单调栈 (解决next greater element问题)

从后往前开始，找。特别容易理解

而且栈里面的到最后也是从小到大排列了。

栈是很简单的一种是数据结构，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）。

单调栈用途不太广泛，只处理一种典型的问题，叫做next greater element。本节用讲解单调队列的算法模板解决这类问题，并且探讨处理【循环数组】的策略。

下一个更大元素I

给你两个 没有重复元素 的数组 nums1 和 nums2 ，其中nums1 是 nums2 的子集。

请你找出 nums1 中每个元素在 nums2 中的下一个比其大的值。

nums1 中数字 x 的下一个更大元素是指 x 在 nums2 中对应位置的右边的第一个比 x 大的元素。如果不存在，对应位置输出 -1 。

示例 1:

输入: nums1 = [4,1,2], nums2 = [1,3,4,2].
输出: [-1,3,-1]
解释:
对于 num1 中的数字 4 ，你无法在第二个数组中找到下一个更大的数字，因此输出 -1 。
对于 num1 中的数字 1 ，第二个数组中数字1右边的下一个较大数字是 3 。
对于 num1 中的数字 2 ，第二个数组中没有下一个更大的数字，因此输出 -1 。

> 先对nums2处理，找到其的下一个更大的数字

```
class Solution {
    public int[] nextGreaterElement(int[] nums1, int[] nums2) {
        //nums1是nums2的子集
        // 先对nums2进行求取
        int n2 = nums2.length;
        // 开辟一个数组来存储
        HashMap<Integer,Integer> hashMap = new HashMap<>();
        // 单调栈来辅助
        Stack<Integer> stack = new Stack<>();
        // 从后往前走
        for(int i=n2-1;i>=0;i--){
            // 出栈
            while(!stack.isEmpty()&&nums2[i]>=stack.peek()){
                stack.pop();
            }
```

```
            hashMap.put(nums2[i], stack.isEmpty()?-1:stack.peek());
            // 入栈
            stack.push(nums2[i]);
        }
        // 对nums1进行处理
        int n1 = nums1.length;
        int[] res_1 = new int[n1];
        for(int i=0;i<n1;i++){
            res_1[i] = hashMap.get(nums1[i]);
        }
        return res_1;


    }
}
```

下一个更大元素II

定一个循环数组（最后一个元素的下一个元素是数组的第一个元素），输出每个元素的下一个更大元素。数字 x 的下一个更大的元素是按数组遍历顺序，这个数字之后的第一个比它更大的数，这意味着你应该循环地搜索它的下一个更大的数。如果不存在，则输出 -1。

示例 1:

输入: [1,2,1]
输出: [2,-1,2]
解释: 第一个 1 的下一个更大的数是 2；
数字 2 找不到下一个更大的数；
第二个 1 的下一个最大的数需要循环搜索，结果也是 2。

> 循环链表，相当于在后面又复制了一个

```
class Solution {
    public int[] nextGreaterElements(int[] nums) {
        // 正常找
        int n = nums.length;
        // 结果
        int[] res = new int[n];
        // 单调栈
        Stack<Integer> stack = new Stack<>();
        // 开始循环
        for(int i=2*n-1;i>=0;i--){
            // 出栈
            while(!stack.isEmpty()&&nums[i%n]>=stack.peek()){
                stack.pop();
            }
            res[i%n] = stack.isEmpty()?-1:stack.peek();
            // 入栈
            stack.push(nums[i%n]);
        }
        return res;
    }
}
```

Leetcode739每日温度

请根据每日 气温 列表，重新生成一个列表。对应位置的输出为：要想观测到更高的气温，至少需要等待的天数。如果气温在这之后都不会升高，请在该位置用 0 来代替。

例如，给定一个列表 temperatures = [73, 74, 75, 71, 69, 72, 76, 73]，你的输出应该是 [1, 1, 4, 2, 1, 1, 0, 0]。

提示：气温 列表长度的范围是 [1, 30000]。每个气温的值的均为华氏度，都是在 [30, 100] 范围内的整数。

```
class Solution {
    public int[] dailyTemperatures(int[] T) {
        // 结果数组
        int n = T.length;
        int[] res = new int[n];
        // 单调栈 存储下标索引
        Stack<Integer> stack = new Stack<>();
        // 从后往前开始
        for(int i=n-1;i>=0;i--){
            // 出栈
            while(!stack.isEmpty()&&T[i]>T[stack.peek()]){
                stack.pop();
            }
            // 结果记录等待天数
            res[i] = stack.isEmpty()?0:stack.peek()-i;
            // 当前下标索引
            stack.push(i);
        }
        return res;
    }
}
```