641.设计循环双端队列

设计循环双端队列

641. 设计循环双端队列

难度 中等 82 收藏 分享 切换为英文 接收动态 反馈

设计实现双端队列。
你的实现需要支持以下操作：

- MyCircularDeque(k)：构造函数,双端队列的大小为k。
- insertFront()：将一个元素添加到双端队列头部。 如果操作成功返回 true。
- insertLast()：将一个元素添加到双端队列尾部。如果操作成功返回 true。
- deleteFront()：从双端队列头部删除一个元素。 如果操作成功返回 true。
- deleteLast()：从双端队列尾部删除一个元素。如果操作成功返回 true。
- getFront()：从双端队列头部获得一个元素。如果双端队列为空，返回 -1。
- getRear()：获得双端队列的最后一个元素。 如果双端队列为空，返回 -1。
- isEmpty()：检查双端队列是否为空。
- isFull()：检查双端队列是否满了。

示例：

```
MyCircularDeque circularDeque = new MycircularDeque(3); // 设置容量大小为3
circularDeque.insertLast(1);                            // 返回 true
circularDeque.insertLast(2);                            // 返回 true
circularDeque.insertFront(3);                           // 返回 true
circularDeque.insertFront(4);                           // 已经满了，返回 false
circularDeque.getRear();                                // 返回 2
circularDeque.isFull();                                 // 返回 true
circularDeque.deleteLast();                             // 返回 true
circularDeque.insertFront(4);                           // 返回 true
circularDeque.getFront();                               // 返回 4
```

提示：

- 所有值的范围为 [1, 1000]
- 操作次数的范围为 [1, 1000]
- 请不要使用内置的双端队列库。

641. Design Circular Deque

难度 中等 82 收藏 分享 切换为中文 接收动态 反馈

Design your implementation of the circular double-ended queue (deque).

Implement the `MyCircularDeque` class:

- `MyCircularDeque(int k)` Initializes the deque with a maximum size of `k`.
- `boolean insertFront()` Adds an item at the front of Deque. Returns `true` if the operation is successful, or `false` otherwise.
- `boolean insertLast()` Adds an item at the rear of Deque. Returns `true` if the operation is successful, or `false` otherwise.
- `boolean deleteFront()` Deletes an item from the front of Deque. Returns `true` if the operation is successful, or `false` otherwise.
- `boolean deleteLast()` Deletes an item from the rear of Deque. Returns `true` if the operation is successful, or `false` otherwise.
- `int getFront()` Returns the front item from the Deque. Returns `-1` if the deque is empty.
- `int getRear()` Returns the last item from Deque. Returns `-1` if the deque is empty.
- `boolean isEmpty()` Returns `true` if the deque is empty, or `false` otherwise.
- `boolean isFull()` Returns `true` if the deque is full, or `false` otherwise.

Example 1:

```
Input
["MyCircularDeque", "insertLast", "insertLast", "insertFront",
"insertFront", "getRear", "isFull", "deleteLast", "insertFront", "getFront"]
[[3], [1], [2], [3], [4], [], [], [], [4], []]
Output
[null, true, true, true, false, 2, true, true, true, 4]

Explanation
MyCircularDeque myCircularDeque = new MyCircularDeque(3);
myCircularDeque.insertLast(1);  // return True
myCircularDeque.insertLast(2);  // return True
myCircularDeque.insertFront(3); // return True
myCircularDeque.insertFront(4); // return False, the queue is full.
myCircularDeque.getRear();      // return 2
myCircularDeque.isFull();       // return True
myCircularDeque.deleteLast();   // return True
myCircularDeque.insertFront(4); // return True
myCircularDeque.getFront();     // return 4
```

Constraints:

- `1 <= k <= 1000`
- `0 <= value <= 1000`
- At most `2000` calls will be made to `insertFront`, `insertLast`, `deleteFront`, `deleteLast`, `getFront`, `getRear`, `isEmpty`, `isFull`.

思路 1 ： 数组

这道题的前导问题是「力扣」第 622 题：设计循环队列。
在实现上几乎是一模一样的，要注意的地方有：

1. 定义循环变量 front 和 rear
   - 一直保持这个定义，到底是先赋值还是先移动指针就很容易想清楚了。
   - front：指向队列头部第 11 个有效数据的位置；
   - rear：指向队列尾部（即最后 11 个有效数据）的 下一个位置，即下一个从队尾入队元素的位置。
   - 说明：这个定义是依据「动态数组」的定义模仿而来。

2. 为了避免「队列为空」和「队列为满」的判别条件冲突，我们有意浪费了一个位置
      - 浪费一个位置是指：循环数组中任何时刻一定至少有一个位置不存放有效元素。
      - 判别队列为空的条件是：front == rear; ；
      - 判别队列为满的条件是：(rear + 1) % capacity == front;。可以这样理解，当 rear 循环到数组的前面，要从后面追上 front，还差一格的时候，判定队列为满。
3. 因为有循环的出现，要特别注意处理数组下标可能越界的情况。
   - 指针后移的时候，下标 + 1+1，要取模；
   - 指针前移的时候，为了循环到数组的末尾，需要先加上数组的长度，然后再对数组长度取模。

代码

```
// Java
// Time : 2021 - 07 - 20

lass MyCircularDeque {

    // 1. 不用设计成动态数组，使用静态数组即可
    // 2. 设计 head 和 tail 指针变量
    // 3. head == tail 成立的时候表示队列为空
    // 4. tail + 1 == head

    private int capacity;
    private int[] arr;
    private int front;
    private int rear;
```

```
    /** Initialize your data structure here. Set the size of the deque to be k.
*/
    public MyCircularDeque(int k) {
        capacity = k + 1;
        arr = new int[capacity];

        // 头部指向第1个存放元素的位置
        // 插入时，先减、再赋值
        // 删除时，索引 +1 （注意取模）
        front = 0;

        // 尾部指向下一个插入元素的位置
        // 插入时，先赋值，再加
        // 删除时，索引 -1（注意取模）
        rear = 0;
    }

    /**
     * Adds an item at the front of Deque. Return true if the operation is
     * successful.
     */
    public boolean insertFront(int value) {
        if (isFull())
            return false;

        front = (front - 1 + capacity) % capacity;
        arr[front] = value;
        return true;
    }

    /**
     * Adds an item at the rear of Deque. Return true if the operation is
     * successful.
     */
    public boolean insertLast(int value) {
        if (isFull())
            return false;

        arr[rear] = value;
```

```
        rear = (rear + 1) % capacity;
        return true;
    }

    /**
     * Deletes an item from the front of Deque. Return true if the operation is
     * successful.
     */
    public boolean deleteFront() {
        if (isEmpty())
            return false;

        // front 被设计在数组的开头，所以是 +1
        front = (front + 1) % capacity;
        return true;
    }

    /**
     * Deletes an item from the rear of Deque. Return true if the operation is
     * successful.
     */
    public boolean deleteLast() {
        if (isEmpty())
            return false;

        // rear 被设计在数组的末尾，所以是 -1
        rear = (rear - 1 + capacity) % capacity;
        return true;
    }

    /** Get the front item from the deque. */
    public int getFront() {
        if (isEmpty())
            return -1;
        return arr[front];
    }

    /** Get the last item from the deque. */
    public int getRear() {
```

```
        if (isEmpty())
            return -1;

        // 当 rear 为 0 时防止数组越界
        return arr[(rear - 1 + capacity) % capacity];
    }

    /** Checks whether the circular deque is empty or not. */
    public boolean isEmpty() {
        return front == rear;
    }

    /** Checks whether the circular deque is full or not. */
    public boolean isFull() {
        // 注意这个设计是非常经典的做法
        return (rear + 1) % capacity == front;
    }
}
```

思路 2 ： 链表

思路 3 ： 栈

#Leetcode/Queue #ToDo