456. 132 Pattern(need to review)

link

5,7,1,3...正常是看下一个数在不在5-7, 1-3区间里面.

找一个数字在前面所有的区间中, 我们从尾到头反过来看,

5,7,1,3找到一个数后面的数字比他小, 然后又有比最小的数字大的.

找到一个数在后面降序的数字中

1. 利用缓存, 3的时候, 存储前面数字之前最小的数字, 如果最小的数字是小于1, 并且1>3那么久找到了. 显然1不行, 那么3也不行
2. 然后看1, 7前面最小的是5, 并且7大于1, 所以找到了.

错误的写法: 原因, 写的时候总是想通过当前元素之前的最大最小值判断, 实际上max的值不是一个固定的, 所以要用stack存储, 理解的还不透彻.

```
if(nums.length < 3) return false;
Deque<int[]> stack = new ArrayDeque<>();
int min = nums[nums.length - 1];
int max = Integer.MIN_VALUE;
for(int i = nums.length - 1; i >= 0; --i){
    if(nums[i] < min){
        if(nums[i] < max){
            System.out.println(min);
            return true;
        }
        else{
            min = nums[i];
        }
    }else{
        max = nums[i];
    }
}
return false;
```

一个很好的说明O(n^3 - n)link

例子[1,2,3,4,-4,-3,-5,-1]

```
public boolean find132pattern(int[] nums) {
    int[] arr = Arrays.copyOf(nums, nums.length);
    for (int i = 1; i < nums.length; i++) {
        arr[i] = Math.min(nums[i - 1], arr[i - 1]);
    }
    for (int j = nums.length - 1, top = nums.length;
            j >= 0; j--) {
        if (nums[j] <= arr[j]) continue;
        while (top < nums.length && arr[top] <= arr[j])
            top++;
        if (top < nums.length && nums[j] > arr[top])
            return true;
        arr[--top] = nums[j];
    }

    return false;
}
```

这个题实在是没看懂.....

```
class Solution {
    public boolean find132pattern(int[] nums) {
        int[] min = new int[nums.length];
        min[0] = nums[0];
        for (int j = 1; j < nums.length; j++) {
            min[j] = Math.min(nums[j], min[j-1]);
        }

        Stack<Integer> stack = new Stack<>();
        for (int j = nums.length - 1; j >= 0; j--) {
            while (!stack.empty() && stack.peek() < nums[j]) {
                if (stack.peek() > min[j]) {
                    return true;
                }
                stack.pop();
            }

            stack.push(nums[j]);
        }
        return false;
    }
```

link 官方答案好像容易理解一些

496. Next Greater Element I

类似的题目

link

我写的, 过了 但是理解不太行...

```
public int[] nextGreaterElement(int[] nums1, int[] nums2) {
    Set<Integer> set = new HashSet<>();
    for(int n : nums1){
        set.add(n);
    }
    int l2 = nums2.length;
    Map<Integer, Integer> map = new HashMap<>();
    Deque<Integer> stack = new ArrayDeque<>();
    int min = Integer.MIN_VALUE;
    for(int i = l2 - 1; i >=0; --i){
        while(!stack.isEmpty() && nums2[i] > stack.peek()){
            stack.pop();
        }
        if(!stack.isEmpty() && stack.peek() > nums2[i])
            map.put(nums2[i], stack.peek());
        else
            map.put(nums2[i], -1);
        stack.push(nums2[i]);
    }
    int[] res = new int[nums1.length];
    for(int i = 0; i < nums1.length; ++i){
        res[i] = map.get(nums1[i]);
    }
    return res;
}
```

这个正序好像也可以写link

```
public int[] nextGreaterElement(int[] findNums, int[] nums) {
    Map<Integer, Integer> map = new HashMap<>(); // map from x to next greater
element of x
    Stack<Integer> stack = new Stack<>();
    for (int num : nums) {
        while (!stack.isEmpty() && stack.peek() < num)
            map.put(stack.pop(), num);
        stack.push(num);
    }
    for (int i = 0; i < findNums.length; i++)
        findNums[i] = map.getOrDefault(findNums[i], -1);
    return findNums;
```

```
}
```

503. Next Greater Element II

最开始的O(n^2)的解法, [5,4,3,2,1,6]降序数组是最慢的.

```
int[] res = new int[nums.length];
if(nums.length == 0) return res;
int[] leftMax = new int[nums.length];
int[] rightMax = new int[nums.length];
leftMax[0] = nums[0];
rightMax[nums.length - 1] = nums[nums.length - 1];
for(int i = 1; i < nums.length; ++i){
    leftMax[i] = Math.max(leftMax[i - 1], nums[i - 1]);
}
for(int i = nums.length - 1 - 1; i >= 0; --i){
    rightMax[i] = Math.max(rightMax[i + 1], nums[i + 1]);
}
for(int i = 0; i < nums.length; ++i){
    if(nums[i] >= leftMax[i] && nums[i] >= rightMax[i])
        res[i] = -1;
    else if(nums[i] < rightMax[i]){
        for(int j = i; i < nums.length; ++j){
            if(nums[j] > nums[i]){
                res[i] = nums[j];
                break;
            }
        }
    }else if(nums[i] < leftMax[i]){
        for(int j = 0; j <= i; ++j){
            if(nums[j] > nums[i]){
                res[i] = nums[j];
                break;
            }
        }
    }
}
return res;
```

是圆环形的, 可以用循环2n然后存index(还没完全看懂)

```
public int[] nextGreaterElements(int[] nums) {
    int n = nums.length, res[] = new int[n];
    Arrays.fill(res, -1);
    Stack<Integer> stack = new Stack<>();
    for (int i = 0; i < n * 2; i++) {
        while (!stack.isEmpty() && nums[stack.peek()] < nums[i % n])
            res[stack.pop()] = nums[i % n];
        stack.push(i % n);
    }
```

```
        return res;
    }
```