84.柱状图中最大的矩形 (重要)

亚马逊、微软、字节跳动在半年内面试中考过

柱状图中最大的矩形

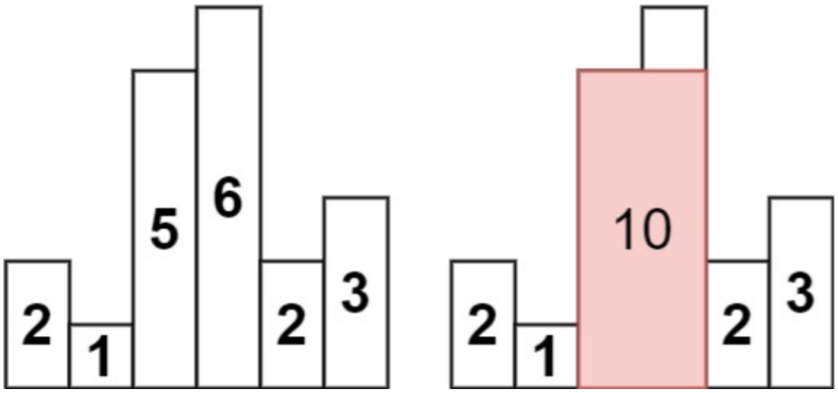
84. 柱状图中最大的矩形

难度 困难 1436 收藏 分享 切换为英文 接收动态 反馈

给定 *n* 个非负整数，用来表示柱状图中各个柱子的高度。每个柱子彼此相邻，且宽度为 1 。

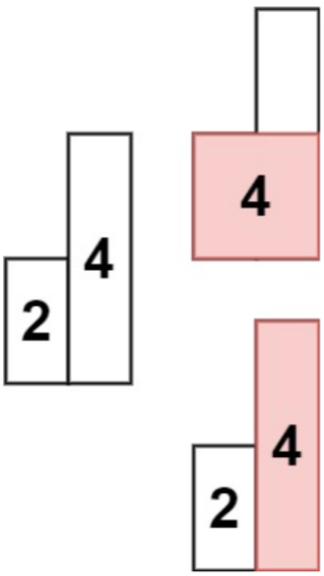
求在该柱状图中，能够勾勒出来的矩形的最大面积。

示例 1:




```
输入: heights = [2,1,5,6,2,3]
输出: 10
解释：最大的矩形为图中红色区域，面积为 10
```

示例 2：



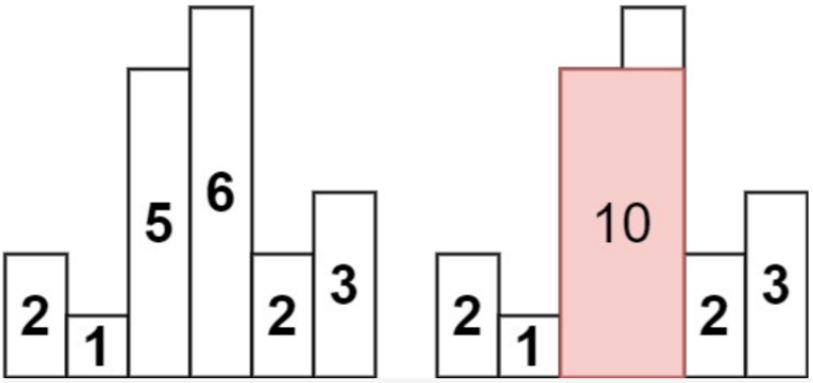

```
输入: heights = [2,4]
输出: 4
```

84. Largest Rectangle in Histogram

难度 困难 1436 收藏 分享 切换为中文 接收动态 反馈

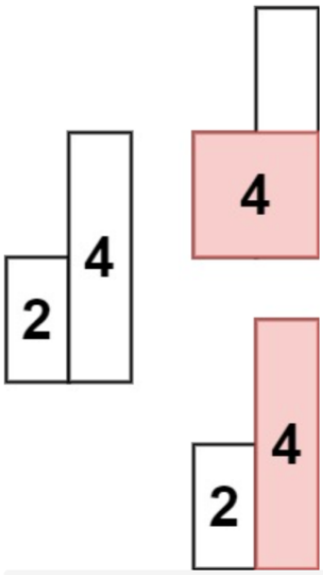
Given an array of integers `heights` representing the histogram's bar height where the width of each bar is `1`, return *the area of the largest rectangle in the histogram*.

Example 1:




```
Input: heights = [2,1,5,6,2,3]
Output: 10
Explanation: The above is a histogram where width of each bar is 1.
The largest rectangle is shown in the red area, which has an area = 10 units.
```

Example 2:




```
Input: heights = [2,4]
Output: 4
```

Constraints:

- `1 <= heights.length <= 10^5`
- `0 <= heights[i] <= 10^4`

思路 1 ： 递增栈

1. 左右边界的查找规律：
    - 如果柱子的高度递减，那么每个柱子的左边界就是第一根柱子，右边界就是本身；
    - 如果柱子的高度递增，那么每个柱子的右边界就是最后一根柱子，左边界就是本身.
2. 面积公式 ： area = right boundary - left boundary - 1
3. 创建栈 stack 用来维护边界；
4. 遍历每一个柱子：
    - 如果当前遍历的柱子比栈顶大，那么当前柱子入栈，说明还没有找到右边界；
    - 如果当前遍历的柱子比栈顶小，那么栈顶的柱子找到了右边界，当前遍历柱子的左边界就是栈中栈顶元素的低下的一个元素的位置,这时候就可以计算面积，并记录最大面积 .
5. 清空栈 ： 遍历结束后，栈中可能有数据，这时候的数据必然是递增的，同时每根柱子的右边界是柱状图的最后一根柱子，每个柱子的左边界就是他本身，计算面积，并记录最大面积.
6. 返回最大面积.

总结：大则入栈，小则处理栈中元素计算最大面积，直到栈中没有比当前遍历的柱子更短的柱子，然后将当前遍历的柱子再入栈.

代码

```
// Java
// Time : 2021 - 07 - 20

class Solution {
    public int largestRectangleArea(int[] heights) {
        int len = heights.length;
        int maxarea = 0;
        Stack<Integer> stk = new Stack<Integer>();

        for (int i = 0; i < len; ++i) {
            // 只要栈不为空，并且当前遍历的数据小于栈顶元素，则说明找到了右边界
            while (!stk.empty() && heights[i] < heights[stk.peek()]) {
                // 右边界 i
                int h = heights[stk.peek()];
                stk.pop();
                // 出栈后，如果栈为空，说明出栈的柱子目前遍历的最短的柱子，否则计算宽度差
                int w = stk.empty() ? i : i - stk.peek() - 1;
```

```
                maxarea = Math.max(h * w, maxarea);
            }
            // 栈为空或者当前遍历的数据大于等于栈顶数据，则入栈，不会执行上面的while循环
            // 保证了栈中的数据总是递增的 比如 0 1 2 2 3 4 4 5 6 ...
            stk.push(i);
        }


        // 处理剩余栈中的数据(递增的数据，右边界是柱状图中最右边的柱子)
        while (!stk.empty()) {
            int h = heights[stk.peek()];
            stk.pop();
            // 出栈后，如果栈为空，说明出栈的柱子目前遍历的最短的柱子，否则计算宽度差
            int w = stk.empty() ? len : len - stk.peek() - 1;
            maxarea = Math.max(h * w, maxarea);
        }

        return maxarea;

    }
}
```

复杂度分析

- 时间复杂度：O(n);
- 空间复杂度：O(n).

#Leetcode/Stack