

22 | 栈与单调栈：最大矩形面积

2020-03-07 胡光

人人都能学会的编程入门课

[进入课程 >](#)



讲述：胡光

时长 12:49 大小 11.75M



你好，我是胡光，欢迎回来。

上节课我们讲了单调队列这种具有单调性的结构，并且说明了单调队列适合：维护**队列处理顺序**中的区间最大值，并且我还提到单调队列只是一个铺垫，搞清楚了单调队列的内容，才能更好地学习新的数据结构。

今天我将带你学习一种队列和单调队列的兄弟数据结构，它的性质也很有趣，就是：栈与单调栈。学习这个数据结构的时候呢，我还是要再次强调一下那句话：数据结构，就是定义一种性质，并且维护这种性质。



今日任务

在正式开始学习之前呢，先来看一下今天这 10 分钟的任务吧。

假设有一面木板墙，每块木板的宽度都是 1，你现在想在木板墙上，沿着平行于地面的方向，切割出一块矩形区域。问题来了，如果给出了每一块木板的高度，那么如何切出面积最大的矩形区域？矩形木板墙如下图所示：

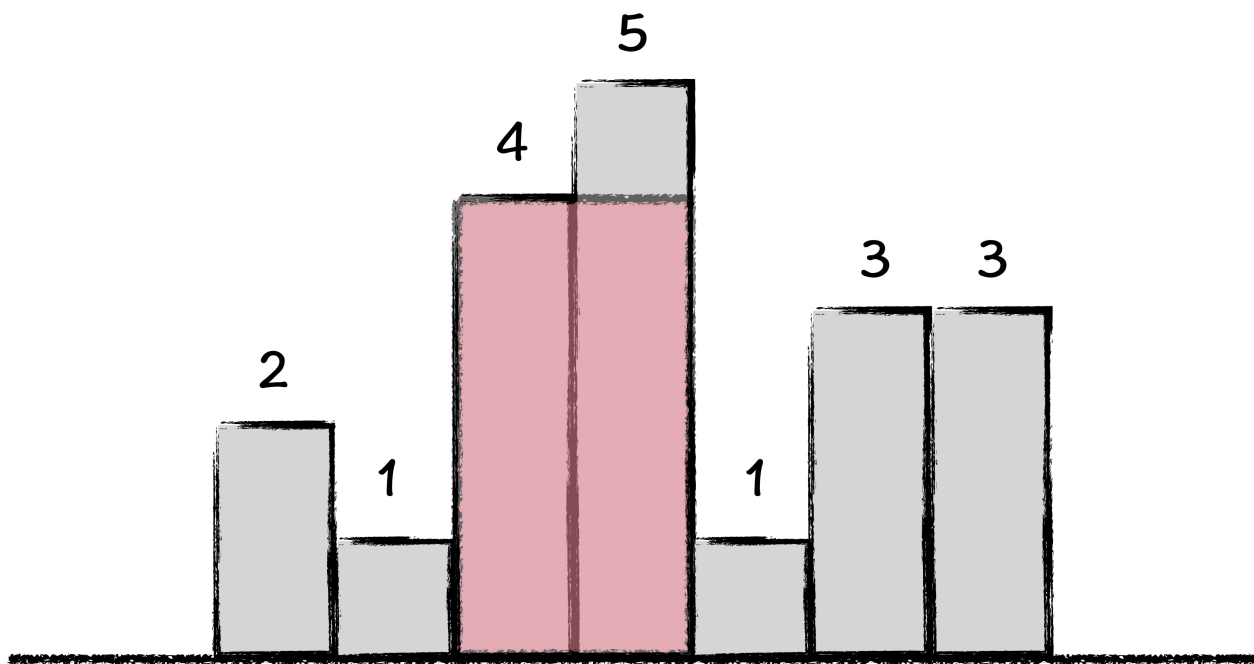


图1：木板墙示意图

如你所见，图中有 7 块木板，每块木板的高度分别为：2、1、4、5、1、3、3。经过尝试，我们发现最大矩形就是红色阴影部分所示，也就是切割了高度为 4 和 5 两块木板，形成了一个高度为 4，宽度为 2 的矩形区域，这个最大面积为 8。

显而易见的结论：就是**切下来的最大的矩形，一定是以最大矩形所在区域最短那块木板作为其高度值**。如果不是这样的话，我们就可以提升一点点高度，让切下来的部分更大一点儿。

有了如上这个结论，我们就可以枚举每一块木板，每次都当前木板作为高度，就是把当前这块木板，当成是切出来的矩形区域中的最矮的木板，然后向左边和右边分别做延伸，切出此时的最大矩形区域。当把所有木板都试过一遍后，我们在所有枚举结果中比较出最大值，这个最大值就是我们要求的最大矩形面积。如果木板的个数为 n ，那这种做法的时间复杂度接近于 $O(n^2)$ 。

而今天，我要给你讲的方法，能将这个时间复杂度降低到 $O(n)$ ，这种结构方法就像我们上次讲的单调队列一样有趣。接下来，就让我们一点点地揭开这个结构神秘的面纱。

必知必会，查缺补漏

想要完成今天这个任务呢，你必须掌握接下来我要教给你的一个新的数据结构：单调栈。

1. 栈：维护一种完全包含关系的结构

首先让我们来认识一下最简单的栈结构，所谓栈结构，你可以想象成只有一个口的羽毛球桶，羽毛球只能从唯一的一个口放入和取出。我们把编号 1、2、3 三个羽毛球按顺序放进球桶后，如果想取出来，那么这些球被取出来的顺序一定是编号 3、2、1。也就是说后放入的羽毛球，在取出的时候，会最先被取出来，它们放入和取出的顺序是相反的。

如果说，上一节我们学习的队列结构是**先进先出**的结构，那么今天我们学习的栈就是一种**后进先出**的结构。栈和队列一样，都是计算机中，用来规范处理顺序的基础结构。

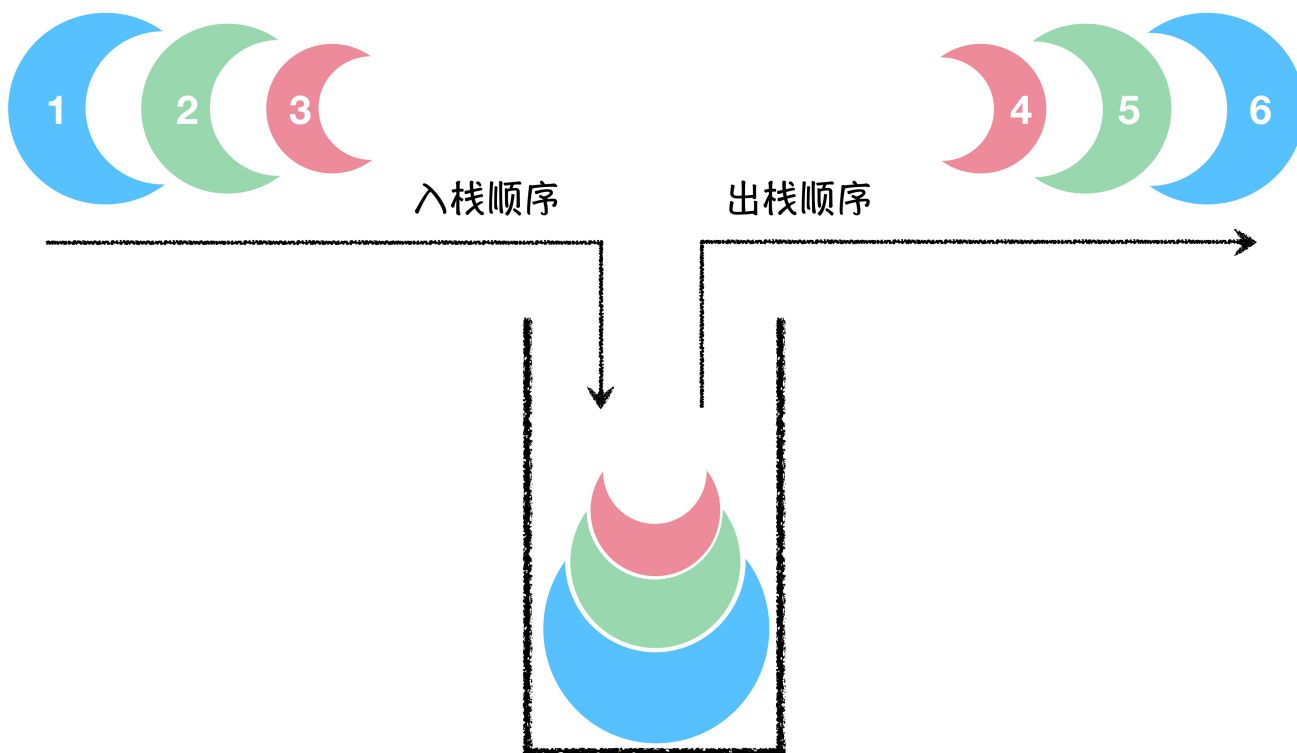


图2：栈结构示意图

图中所示，入栈顺序分别是 蓝、绿、红，那么出栈顺序就一定是红、绿、蓝。图中每一个颜色的方块上标注的数字，就是每一个方块入栈及出栈的顺序。

从示意图中，我们还可以观察到一个有趣的事情，在顺序上而言，红色方块被绿色方块包裹着，绿色方块被蓝色方块包裹着。这种结构，像是程序的调用过程，如果把蓝色方块，看成是主函数的话，那么绿色方块就是主函数中调用的一个函数 A，红色方块就是 A 函数中调用的另外一个函数 B，三个函数调用的顺序是主函数、函数 A、函数 B。

而它们的执行结束顺序恰恰是相反的，首先是函数 B 结束，然后是函数 A 结束，最后是主函数结束。实际上，我们计算机用来维护函数执行的底层系统，就是用的这种栈结构。

你可以认为，栈结构本身维护的是一种完全包含关系。其实函数之间的运行，就是一种完全包含关系，只要在主函数中调用函数 B，那么函数 B 一定在主函数结束之前结束，这就可以视为是主函数包含函数 B。

2. 单调栈：维护最近大于或小于关系的结构

我们了解了最简单的栈结构以后，接下来，就来让我们学习一种栈的升级产物：单调栈。理解单调栈的最简单方法，就是基于对单调队列的理解去学习它。如果你单调队列还没有掌握，那我建议你再好好看一看上节课中关于单调队列的相关内容。

我先问你一个问题，队列结构和栈结构到底有什么区别？你可能会说，它们唯一的区别就是，队列是从一端进另外一端出，栈是在同一端进出。

那我再问你一个问题，堵住出口的单调队列，和栈有什么区别？你会发现，好像没什么区别了，单调队列为了维护单调性，在入队列的时候，也会将违反单调性的元素弹出，所以，这就相当于栈的从同一端进出。

好了，如果你明白这些问题，我可以明确地告诉你：堵住出口的单调队列，就是我们今天要学习的“单调栈”。

既然堵住了单调队列的出口，那么这种所谓单调栈的结构，就再也维护不了区间最大值了。那它维护的是什么呢？让我们以单调递减栈为例。

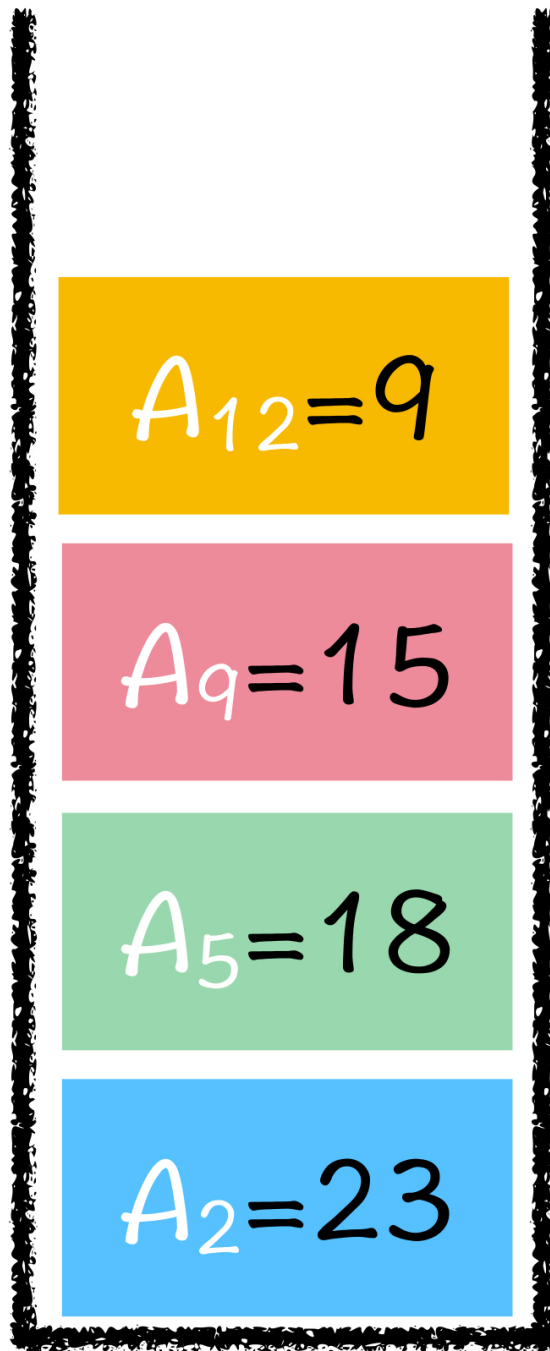


图3：单调栈示意图

如图所示，这是我假设的一种单调栈中元素的情况：序列第 12 号元素入栈以后，单调栈中只剩下了 4 个元素，从栈底到栈顶，值分别为 23、18、15 和 9，分别对应了原序列的第 2 号、第 5 号，第 9 号 以及 第 12 号。

关于单调栈性质的思考，我们只需要重点关注栈顶的 12 号元素和 9 号元素之间的关系即可。如果 12 号元素入栈以后，为了保持栈中的单调递减性，它最终放在了 9 号元素上面，那说明什么呢？是不是说明从 9 号元素到 12 号之间的元素值，均小于 12 号元素值呢？也

就是说，10 号、11 号这两个元素的值，我们虽然不知道具体是多少，可这两个元素的值，肯定比 9 号元素小，甚至也比 12 号元素小。否则，按照单调栈的入栈规则，12 号元素和 9 号元素就不可能在栈中相邻。

其实，说到这里，你应该已经对单调栈的性质有所感觉了。如果我们将一个元素压入单调递减栈，那么这个元素会落在离它最近，且比它大的元素上面。就像上面的例子中，当 12 号元素入栈以后，它落在了 9 号元素上面，说明从 12 号元素向前找，9 号元素是第一个比 12 号元素值大的元素。

如果说单调队列是维护区间最值的高效结构，单调栈就是维护最近大于或小于关系的高效结构。如果想要维护最近大于关系，就建立一个单调递减栈，然后将每个元素依次入栈，在这个过程中，我们就可以统计得到每一个元素之前离它最近的，且大于它的元素。那要是想维护最近小于关系呢？就建立一个单调递增栈就好了！

至此，我们就掌握了单调栈的基本性质了。

一起动手，搞事情

今天的思考题呢，跟括号匹配有关系。任务很简单，就是给你一串括号序列，括号序列中可能包含小括号 ()，中括号 [] 或者 大括号 {}，你需要写程序，判断这个括号序列是否合法。只要括号之间，没有交错重叠的情况，就是合法的括号序列。

下列给出了一些合法的序列的示例：

```
1  ({})  
2  []([]){()}
```

 复制代码

下面是一些非法的括号序列的示例：

```
1  ([ ]  
2  (( ){ }
```

 复制代码


通过观察括号序列，你会发现合法的括号序列，其实就是一种完全包含的结构，关于这种结构合法性的判断，和我们今天讲的栈结构有什么关系呢？开动你聪明的大脑，思考一下吧！

最大矩形面积

最后我们回到今天的任务，先来回顾一下之前所说的解题过程：我们通过枚举每一块木板作为切割出的木板墙的高度，每次都需要向左边和右边分别做查找，一直找到一块高度小于当前木板高度的位置，这样就确定了切割木板墙的长度。

以图 1 中高度为 4 的木板为例，我们通过向左延伸查找，发现左边第一块就比它短，这样就确定了向左延伸的长度是 0；往右延伸查找，发现第二块木板比它短，也就是向右延伸的长度是 1。说到这里，你会发现，上面这个过程，不就是我们之前所说的，维护最近小于关系么！只需要建立一个单调递增栈，就可以完成这个任务！

下面，是一份我给出的示例代码：

 复制代码

```
1 #define MAX_N 1000
2 #define max(a, b) ((a) > (b) ? (a) : (b))
3 int s[MAX_N + 5], top;
4 int l[MAX_N + 5], r[MAX_N + 5];
5 int max_matrix_area(int *h, int n) {
6     h[0] = h[n + 1] = -1;
7     top = -1, s[++top] = 0;
8     // 找到每一块木板，左边第一块比其矮的木板编号
9     for (int i = 1; i <= n; i++) {
10         while (top >= 0 && h[s[top]] >= h[i]) --top;
11         l[i] = s[top];
12         s[++top] = i;
13     }
14     // 找到每一块木板，右边第一块比其矮的木板编号
15     top = -1, s[++top] = n + 1;
16     for (int i = n; i >= 1; i--) {
17         while (top >= 0 && h[s[top]] >= h[i]) --top;
18         r[i] = s[top];
19         s[++top] = i;
20     }
21     // 在所有木板中，找到面积最大的矩形
22     int ans = 0;
23     for (int i = 1; i <= n; i++) {
24         ans = max(ans, (r[i] - l[r] - 1) * h[i]);
25     }
26     return ans;
27 }
```

如上代码所示，`max_matrix_area` 函数传入两个参数，木板高度数组首地址 `h`，和木板数量 `n`。代码中的 `s` 数组，后续的作用就是模拟单调栈，`top` 代表了栈顶元素的下标。

你需要注意的是，代码中假设木板的编号是从 1 到 `n` 的，然后，在数组的 0 位及 `n + 1` 位分别加入两块高度为 -1 的虚拟木板，这是边界控制的一种技巧。也就是说，在每块木板向左搜索的时候，最远也就搜索到 0 号位就停止了，向右搜索的时候呢，最远搜索到 `n + 1` 位也就停止了。通过加入虚拟木板，代码中就少了相关的边界条件判断，这是一种很实用的技巧，你一定要理解和掌握。

课程小结

以上就是我们今天要学习的单调栈的内容，关于单调栈，其实你只需要对比着单调队列进行学习和记忆即可，记住以下两点：

1. 单调栈是用来维护最近大于或小于关系的数据结构。
2. 单调栈就是堵住出口的单调队列，所以其时间复杂度与单调队列一致，平均到每个处理元素上，都是 $O(1)$ 的时间复杂度。

好了，单调栈的知识，就讲到这里了。我是胡光，我们下期见。

课程学习计划

关注极客时间服务号 每日学习签到

月领 25+ 极客币

【点击】保存图片，打开【微信】扫码>>>



上一篇 21 | 队列与单调队列：滑动区间最大值

下一篇 23 | 深入理解：容斥原理与递推算法

精选留言 (4)

写留言



胖胖胖

2020-03-08

作业打卡：

```
#define MAX 1000
```

```
char stack[MAX];
```

```
int top = 0;
```

```
int f(char* s){...
```

展开 ∨

作者回复: 不错！你理解的很到位！



2



HappyJoo

2020-03-07

老师，第二行：#define max(a, b) ((a) > (b) : (a) : (b))

应该是：#define max(a, b) ((a) > (b) ? (a) : (b))

老师你快醒醒啊，周六了不用工作了！！

展开 ∨

作者回复: yes，你是对的，另外你太皮了。_-|||，还我的美梦。



HappyJoo

2020-03-07

Leetcode 题号：有效的括号--20、柱状图中最大的矩形--84。

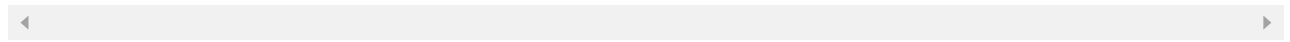
好久以前用python做过第一题，现在早忘了哈哈~

思考：

1, 加入数组两端的 '-1' 就是传说中的哨兵技巧, 加入哨兵可以减少边界条件的判断, ...
展开 ▾

作者回复: 醒了, 第6行是 h!

代码的话, 只是对于思想的一种直白翻译。弄清操作顺序, 再对应到代码即可。



一步

2020-03-07

有效括号这个就是 leetcode <https://leetcode-cn.com/problems/valid-parentheses/> 这道题, 但是没有用到单调栈的性质啊

展开 ▾

作者回复: 对, 不是单调栈的题目。就是一道纯的理解栈性质的题目。

