42.接雨水 (重要)

接雨水

42. 接雨水

难度 困难 2492 收藏 分享 切换为英文 接收动态 反馈

给定 *n* 个非负整数表示每个宽度为 1 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。

示例 1：


```
输入：height = [0,1,0,2,1,0,1,3,2,1,2,1]
输出：6
解释：上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 表示的高度图，在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。
```

示例 2：

```
输入：height = [4,2,0,3,2,5]
输出：9
```

提示：

- `n == height.length`
- `0 <= n <= 3 * 10^4`
- `0 <= height[i] <= 10^5`

通过次数 275,031 提交次数 486,569

42. Trapping Rain Water

难度 困难 2492 收藏 分享 切换为中文 接收动态 反馈

Given `n` non-negative integers representing an elevation map where the width of each bar is `1`, compute how much water it can trap after raining.

Example 1:


```
Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: The above elevation map (black section) is represented by array
[0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.
```

Example 2:

```
Input: height = [4,2,0,3,2,5]
Output: 9
```

Constraints:

- `n == height.length`
- `0 <= n <= 3 * 10^4`
- `0 <= height[i] <= 10^5`

思路 1 ： 栈

1. 类似于括号匹配，蓝色雨水的部分，每次匹配出一对括号（这里是墙），就计算这两堵墙中的水；
2. 用栈保存每堵墙；
3. 当前高度小于栈顶高度，说明这里有会有积水，将墙的高度下标入栈，指针后移；
4. 当前高度大于栈顶高度，说明之前的积水到这里停下，出栈，计算当前墙和栈顶墙之间水的多少；
5. 计算当前的高度和新栈的高度关系，重复第 3 步，直到当前墙的高度不大于栈顶高度或者栈空，然后把当前墙入栈，指针后移。

代码

```
// Java
```

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

public int trap(int[] height) {
        int sum = 0;
        Stack<Integer> stack = new Stack<>();
        int current = 0;

        while(current < height.length){
            // 如果栈不空并且当前指向的高度大于栈顶高度就一直循环
            while(!stack.empty()&&height[current]>height[stack.peek()]){
                int h = height[stack.peek()];    // 取出要出栈道的元素
                stack.pop();     // 出栈道
                if(stack.empty()) //  栈空就出去
                    break;

                int distance = current - stack.peek() - 1;  // 两堵墙之前的距离
                int min = Math.min(height[stack.peek()],height[current]);
                sum = sum + distance * (min - h);
            }

            stack.push(current);     // 当前指向的墙入栈
            current++;  // 指针后移
        }

        return sum;
    }
```

复杂度分析

- 时间复杂度 O(n) : 虽然 while 循环里嵌套了一个 while 循环，但是考虑到每个元素最多访问两次，入栈一次和出栈一次，所以时间复杂度为 O(n).
- 空间复杂度 O(n) : 需要给栈分配 n 个空间.

#Leetcode/Stack