如何使用单调栈解题

Stars 79k 知乎 @labuladong 公众号 @labuladong B站 @labuladong

相关推荐：

- 回溯算法解题套路框架
  - 动态规划解题套路框架

读完本文，你不仅学会了算法套路，还可以顺便去 LeetCode 上拿下如下题目：

496.下一个更大元素I

503.下一个更大元素II

739.每日温度

栈（stack）是很简单的一种数据结构，先进后出的逻辑顺序，符合某些问题的特点，比如说函数调用栈。

单调栈实际上就是栈，只是利用了一些巧妙的逻辑，使得每次新元素入栈后，栈内的元素都保持有序（单调递增或单调递减）。

听起来有点像堆（heap）？不是的，单调栈用途不太广泛，只处理一种典型的问题，叫做 Next Greater Element。本文用讲解单调队列的算法模版解决这类问题，并且探讨处理「循环数组」的策略。

单调栈模板

首先，看一下 Next Greater Number 的原始问题，这是力扣第 496 题「下一个更大元素 I」：

给你一个数组，返回一个等长的数组，对应索引存储着下一个更大元素，如果没有更大的元素，就存 -1。

函数签名如下：

```
vector<int> nextGreaterElement(vector<int>& nums);
```

比如说，输入一个数组 `nums = [2,1,2,4,3]`，你返回数组 `[4,2,4,-1,-1]`。

解释：第一个 2 后面比 2 大的数是 4; 1 后面比 1 大的数是 2；第二个 2 后面比 2 大的数是 4; 4 后面没有比 4 大的数，填 -1；3 后面没有比 3 大的数，填 -1。

这道题的暴力解法很好想到，就是对每个元素后面都进行扫描，找到第一个更大的元素就行了。但是暴力解法的时间复杂度是 `O(n^2)`。

这个问题可以这样抽象思考：把数组的元素想象成并列站立的人，元素大小想象成人的身高。这些人面对你站成一列，如何求元素「2」的 Next Greater Number 呢？很简单，如果能够看到元素「2」，那么他后面可见的第一个人就是「2」的 Next Greater Number，因为比「2」小的元素身高不够，都被「2」挡住了，第一个露出来的就是答案。

这个情景很好理解吧？带着这个抽象的情景，先来看下代码。

```
vector<int> nextGreaterElement(vector<int>& nums) {
    vector<int> res(nums.size()); // 存放答案的数组
    stack<int> s;
    // 倒着往栈里放
    for (int i = nums.size() - 1; i >= 0; i--) {
        // 判定个子高矮
        while (!s.empty() && s.top() <= nums[i]) {
            // 矮个起开，反正也被挡着了。。。
            s.pop();
        }
        // nums[i] 身后的 next great number
        res[i] = s.empty() ? -1 : s.top();
        //
        s.push(nums[i]);
    }
    return res;
}
```

这就是单调队列解决问题的模板。for 循环要从后往前扫描元素，因为我们借助的是栈的结构，倒着入栈，其实是正着出栈。while 循环是把两个「个子高」元素之间的元素排除，因为他们的存在没有意义，前面挡着个「更高」的元素，所以他们不可能被作为后续进来的元素的 Next Great Number 了。

这个算法的时间复杂度不是那么直观，如果你看到 for 循环嵌套 while 循环，可能认为这个算法的复杂度也是 `O(n^2)`，但是实际上这个算法的复杂度只有 `O(n)`。

分析它的时间复杂度，要从整体来看：总共有 `n` 个元素，每个元素都被 `push` 入栈了一次，而最多会被 `pop` 一次，没有任何冗余操作。所以总的计算规模是和元素规模 `n` 成正比的，也就是 `O(n)` 的复杂度。

问题变形

单调栈的使用技巧差不多了，来一个简单的变形，力扣第 739 题「每日温度」：

给你一个数组 `T`，这个数组存放的是近几天的天气气温，你返回一个等长的数组，计算：**对于每一天，你还要至少等多少天才能等到一个更暖和的气温；如果等不到那一天，填 0**。

函数签名如下：

```
vector<int> dailyTemperatures(vector<int>& T);
```

比如说给你输入 `T = [73,74,75,71,69,76]`，你返回 `[1,1,3,2,1,0]`。

解释：第一天 73 华氏度，第二天 74 华氏度，比 73 大，所以对于第一天，只要等一天就能等到一个更暖和的气温，后面的同理。

这个问题本质上也是找 Next Greater Number，只不过现在不是问你 Next Greater Number 是多少，而是问你当前距离 Next Greater Number 的距离而已。

相同的思路，直接调用单调栈的算法模板，稍作改动就可以，直接上代码吧：

```
vector<int> dailyTemperatures(vector<int>& T) {
    vector<int> res(T.size());
    // 这里放元素索引，而不是元素
    stack<int> s;
    /* 单调栈模板 */
    for (int i = T.size() - 1; i >= 0; i--) {
        while (!s.empty() && T[s.top()] <= T[i]) {
            s.pop();
        }
        // 得到索引间距
        res[i] = s.empty() ? 0 : (s.top() - i);
        // 将索引入栈，而不是元素
        s.push(i);
    }
    return res;
}
```

单调栈讲解完毕，下面开始另一个重点：如何处理「循环数组」。

如何处理环形数组

同样是 Next Greater Number，现在假设给你的数组是个环形的，如何处理？力扣第 503 题「下一个更大元素 II」就是这个问题：

比如输入一个数组 `[2,1,2,4,3]`，你返回数组 `[4,2,4,-1,4]`。拥有了环形属性，**最后一个元素 3 绕了一圈后找到了比自己大的元素 4**。

一般是通过 % 运算符求模（余数），来获得环形特效：

```
int[] arr = {1,2,3,4,5};
int n = arr.length, index = 0;
while (true) {
    print(arr[index % n]);
    index++;
}
```

这个问题肯定还是要用单调栈的解题模板，但难点在于，比如输入是 `[2,1,2,4,3]`，对于最后一个元素 3，如何找到元素 4 作为 Next Greater Number。

对于这种需求，常用套路就是将数组长度翻倍：

这样，元素 3 就可以找到元素 4 作为 Next Greater Number 了，而且其他的元素都可以被正确地计算。

有了思路，最简单的实现方式当然可以把这个双倍长度的数组构造出来，然后套用算法模板。但是，**我们可以不用构造新数组，而是利用循环数组的技巧来模拟数组长度翻倍的效果**。

直接看代码吧：

```
vector<int> nextGreaterElements(vector<int>& nums) {
    int n = nums.size();
    vector<int> res(n);
    stack<int> s;
    // 假装这个数组长度翻倍了
    for (int i = 2 * n - 1; i >= 0; i--) {
        // 索引要求模，其他的和模板一样
        while (!s.empty() && s.top() <= nums[i % n])
            s.pop();
        res[i % n] = s.empty() ? -1 : s.top();
        s.push(nums[i % n]);
    }
    return res;
}
```

这样，就可以巧妙解决环形数组的问题，时间复杂度 `O(N)`。

如果本文对你有帮助，请三连，这次一定。

刷算法，学套路，认准 labuladong，公众号和 在线电子书 持续更新最新文章。

本小抄即将出版，微信扫码关注公众号，后台回复「小抄」限时免费获取，回复「进群」可进刷题群一起刷题，带你搞定 LeetCode。

==其他语言代码==