**

The hasNext() method checks if there is another integer available.

The stack is not empty, and the top element is 1 (an integer).

No need to flatten anything. hasNext() returns true.

**Step 11: Fifth next() Call**

The next() method returns the top element of the stack, which is 1.

Pop 1 from the stack.

Return 1.

Stack After next():

Stack: []

**Step 12: Sixth hasNext() Call**

The hasNext() method checks if there is another integer available.

The stack is empty.

hasNext() returns false.

**Final Output**

The iterator returns the integers in the following order:

1, 2, 2, 1, 1

This matches the flattened version of the input nestedList = [[1,2],2,[1,1]].

**Refer to**

[L565.Array Nesting (Ref.L341)](note://674037E23BE340FBBD975317FF31B3A5)