

算法13 栈与队列2

单调栈

单调栈是面试中非常容易考到的知识点，也是整个栈学习中的重点。单调栈有三个模板，分别介绍如下：

（以下说的单调上升下降都是指从栈底到栈顶）

模板1 push的时候栈顶是答案

考虑：给定一个数组 a ，求每个数左边离它最近的且比它小的数的下标。如果不存在则标记为 -1 。

例如 $a = [1, -1, 3, 2]$ ，设下标从0开始，设答案数组是 r 。

开一个严格上升的栈，初始栈空

从左到右扫描 a ，首先遇到了 $a[0] = 1$ ，push的时候知道答案，此时栈空，也就是答案不存在，所以 $r[0] = -1$ ，栈变为 $[1]$

接着遇到 $a[1] = -1$ ，push会违反单调性，从而pop栈顶，而push的时候知道答案，此时栈空，也就是答案不存在，所以 $r[1] = -1$ ，栈变为 $[-1]$

接着遇到 $a[2] = 3$ ，push不会违反单调性，而push的时候知道答案，此时栈顶是 -1 ，所以 $r[2] = 1$ （即 -1 的下标），栈变为 $[-1, 3]$

接着遇到 $a[3] = 2$ ，push会违反单调性，从而pop栈顶，而push的时候知道答案，此时栈顶是 -1 ，所以 $r[3] = 1$ ，栈变为 $[-1, 2]$

所以最终答案就是 $r = [-1, -1, 1, 1]$

模板2 pop的时候知道pop值的答案

考虑：给定一个数组 a ，求每个数右边离它最近的且比它小的数的下标。如果不存在则标记为 -1 。

例如 $a = [2, 3, -1, -1, -2]$ ，设下标从0开始，设答案数组是 r 。

开一个非严格上升的栈，初始栈空

从左到右扫描 a ，首先遇到了 $a[0] = 2$ ，push不会违反单调性，将其push进栈，栈变为 $[2]$

接着遇到 $a[1] = 3$ ，push不会违反单调性，将其push进栈，栈变为 $[2, 3]$

接着遇到 $a[2] = -1$ ，push会违反单调性，pop的时候知道答案，pop了3，其对应的答案就是 $a[2] = -1$ ，所以 $r[1] = 2$ （ $r[1]$ 的1是要pop的数字3的下标）；再pop了2，其对应的答案也是 $a[2] = -1$ ，所以 $r[0] = 2$ 。再将 $a[2]$ 进栈，此时栈变为 $[-1]$

接着遇到 $a[3] = -1$ ，push不违反单调性，此时栈变为 $[-1, -1]$

接着遇到 $a[4] = -2$ ，push会违反单调性，pop了 -1 ，其对应的答案就是 $a[4] = -2$ ，从而 $r[3] = 4$ （ $r[3]$ 的3是pop的 -1 的下标）；再pop了 -1 ，其对应的答案也是 $a[4] = -2$ ，从而 $r[2] = 4$ 。再将 $a[4]$ 进栈，栈变为 $[-2$

最后栈内的元素都是答案不存在的元素，从而 $r[4] = -1$

最终答案 $r = [2, 2, 4, 4, -1]$

注意：

按上面的算法，细心的读者可以发现，每次push的时候，pop完违反的元素之后，push的元素的左边最近的小于等于它的元素恰好就是栈顶（当然如果栈空那就不存在答案）。所以，模板2是能看两边的，但是，如果右边看的是最近的小于的数的下标，左边就只能看最近的小于等于的数的下标，也就是一边看严格小于，另一边只能看非严格小于等于。

模板3 单调栈 + 二分

这个模板比较特殊，能高效的解决一类不常见的问题。

考虑：给定一个数组 a ，求每个数左边离它最远的且比它小的数的下标。如果不存在则标记为 -1 。

例如 $a = [4, 2, 3, 4]$ ，设下标从0开始，设答案数组是 r 。

开一个非严格单调下降的栈，初始栈空，[

遍历到 $a[0] = 4$ ，由于任意时刻栈都是非严格单调下降的，所以其答案通过对栈进行二分得到。当前栈空，所以 $r[0] = -1$ ；接着将 $a[0]$ 入栈，栈变为 $[4$

接着遇到 $a[1] = 2$ ，对栈进行二分得其答案，所以 $r[1] = -1$ ，push不会违反单调性，从而栈变为 $[4, 2$

接着 $a[2] = 3$ ，对栈进行二分，得到其答案是 $a[1] = 2$ ，所以 $r[2] = 1$ ；push会违反单调性，从而不push（这也很好理解， $a[1] = 2$ 比 $a[2] = 3$ 离得更远，数值还更小，那当然没必要push进 $a[2]$ 了），栈为 $[4, 2$

接着 $a[3] = 4$ ，对栈进行二分，所以 $r[3] = 1$ ；push会违反单调性，所以不push

最终答案 $r = [-1, -1, 1, 1]$

总结

前两个模板时间是线性的，而最后一个时间是 $O(n \log n)$ 的，对于这种特殊的问题来讲已经相当高效。

单调队列

单调队列几乎只用来解决一个问题，就是滑动窗口的最值问题及其变种。

例题：https://blog.csdn.net/qq_46105170/article/details/103813612

给定一个长 n 数组 a 和一个正整数 k ，返回一个数组，每个数字代表原数组每 k 个数的滑动窗口的最大值。

具体思路不难，直接看博客。这个方法的精髓在于，队列里每次都会清除掉各种原因不可能成为答案的值，清除完之后发现队列恰好只需要满足单调性就行了。

代码如下：

```
1  class Solution {
2      public:
3          vector<int> maxSlidingWindow(vector<int>& a, int k) {
4              int n = a.size();
5              vector<int> res;
6              deque<int> dq;
7              for (int i = 0; i < n; i++) {
8                  // pop掉不可能成为答案的值
9                  while (dq.size() && a[dq.back()] <= a[i]) dq.pop_back();
10                 // pop掉出界的值
11                 if (dq.size() && i - dq.front() >= k) dq.pop_front();
12                 // push当前值
13                 dq.push_back(i);
14                 // 每一轮的队尾就是答案
15                 if (i >= k - 1) res.push_back(a[dq.front()]);
16             }
17
18             return res;
19         }
20     };
```

作业

Leetcode 496 907 42 1201 1060 84 85 739 239 862