# 单调队列

解决该类问题的重点维护一个队列，**从队首到队尾是递减的**，队首是最大的。队尾是最小的。

**队尾接受值，队首排出值。**

Java实现用双端队列，前面接收值，后面排出来值。

> 这类题目往往是跟滑动窗口一起出现的，在滑动窗口K的范围内查找最大最小值。

## 1.Leetcode239 滑动窗口最大值

给你一个整数数组 nums，有一个大小为 k 的滑动窗口从数组的最左侧移动到数组的最右侧。你只可以看到在滑动窗口内的 k 个数字。滑动窗口每次只向右移动一位。

返回滑动窗口中的最大值。

示例 1：

输入：nums = [1,3,-1,-3,5,3,6,7], k = 3
输出：[3,3,5,5,6,7]
解释：
滑动窗口的位置　　最大值

---

[1 3 -1] -3 5 3 6 7　　3
1 [3 -1 -3] 5 3 6 7　　3
1 3 [-1 -3 5] 3 6 7　　5
1 3 -1 [-3 5 3] 6 7　　5
1 3 -1 -3 [5 3 6] 7　　6
1 3 -1 -3 5 [3 6 7]　　7

> 求滑动窗口中的最大值，存储索引

```
class Solution {
    public int[] maxSlidingWindow(int[] nums, int k) {
        // 用双端队列实现一个单调队列
        Deque<Integer> queue = new LinkedList<>();
        // 队首最大值中的下标索引
        // 滑动窗口的思路
        int n = nums.length;
        int[] res = new int[n-k+1];
        for(int i=0;i<n;i++){
            while(!queue.isEmpty()&&nums[i]>nums[queue.getLast()]){
                // 队尾的其其小，排出来
                queue.pollLast();
            }
            // 入队列
            queue.offerLast(i);
            // 需要判断队列中的队首是否还在滑动窗口有效范围内
            if(queue.peekFirst()+k<=i){
                //不在有效范围内了 最大值排出来
                queue.pollFirst();
```

```
            }

            // 滑窗中最大值记录从下标2开始
            if(i+1>=k){
                res[i+1-k] = nums[queue.peekFirst()];
            }
        }
        return res;
    }
}
```

## 2.洛谷P1714切蛋糕

今天是小Z的生日，同学们为他带来了一块蛋糕。这块蛋糕是一个长方体，被用不同色彩分成了N个相同的小块，每小块都有对应的幸运值。

小Z作为寿星，自然希望吃到的第一块蛋糕的幸运值总和最大，但小Z最多又只能吃M小块(M≤N)的蛋糕。

吃东西自然就不想思考了，于是小Z把这个任务扔给了学OI的你，请你帮他从这N小块中找出连续的k块蛋糕(k≤M)，使得其上的幸运值最大。

**输入格式**

输入文件cake.in的第一行是两个整数N,M。分别代表共有N小块蛋糕，小Z最多只能吃M小块。

第二行用空格隔开的N个整数，第i个整数Pi代表第i小块蛋糕的幸运值。

**输出格式**

输出文件cake.out只有一行，一个整数，为小Z能够得到的最大幸运值。

**输入输出样例**

**输入 #1**复制

```
5 2
1 2 3 4 5
```

**输出 #1**复制

```
9
```

**输入 #2**复制

```
6 3
1 -2 3 -4 5 -6
```

**输出 #2**复制

```
5
```

> 维护一个最小的值；在单调队列中，队首最小，队首到队尾单调递增

```

```

```java
import java.util.*;


 class Main{
    // 主函数
public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    int m = sc.nextInt();
    int[] pre = new int[n];
    for(int i=0;i<n;i++){
        pre[i] = sc.nextInt();
        if(i!=0) {
            pre[i] = pre[i] + pre[i-1];
        }
    }

    // 双端单调队列维护下标值
    Deque<Integer> queue = new LinkedList<>();
    int res = Integer.MIN_VALUE;
    // 滑动窗口 里面维护的一个最小的 即队首到队尾是递增的 队首是最小的
    for(int i=0;i<n;i++){
        while(!queue.isEmpty()&&pre[i]<pre[queue.getLast()]){
            // 队尾排出来
            queue.pollLast();
        }
        // 入队列
        queue.offerLast(i);
        // 控制最大值在滑窗范围内不在就排出来
        while(queue.peekFirst()+m<=i){
            queue.pollFirst();
        }

        // 不需要非得滑窗内记录值
        res = Math.max(res,pre[i]-pre[queue.peekFirst()]);
    }
        // 输出结果
        System.out.println(res);

    }
}
```