Java O(n) solution using stack with detail explanation

32

yfcheng
Reputation: 932

First, given `"bcabc"`, the solution should be `"abc"`. If we think about this problem intuitively, you would sort of go from the beginning of the string and start removing one if there is still the same character left and a smaller character is after it. Given `"bcabc"`, when you see a `'b'`, keep it and continue with the search, then keep the following `'c'`, then we see an `'a'`. Now we get a chance to get a smaller lexi order, you can check if after `'a'`, there is still `'b'` and `'c'` or not. We indeed have them and `"abc"` will be our result.

Come to the implementation, we need some data structure to store the previous characters `'b'` and `'c'`, and we need to compare the current character with previous saved ones, and if there are multiple same characters, we prefer left ones. This calls for a stack.

After we decided to use stack, the implementation becomes clearer. From the intuition, we know that we need to know if there are still remaining characters left or not. So we need to iterate the array and save how many each characters are there. A visited array is also required since we want unique character in the solution. The line `while(!stack.isEmpty() && stack.peek() > c && count[stack.peek()-'a'] > 0)` checks that the queued character should be removed or not, like the `'b'` and `'c'` in the previous example. After removing the previous characters, push in the new char and mark the visited array.

Time complexity: O(n), n is the number of chars in string.

Space complexity: O(n) worst case.

```
public String removeDuplicateLetters(String s) {
    Stack<Character> stack = new Stack<>();
    int[] count = new int[26];
    char[] arr = s.toCharArray();
    for(char c : arr) {
        count[c-'a']++;
    }
    boolean[] visited = new boolean[26];
    for(char c : arr) {
        count[c-'a']--;
        if(visited[c-'a']) {
            continue;
        }
        while(!stack.isEmpty() && stack.peek() > c && count[stack.peek()-'a'] > 0) {
            visited[stack.peek()-'a'] = false;
            stack.pop();
        }
        stack.push(c);
        visited[c-'a'] = true;
    }
    StringBuilder sb = new StringBuilder();
    for(char c : stack) {
        sb.append(c);
    }
    return sb.toString();
}
```

1

eric_haibin_lin
Reputation: ★ 5

Interesting that the `foreach` loop for Java's `Stack` iterates from the bottom of the stack.

0

lisepher
Reputation: ★ 1

Thank you very much for the explanation, clean and easy to understand!

2

cdai
Reputation: ★ 128

Very clear explanation! At last I read yours and got it. Thanks a lot. Here is the shorter one. And I waste some space to make it more concise. (It's really annoying to convert char to int again and again...)

```
    // O(MN) time: M is size of alphabet, like O(26*N) for this problem.
    // pre: stack holding previous char
    // post: how many char left after cur position
    // dup: if char is already on stack (in result)
    public String removeDuplicateLetters(String s) {
        int[] post = new int[256];
        for (char c : s.toCharArray()) post[c]++;

        boolean[] dup = new boolean[256];
        Stack<Character> pre = new Stack<>();
        for (char c : s.toCharArray()) {
            post[c]--;
            if (dup[c]) continue;
```

```
            while (!pre.isEmpty() && c < pre.peek() && post[pre.peek()] > 0)
                dup[pre.pop()] = false;
            pre.push(c);
            dup[c] = true;
        }

        StringBuilder ret = new StringBuilder();
        for (char c : pre) ret.append(c);
        return ret.toString();
    }
```

2

oldfish
Reputation: ★ 15

I like this explanation, but, dame, I just don't have such intuition...

0

grv20.mishra @oldfish
Reputation: ★ 0

@oldfish Same here buddy

JAVA 7710 SOLUTION-SHARING 15815 STACK 240

6 POSTS

2455 VIEWS

Log in to reply