Mini Parser

Given a nested list of integers represented as a string, implement a parser to deserialize it.

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

Note: You may assume that the string is well-formed:

- String is non-empty.
- String does not contain white spaces.
- String contains only digits `0-9`, `[`, `-` `,`, `]`.

Example 1:

```
Given s = "324",

You should return a NestedInteger object which contains a single integer 324.
```

Example 2:

```
Given s = "[123,[456,[789]]]",

Return a NestedInteger object containing a nested list with 2 elements:

1. An integer containing value 123.
2. A nested list containing two elements:
    i.  An integer containing value 456.
    ii. A nested list with one element:
         a. An integer containing value 789.
```

Solution 1

This approach will just iterate through every char in the string (no recursion).

- If encounters '[', push current NestedInteger to stack and start a new one.
- If encounters ']', end current NestedInteger and pop a NestedInteger from stack to continue.
- If encounters ',', append a new number to curr NestedInteger, if this comma is not right after a brackets.
- Update index l and r, where l shall point to the start of a integer substring, while r shall points to the end+1 of substring.

Java Code:

```
public NestedInteger deserialize(String s) {
    if (s.isEmpty())
        return null;
    if (s.charAt(0) != '[') // ERROR: special case
        return new NestedInteger(Integer.valueOf(s));

    Stack<NestedInteger> stack = new Stack<>();
    NestedInteger curr = null;
    int l = 0; // l shall point to the start of a number substring;
               // r shall point to the end+1 of a number substring
    for (int r = 0; r < s.length(); r++) {
        char ch = s.charAt(r);
        if (ch == '[') {
            if (curr != null) {
                stack.push(curr);
            }
            curr = new NestedInteger();
            l = r+1;
        } else if (ch == ']') {
            String num = s.substring(l, r);
            if (!num.isEmpty())
                curr.add(new NestedInteger(Integer.valueOf(num)));
            if (!stack.isEmpty()) {
                NestedInteger pop = stack.pop();
                pop.add(curr);
                curr = pop;
            }
            l = r+1;
        } else if (ch == ',') {
            if (s.charAt(r-1) != ']') {
                String num = s.substring(l, r);
                curr.add(new NestedInteger(Integer.valueOf(num)));
            }
            l = r+1;
        }
    }

    return curr;
}
```

written by AlexTheGreat original link here

Solution 2

Python using `eval`:

```
def deserialize(self, s):
    def nestedInteger(x):
        if isinstance(x, int):
            return NestedInteger(x)
        lst = NestedInteger()
        for y in x:
            lst.add(nestedInteger(y))
        return lst
    return nestedInteger(eval(s))
```

Python one-liner

```
def deserialize(self, s):
    return NestedInteger(s) if isinstance(s, int) else reduce(lambda a, x: a.add(
self.deserialize(x)) or a, s, NestedInteger()) if isinstance(s, list) else self.d
eserialize(eval(s))
```

Python Golf (136 bytes or 31 bytes)

```
class Solution:deserialize=d=lambda S,s,N=NestedInteger:s<[]and N(s)or s<''and red
uce(lambda a,x:a.add(S.d(x))or a,s,N())or S.d(eval(s))
```

Or abusing how the judge judges (yes, this gets accepted):

```
class Solution:deserialize=eval
```

Python parsing char by char

Here I turned the input string into a list with sentinel for convenience.

```
def deserialize(self, s):
    def nestedInteger():
        num = ''
        while s[-1] in '1234567890-':
            num += s.pop()
        if num:
            return NestedInteger(int(num))
        s.pop()
        lst = NestedInteger()
        while s[-1] != ']':
            lst.add(nestedInteger())
            if s[-1] == ',':
                s.pop()
        s.pop()
        return lst
    s = list(' ' + s[::-1])
    return nestedInteger()
```

C++ using `istringstream`

```
class Solution {
public:
    NestedInteger deserialize(string s) {
        istringstream in(s);
        return deserialize(in);
    }
private:
    NestedInteger deserialize(istringstream &in) {
        int number;
        if (in >> number)
            return NestedInteger(number);
        in.clear();
        in.get();
        NestedInteger list;
        while (in.peek() != ']') {
            list.add(deserialize(in));
            if (in.peek() == ',')
                in.get();
        }
        in.get();
        return list;
    }
};
```

written by StefanPochmann original link here

Solution 3

Solution in a Glance:
This solution uses a stack to record the NestedInteger's.
At the very beginning, an empty NestedInteger is placed in the stack. This NestedInteger will be regarded as a list that holds one but only one NestedInteger, which will be returned in the end.
Logic: When encountering '[', the stack has one more element. When encountering ']', the stack has one less element.

Complexities:

- Time: *O*(n)
- Space: worse-case *O*(n) (worse case: [1,[2,[3,[....[n-1,[n]]]....])

C++ Accepted Code:
(A possible implementation of **NestedInteger** is also provided in the first reply.)

```
class Solution {
public:
    NestedInteger deserialize(string s) {
        function<bool(char)> isnumber = [](char c){ return (c == '-') || isdigit(
c); };

        stack<NestedInteger> stk;
        stk.push(NestedInteger());

        for (auto it = s.begin(); it != s.end();) {
            const char & c = (*it);
            if (isnumber(c)) {
                auto it2 = find_if_not(it, s.end(), isnumber);
                int val = stoi(string(it, it2));
                stk.top().add(NestedInteger(val));
                it = it2;
            }
            else {
                if (c == '[') {
                    stk.push(NestedInteger());
                }
                else if (c == ']') {
                    NestedInteger ni = stk.top();
                    stk.pop();
                    stk.top().add(ni);
                }
                ++it;
            }
        }

        NestedInteger result = stk.top().getList().front();
        return result;
    }
};
```

written by zhiqing_xiao original link here

From Leetcoder.