**难度: 简单**

给定一个数组 nums 和滑动窗口的大小 k，请找出所有滑动窗口里的最大值。

示例:

输入: nums = [1,3,-1,-3,5,3,6,7], 和 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

提示：

你可以假设 k 总是有效的，在输入数组不为空的情况下，1 ≤ k ≤ 输入数组的大小。

来源：力扣（LeetCode）

链接：https://leetcode-cn.com/problems/hua-dong-chuang-kou-de-zui-da-zhi-lcof

著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。

**思路:**

1). 创建一个链表来模拟队列

2). 创建一个来存放结果的数组

3). 每次队列满了,就把最大值加入到数组,并且重新比较最大值

4). 最后当nums的长度到最后之后,返回数组

|  |
| --- |
| public int[] maxSlidingWindow(int[] nums, int k) {  int len = nums.length; //获取长度  //长度为0直接返回0  if ( len == 0) {  return new int[0];  }  //定义一个链表用来模拟栈  LinkedList<Integer> queue = new LinkedList<Integer>();  //定义结果集容量  int[] res = new int[len - k +1];  //容量的下标  int index = 0;  //把数组的第一个先入栈  queue.addLast(nums[0]);  //先把数组第一个当成最大值  int max = nums[0];  for (int i = 1; i < len; i++) {  //判断是否已经到k窗口大小  if (queue.size() != k) {  //没到,比较最大值  if (max < nums[i]) {  //最大值小于当前的值,就进行替换  max = nums[i];  }  //添加到最后  queue.addLast(nums[i]);  }else {    //最大值加入到结果  res[index++] = max;    //满了,抛出先进的值,并且判断是否是最大值  if (max == queue.removeFirst()) {  //如果是最大值,就把最后一个假设为最大值  max = nums[i];  //循环的次数  int count = 1;  //循环剩余的栈中的最大值  //因为最后一个已经加入了, 就需要把窗口k个 -1次进行判断  while(count <= k - 1) {    //倒着比较,如果小于  if (max < nums[i-count]) {  //就进行替换  max = nums[i-count];  }    count++;  }    //如果最大值小于添加的值  }else if (max < nums[i]) {  //进行替换  max = nums[i];  }  //把值,添加到最后  queue.addLast(nums[i]);  }  }    //加入最后一次的  res[index] = max;    //返回结果  return res;  } |