ACM金牌选手讲解LeetCode算法《单调栈和滑动窗口》

上一篇文章讲解了《线性表》中的数组、链表、栈和队列的概念和基本应用，本文讲解栈和队列的高级应用。

- 单调栈
- 双端队列
- 滑动窗口

单调栈

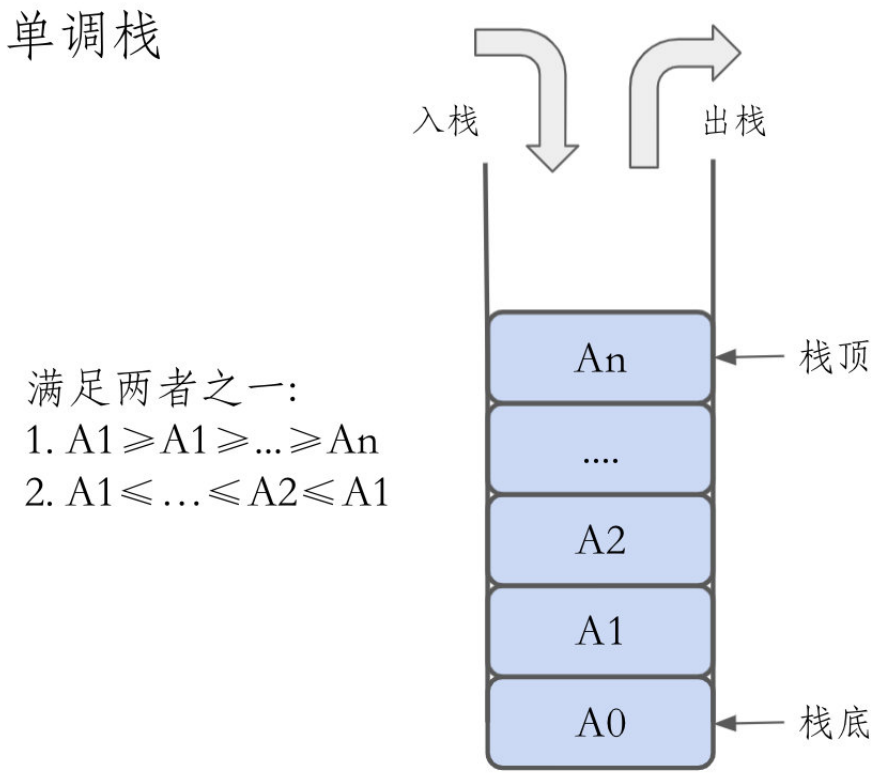
介绍

单调栈 = 单调 + 栈，因此其同时满足两个特性: 单调性、栈的特点。

- 单调性: 单调栈里面所存放的数据是有序的(单调递增或递减)。
- 栈: 后进先出。

因其满足单调性和每个数字只会入栈一次，所以可以在时间复杂度 `O(n)` 的情况下解决一些问题。

下图是单调栈的图解，栈内数字满足单调性，且满足栈的后进先出的特性。



例题

LeetCode 739. 每日温度

题意

给定每天的温度，求对于每一天需要等几天才可以等到更暖和的一天。如果该天之后不存在 更暖和的天气，则记为 0。

输出一个一维数组，表示每天需要等待的天数。

示例

```
输入: temperatures = [73,74,75,71,69,72,76,73]
输出: [1,1,4,2,1,1,0,0]
```

题解

建立单调(非增)栈，栈存放每天的温度，为了方便计算天数，栈中存储的每天的温度在数组中下标，可以通过下标得到对应天的温度。

设温度数组为 `a`，从左向右依次遍历数组 `a`，假设当前遍历到数组位置为 `j`，则对应的天温度为 `a[j]`，设栈顶元素的位置为 `i`，则对应的天的温度`a[i]`，分为两种情况讨论。

- 如果 `a[j] > a[i]`，执行以下三步。
  - 表明比第 `i` 天更暖和的一天为第 `j` 天，则第 `i` 天的答案为 `j-i`，那么可以将栈顶元素弹出。
  - 重复检查栈顶元素，直至栈顶元素的 `a[j] <= a[i]` 或者 栈为空。
  - 将 `j` 入栈。
- 如果 `a[j] <= a[i]`，
  - 表明第 `i` 天没有找到更暖和的一天，无需对栈操作。
  - 将 `j` 入栈。

然后继续遍历温度数组 `a`，考虑下一天，直至结束。

遍历结束，若栈不为空，则说明栈内的天找不到更暖和的一天，记为 `0`。

代码

```
class Solution {
    public int[] dailyTemperatures(int[] T) {
        int[] ans = new int[T.length];
        Deque<Integer> s = new LinkedList<Integer>();
        for(int i = 0; i < T.length; i++) {
            while(!s.isEmpty() && T[i] > T[s.peek()]) {
                ans[s.peek()] = i - s.pop();
            }
            s.push(i);
        }
        return ans;
    }
}
```

LeetCode 316. 去除重复字母

题意

给你一个字符串 `s` ，请你去除字符串中重复的字母，使得每个字母只出现一次。需保证**返回结果的字典序最小**（要求不能打乱其他字符的相对位置）。

示例

```
输入：s = "bcabc"
输出："abc"
```

题解

首先思考这个问题的一个简单版本。给一个字符串删除一个字符，使得字典序最小。

- 解法: 字典序就是字母的大小顺序，我们想字典序最小，那应删除满足 `s[i] > s[i+1]` 的最小位置 `i` 上的字符。

回到这个问题，我们也是想尽可能的删除满足 `s[i] > s[i+1]` 的最小位置 `i` 上的字符，如果每次都是遍历一遍字符串删除一个字符，这样时间复杂度可能退化到 `O(n^2)`。

优化方法: 单调栈。

单调栈中存放的是字符，从左往右遍历字符串 `s`， 设当前遍历到字符串的位置 `i`，栈顶字符为`c`，考虑 `s[i]` 和 栈顶字符的大小关系、位置 `i` 的字符不在栈中，可分为两种。

- 若 `c > s[i]` **并且** 位置 `i` 的字符不在栈中 **并且** 在位置 `i` 后面还存在字符 `c`，那么将 `c` 从栈中弹出。重复这个过程，直到 `c > s[i]` 不成立 或者 栈为空。
- 不满足上述条件，直接将 `s[i]` 放入栈中。

继续遍历字符串 `s`，直至结束，最后栈中的字符就是题目要求的字典序最小的字符串。

代码

```
class Solution {
    public String removeDuplicateLetters(String s) {
        int[] count = new int[30];
        for (int i = 0; i < s.length(); i++) {
            count[s.charAt(i) - 'a']++;
        }
        boolean[] vis = new boolean[30];
        StringBuffer ans = new StringBuffer();
        for (int i = 0; i < s.length(); i++) {
            int c = s.charAt(i) - 'a';
            if (!vis[c]) {
                while ((ans.length() > 0) &&
(count[ans.charAt(ans.length() - 1) - 'a'] > 0)
                        && ((ans.charAt(ans.length() - 1) - 'a') > c))
{
                    vis[ans.charAt(ans.length() - 1) - 'a'] = false;
```

```
                    ans.deleteCharAt(ans.length() - 1);
                }
                vis[c] = true;
                ans.append(s.charAt(i));
            }
            count[c]--;
        }
        return ans.toString();
    }
}
```

习题推荐

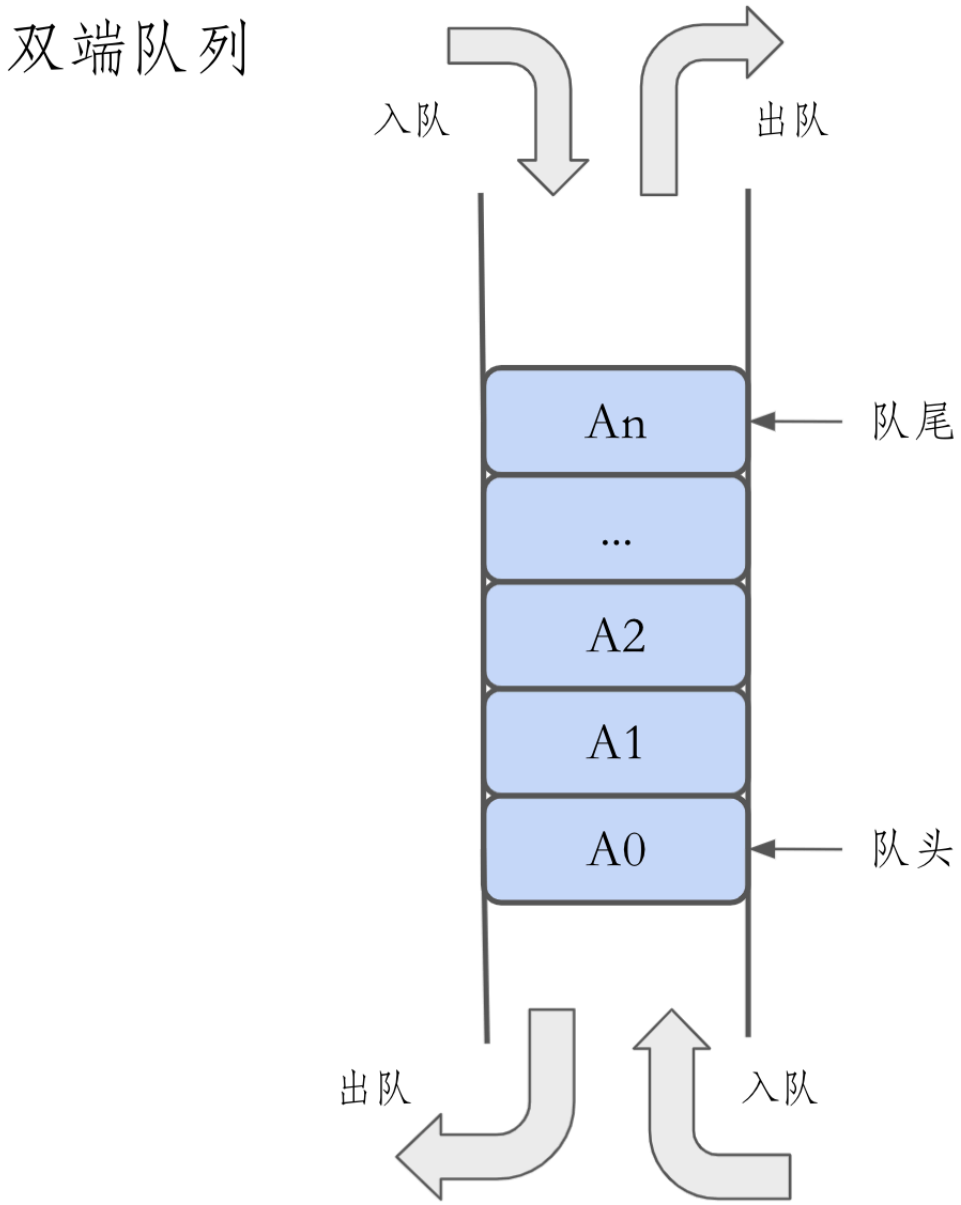
1. LeetCode 496. 下一个更大元素 I
2. LeetCode 1475. 商品折扣后的最终价格
3. LeetCode 503. 下一个更大元素 II

双端队列 & 滑动窗口

介绍

双端队列是普通队列的加强版，区别于队列只能从队头出队，队尾入队；双端队列既可以在队头入队和出队，也可以在队尾入队和出队。

下图是双端队列的的图解，可以看出，双端队列既可以在队头入队和出队，也可以在队尾入队和出队。



例题

LeetCode 239. 滑动窗口最大值

题意

给你一个整数数组 nums，有一个大小为 k 的滑动窗口，从数组的最左侧移动到数组的最右侧。你只可以看到在滑动窗口内的 k 个数字。滑动窗口每次只向右移动一位。返回滑动窗口移动过程中每个窗口中的最大值。

示例

```
输入：nums = [1,3,-1,-3,5,3,6,7], k = 3
输出：[3,3,5,5,6,7]
解释：
滑动窗口的位置                最大值
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
```

题解

滑动窗口经典题，维护一个单调的双端队列，为了方便，双端队列里面的数组的下标，从前往后遍历数组，需要实现两个功能。

- 若 **队头位置下标** 和 **当前遍历位置下标** 的距离大于 `k`，则删除队头元素，**保证了队头下标在当前滑动窗口内**。
- 若 **队尾位置下标对应的值** 小于 **当前位置的值**，则删除队尾元素，**保证了队头下标对应的值是最大的**。

其次将当前遍历位置下标放入双端队列，然后遍历数组的下一个位置，直至结束。

代码

```java
class Solution {
    public int[] maxSlidingWindow(int[] nums, int k) {
        Deque<Integer> q = new LinkedList<>();
        int ans[] = new int[nums.length - k + 1];
        for (int i = 0; i < k; i++) {
            while (!q.isEmpty() && nums[q.getLast()] < nums[i]) {
                q.removeLast();
            }
            q.addLast(i);
        }
        ans[0] = nums[q.getFirst()];
        for (int i = k; i < nums.length; i++) {
            while(!q.isEmpty()  && (i - q.getFirst() >= k)) {
                q.removeFirst();
            }
            while(!q.isEmpty() && nums[q.getLast()] < nums[i]) {
                q.removeLast();
```

```
            }
            q.addLast(i);
            ans[i - k + 1] = nums[q.getFirst()];
        }
        return ans;
    }
}
```

LeetCode 3. 无重复字符的最长子串

题意

给定一个字符串，请你找出其中不含有重复字符的 **最长子串** 的长度。

示例

```
输入: s = "abcabcbb"
输出: 3
解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。
```

题解

观察样例，我们可以发现，依次递增地枚举子串的起始位置，那么合法的结束为止一定是递增的，因为对于起始位置 `i-1` ，假设其不含有重复字符的最远右位置 `j`；那么对于起始位置为 `i` 的子串，因为 `[i-1, j]` 不含有重复字符，其不含有重复字符的最远右位置一定大于等于 `i`，因此我们考虑使用滑动窗口来解决本题。

我们可以固滑动窗口的右边界，找到最远的不含有重复字符的左边界 ，根据上面我们观察得到的性质可以，不含有重复字符的左边界是**非递减**的。

代码具体实现上我们可以用 双端队列实现滑动窗口，辅助数组`cnt` 统计窗口内每个字符出现的次数 ，来判断窗口是否有重复的字符。

代码

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        char[] cnt = new char[128];
        LinkedList<Character> q = new LinkedList<Character>();
        int ans = 0;
        for (int i = 0; i < s.length(); i++) {
            q.add(s.charAt(i));
            cnt[s.charAt(i)]++;
            while (cnt[s.charAt(i)] > 1) {
```

```
                char frontC = q.pollFirst();
                cnt[frontC]--;
            }
            ans = Math.max(ans, q.size());
        }
        return ans;
    }
}
```

习题推荐

LeetCode 209. 长度最小的子数组

往期精彩文章

ACM金牌选手讲解LeetCode算法《线性表》