Leetcode(python) 队列简单题

622设计循环队列

题目：设计你的循环队列实现。 循环队列是一种线性数据结构，其操作表现基于 FIFO（先进先出）原则并且队尾被连接在队首之后以形成一个循环。它也被称为“环形缓冲器”。  
循环队列的一个好处是我们可以利用这个队列之前用过的空间。在一个普通队列里，一旦一个队列满了，我们就不能插入下一个元素，即使在队列前面仍有空间。但是使用循环队列，我们能使用这些空间去存储新的值。  
你的实现应该支持如下操作：

* MyCircularQueue(k): 构造器，设置队列长度为 k 。
* Front: 从队首获取元素。如果队列为空，返回 -1 。
* Rear: 获取队尾元素。如果队列为空，返回 -1 。
* enQueue(value): 向循环队列插入一个元素。如果成功插入则返回真。
* deQueue(): 从循环队列中删除一个元素。如果成功删除则返回真。
* isEmpty(): 检查循环队列是否为空。
* isFull(): 检查循环队列是否已满。

**示例：**

MyCircularQueue circularQueue = new MycircularQueue(3); // 设置长度为3

circularQueue.enQueue(1);  // 返回true

circularQueue.enQueue(2);  // 返回true

circularQueue.enQueue(3);  // 返回true

circularQueue.enQueue(4);  // 返回false,队列已满

circularQueue.Rear();  // 返回3

circularQueue.isFull();  // 返回true

circularQueue.deQueue();  // 返回true

circularQueue.enQueue(4);  // 返回true

circularQueue.Rear();  // 返回4

思路：环形的肯定不好设计，于是我就是直接弄了一个直的，不断的整体往后移，保持最大容纳k个元素。只需要维护好front和rear指针，就能模拟出来一个环状队列。

需要注意的几个点：

1. Front()和Rear()函数要判断是否为空；
2. 在得到元素的时候rear-1，而front不用。

代码：

1. **class** MyCircularQueue(object):
3. **def** \_\_init\_\_(self, k):
4. """
5. Initialize your data structure here. Set the size of the queue to be k.
6. :type k: int
7. """
8. self.queue = []
9. self.size = k
11. **def** enQueue(self, value):
12. """
13. Insert an element into the circular queue. Return true if the operation is successful.
14. :type value: int
15. :rtype: bool
16. """
17. **if** **not** self.isFull():
18. self.queue.append(value)
19. **return** True
20. **else**:
21. **return** False
23. **def** deQueue(self):
24. """
25. Delete an element from the circular queue. Return true if the operation is successful.
26. :rtype: bool
27. """
28. **if** **not** self.isEmpty():
29. self.queue.pop(0)
30. **return** True
31. **else**:
32. **return** False
34. **def** Front(self):
35. """
36. Get the front item from the queue.
37. :rtype: int
38. """
39. **if** self.isEmpty():
40. **return** -1
41. **else**:
42. **return** self.queue[0]
44. **def** Rear(self):
45. """
46. Get the last item from the queue.
47. :rtype: int
48. """
49. **if** self.isEmpty():
50. **return** -1
51. **else**:
52. **return** self.queue[-1]
54. **def** isEmpty(self):
55. """
56. Checks whether the circular queue is empty or not.
57. :rtype: bool
58. """
59. **return** 0 == len(self.queue)
61. **def** isFull(self):
62. """
63. Checks whether the circular queue is full or not.
64. :rtype: bool
65. """
66. **return** len(self.queue) == self.size



71. # Your MyCircularQueue object will be instantiated and called as such:
72. # obj = MyCircularQueue(k)
73. # param\_1 = obj.enQueue(value)
74. # param\_2 = obj.deQueue()
75. # param\_3 = obj.Front()
76. # param\_4 = obj.Rear()
77. # param\_5 = obj.isEmpty()
78. # param\_6 = obj.isFull()

933最近的请求次数

题目：写一个 RecentCounter 类来计算最近的请求。

它只有一个方法：ping(int t)，其中 t 代表以毫秒为单位的某个时间。

返回从 3000 毫秒前到现在的 ping 数。

任何处于 [t - 3000, t] 时间范围之内的 ping 都将会被计算在内，包括当前（指 t 时刻）的 ping。

保证每次对 ping 的调用都使用比之前更大的 t 值。

**示例：**

**输入：**inputs = ["RecentCounter","ping","ping","ping","ping"], inputs = [[],[1],[100],[3001],[3002]]

**输出：**[null,1,2,3,3]

**提示：**

1. 每个测试用例最多调用 10000 次 ping。
2. 每个测试用例会使用严格递增的 t 值来调用 ping。
3. 每次调用 ping 都有 1 <= t <= 10^9。

思路一：题目大意是找出最近的3000毫秒内有多少个调用请求。每个调用请求是ping(t)函数，其中t是请求的时间，可以保证每次ping的参数t是大于前面的。使用一个队列，当t时间到达之后，在t-3000之前的调用全部删除，因为这些不会对后面的产生任何影响了。删除之后，求长度就好了

代码：

1. **class** RecentCounter(object):
3. **def** \_\_init\_\_(self):
4. self.que=collections.deque()
6. **def** ping(self, t):
7. """
8. :type t: int
9. :rtype: int
10. """
11. **while** self.que **and** self.que[0]<t-3000:
12. self.que.popleft()
13. self.que.append(t)
14. **return** len(self.que)


18. # Your RecentCounter object will be instantiated and called as such:
19. # obj = RecentCounter()
20. # param\_1 = obj.ping(t)