LeetCode 394

https://leetcode.com/problems/decode-string/description/

Yifeng Zeng

Description

394. Decode String

Given an encoded string, return it's decoded string.

The encoding rule is: k[encoded_string], where the encoded_string inside the square brackets is being repeated exactly k times. Note that k is guaranteed to be a positive integer.

You may assume that the input string is always valid; No extra white spaces, square brackets are well-formed, etc.

Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, k. For example, there won't be input like 3a or 2[4].

Examples:

s = "3[a]2[bc]", return "aaabcbc".

s = "3[a2[c]]", return "accaccacc".

s = "2[abc]3[cd]ef", return "abcabccdcdcdef".

Idea Report

This is a very straight forward step by step problem. We can check each character (ch), and treat differernt kinds of characters differerntly. We may want two stacks to store our information, one "stack" to store our parentheses and the the letters, another "nums" to store out numbers.

- If ch is a digit, we have a while loop to record this number because it may not be just single digit.
- If ch is a letter, we just store it for future use.
- If ch is '[', we store it for future use. When we meet a ']', this '[' is our signal to stop.
- If ch is ']', we pop out all the letters until we meet a "[". And then we duplicate n times of these poped out letters, n is from the "nums" stack where we store the how many time a string should be duplicated.

In the end, all the letters has correct duplcations in the "stack" without any parentheses, so we just pop everything out reverse it and return.

Code

```
class Solution {
    public String decodeString(String s) {
        Deque<Character> stack = new ArrayDeque<>();
        Deque<Integer> nums = new ArrayDeque<>();
        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
            if (Character.isDigit(ch)) {
                int n = 0;
                while (Character.isDigit(s.charAt(i))) {
                    n = n * 10 + s.charAt(i) - '0';
                    i++;
                }
                nums.push(n);
                i--;
            } else if (ch == '[') {
                stack.push(ch);
            } else if (ch ==']') {
                pop(stack, nums.isEmpty() ? 1 : nums.pop());
            } else if (Character.isLetter(ch)) {
                stack.push(ch);
            }
        }

        StringBuilder sb = new StringBuilder();
        while (!stack.isEmpty()) {
            sb.append(stack.pop());
        }
        return sb.reverse().toString();
    }

    private void pop(Deque<Character> stack, int n) {
        List<Character> helper = new ArrayList<>();
        while (!stack.isEmpty() && stack.peek() != '[') {
            helper.add(stack.pop());
```

```
            }
            stack.pop();
            for (int i = 0; i < n; i++) {
                for (int j = helper.size() - 1; j >= 0; j--) {
                    stack.push(helper.get(j));
                }
            }
        }
    }
```

Summary

- This is the basic calculator kind of problem, we can reduce the problem into a series of sub problems. When a character is number, character, "+""-", "*""/", "[" or "]".

Basic Calculator Template

```
class Solution {
    public int calculate(String s) {
        Deque<Character> stack = new ArrayDeque<>();
        Deque<Integer> nums = new ArrayDeque<>();
        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
            if (Character.isDigit(ch)) {
                int n = 0;
                while (i < s.length() && Character.isDigit(s.charAt(i))) {
                    n = n * 10 + s.charAt(i++) - '0';
                }
                nums.push(n);
                i--;
            } else if (ch == '+' || ch == '-') {
                while (!stack.isEmpty() && stack.peek() != '(') {
                    pop(stack, nums);
                }
                stack.push(ch);
            } else if (ch == '*' || ch == '/') {
                while (!stack.isEmpty() && stack.peek() != '('
                       && stack.peek() != '+' && stack.peek() != '-') {
                    pop(stack, nums);
                }
                stack.push(ch);
            } else if (ch == ')') {
```

```
                while (stack.peek() != '(') {
                    pop(stack, nums);
                }
                stack.pop();
            } else if  (ch == '(') {
                stack.push(ch);
            }
        }

        while (!stack.isEmpty()) {
            pop(stack, nums);
        }
        return nums.pop();
    }

    private void pop(Deque<Character> stack, Deque<Integer> nums) {
        int n1 = nums.pop();
        int n2 = nums.pop();
        int op = stack.pop();
        if (op == '+') {
            nums.push(n2 + n1);
        } else if (op == '-') {
            nums.push(n2 - n1);
        } else if (op == '*') {
            nums.push(n2 * n1);
        } else if (op == '/') {
            nums.push(n2 / n1);
        }
    }
}
```