

## 行难事必有所得

前中后队列，一定要手写哈！

### 1. LeetCode 86. 分隔链表

```
1 public ListNode partition(ListNode head, int x) {
2     ListNode small = new ListNode(0);
3     ListNode smallHair = small;
4     ListNode large = new ListNode(0);
5     ListNode largeHair = large;
6     while (head != null) {
7         if (head.val < x) {
8             small.next = head;
9             small = small.next;
10        } else {
11            large.next = head;
12            large = large.next;
13        }
14        head = head.next;
15    }
16    large.next = null;
17    small.next = largeHair.next;
18    return smallHair.next;
19 }
```

### 2. LeetCode 138 复制带随机指针的链表

```
1 public Node copyRandomList(Node head) {
2     if (head == null) return null;
3     Node pointer = head;
4     while (pointer != null) {
5         Node newNode = new Node(pointer.val);
6         newNode.next = pointer.next;
7         pointer.next = newNode;
8         pointer = pointer.next;
9     }
10    pointer = head;
11    while (pointer != null) {
12        pointer.next.random = (pointer.random != null) ? pointer.random.next
: null;
13        pointer = pointer.next.next;
14    }
15    Node pointerOldList = head;
16    Node pointerNewList = head.next;
17    Node newHead = head.next;
18    while (pointerOldList != null) {
19        pointerOldList.next = pointerOldList.next.next;
```

```

20         pointerNewList.next = (pointerNewList.next != null) ?
pointerNewList.next.next : null;
21         pointerOldList = pointerOldList.next;
22         pointerNewList = pointerNewList.next;
23     }
24     return newHead;
25 }

```

### 3. LeetCode 622. 设计循环队列

方法一：使用数组

```

1 public class MyCircularQueue {
2     int[] queue;
3     int capacity, front, rear, count;
4
5     public MyCircularQueue(int k) {
6         queue = new int[k];
7         capacity = k;
8         front = 0;
9         rear = 0;
10        count = 0;
11    }
12
13    public boolean enqueue(int value) {
14        if (isFull()) return false;
15        queue[rear++] = value;
16        rear %= capacity;
17        count++;
18        return true;
19    }
20
21    public boolean dequeue() {
22        if (isEmpty()) return false;
23        front = (front + 1) % capacity;
24        count--;
25        return true;
26    }
27
28    public int Front() {
29        if (isEmpty()) return -1;
30        return queue[front];
31    }
32
33    public int Rear() {
34        if (isEmpty()) return -1;
35        return queue[(rear - 1 + capacity) % capacity];
36    }
37
38    public boolean isEmpty() {
39        return count == 0;
40    }
41
42    public boolean isFull() {
43        return count == capacity;

```

```
44     }  
45 }
```

方法二：使用单向链表

```
1  public class MyCircularQueue {  
2      class Node {  
3          int value;  
4          Node nextNode;  
5  
6          public Node(int value) {  
7              this.value = value;  
8          }  
9      }  
10  
11     Node head, tail;  
12     int count, capacity;  
13  
14     public MyCircularQueue(int k) {  
15         capacity = k;  
16     }  
17  
18     public boolean enqueue(int value) {  
19         if (count == capacity) return false;  
20         Node newNode = new Node(value);  
21         if (count == 0) {  
22             head = tail = newNode;  
23         } else {  
24             tail.nextNode = newNode;  
25             tail = newNode;  
26         }  
27         count++;  
28         return true;  
29     }  
30  
31     public boolean dequeue() {  
32         if (count == 0) return false;  
33         head = head.nextNode;  
34         count--;  
35         return true;  
36     }  
37  
38     public int Front() {  
39         if (count == 0) return -1;  
40         return head.value;  
41     }  
42  
43     public int Rear() {  
44         if (count == 0) return -1;  
45         return tail.value;  
46     }  
47  
48     public boolean isEmpty() {  
49         return count == 0;  
50     }  
51  
52     public boolean isFull() {
```

```
53     return count == capacity;
54 }
55 }
```

#### 4. LeetCode 641. 设计循环双端队列

第一种方法

```
1 public class MyCircularDeque {
2     int[] queue;
3     int capacity, front, rear, count;
4
5     /**
6      * Initialize your data structure here. Set the size of the deque to be k.
7      */
8     public MyCircularDeque(int k) {
9         capacity = k;
10        front = 0;
11        rear = 0;
12        count = 0;
13        queue = new int[k];
14    }
15
16    /**
17     * Adds an item at the front of Deque. Return true if the operation is
18     * successful.
19     */
20    public boolean insertFront(int value) {
21        if (isFull()) return false;
22        front = (front - 1 + capacity) % capacity;
23        queue[front] = value;
24        count++;
25        return true;
26    }
27
28    /**
29     * Adds an item at the rear of Deque. Return true if the operation is
30     * successful.
31     */
32    public boolean insertLast(int value) {
33        if (isFull()) return false;
34        queue[rear++] = value;
35        rear %= capacity;
36        count++;
37        return true;
38    }
39
40    /**
41     * Deletes an item from the front of Deque. Return true if the operation is
42     * successful.
43     */
44    public boolean deleteFront() {
45        if (isEmpty()) return false;
46        front = (front + 1) % capacity;
```

```

45         count--;
46         return true;
47     }
48
49     /**
50      * Deletes an item from the rear of Deque. Return true if the operation is
successful.
51      */
52     public boolean deleteLast() {
53         if (isEmpty()) return false;
54         rear = (rear - 1 + capacity) % capacity;
55         count--;
56         return true;
57     }
58
59     /**
60      * Get the front item from the deque.
61      */
62     public int getFront() {
63         if (isEmpty()) return -1;
64         return queue[front];
65     }
66
67     /**
68      * Get the last item from the deque.
69      */
70     public int getRear() {
71         if (isEmpty()) return -1;
72         return queue[(rear - 1 + capacity) % capacity];
73     }
74
75     /**
76      * Checks whether the circular deque is empty or not.
77      */
78     public boolean isEmpty() {
79         return count == 0;
80     }
81
82     /**
83      * Checks whether the circular deque is full or not.
84      */
85     public boolean isFull() {
86         return count == capacity;
87     }
88
89 }

```

## 第二种方法

```

1 class MyCircularDeque {
2     private int capacity;
3     private int[] arr;
4     private int front;
5     private int rear;
6
7     public MyCircularDeque(int k) {
8         capacity=k+1;

```

```
9         arr=new int[capacity+1];
10         front=0;
11         rear=0;
12     }
13
14
15     public boolean insertFront(int value) {
16         if(isFull()){
17             return false;
18         }
19         front=(front-1+capacity)%capacity;
20         arr[front]=value;
21         return true;
22     }
23
24
25     public boolean insertLast(int value) {
26         if(isFull()){
27             return false;
28         }
29
30         arr[rear]=value;
31         rear=(rear+1)%capacity;
32         return true;
33     }
34
35     /** Deletes an item from the front of Deque. Return true if the operation is
successful. */
36     public boolean deleteFront() {
37         if(isEmpty()){
38             return false;
39         }
40         front=(front+1)%capacity;
41         return true;
42     }
43
44     /** Deletes an item from the rear of Deque. Return true if the operation is
successful. */
45     public boolean deleteLast() {
46         if(isEmpty()){
47             return false;
48         }
49         rear=(rear-1+capacity)%capacity;
50         return true;
51     }
52
53     /** Get the front item from the deque. */
54     public int getFront() {
55         if(isEmpty()){
56             return -1;
57         }
58         return arr[front];
59     }
60
61     /** Get the last item from the deque. */
62     public int getRear() {
63         if (isEmpty()) {
64             return -1;
```

```

65     }
66     return arr[(rear - 1 + capacity) % capacity];
67 }
68
69 public boolean isEmpty() {
70     return rear==front;
71 }
72
73 public boolean isFull() {
74     return (rear+1) % capacity==front;
75 }
76 }

```

## 5. LeetCode 1670. 设计前中后队列

第一种方法

```

1 public class FrontMiddleBackQueue {
2     class Node {
3         Node pre, next;
4         int val;
5
6
7         public Node() {
8         }
9
10        public Node(int val) {
11            this.val = val;
12        }
13
14        public void insertPre(Node node) {
15            node.pre = pre;
16            node.next = this;
17            if (this.pre != null) this.pre.next = node;
18            this.pre = node;
19        }
20
21        public void insertNext(Node node) {
22            node.next = this.next;
23            node.pre = this;
24            if (this.next != null) this.next.pre = node;
25            this.next = node;
26        }
27
28        public void deletePre() {
29            if (this.pre == null) return;
30            Node pointer = this.pre;
31            this.pre = pointer.pre;
32            if (pointer.pre != null) pointer.pre.next = this;
33        }
34
35        public void deleteNext() {
36            if (this.next == null) return;
37            Node pointer = this.next;
38            this.next = pointer.next;

```

```
39         if (pointer.next != null) pointer.next.pre = this;
40     }
41 }
42
43 class MyQueue {
44     Node dummyHead = new Node(), dummyTail = new Node();
45     int count;
46
47     public MyQueue() {
48         dummyHead.next = dummyTail;
49         dummyHead.pre = null;
50         dummyTail.pre = dummyHead;
51         dummyTail.next = null;
52         count = 0;
53     }
54
55     public void pushFront(int value) {
56         dummyHead.insertNext(new Node(value));
57         count++;
58     }
59
60     public void pushBack(int value) {
61         dummyTail.insertPre(new Node(value));
62         count++;
63     }
64
65     public int popBack() {
66         if (isEmpty()) return -1;
67         int val = dummyTail.pre.val;
68         dummyTail.deletePre();
69         ;
70         count--;
71         return val;
72     }
73
74     public int popFront() {
75         if (isEmpty()) return -1;
76         int val = dummyHead.next.val;
77         dummyHead.deleteNext();
78         count--;
79         return val;
80     }
81
82     public boolean isEmpty() {
83         return dummyHead.next == dummyTail;
84     }
85
86     public int size() {
87         return count;
88     }
89 }
90
91 MyQueue left = new MyQueue(), right = new MyQueue();
92
93 public FrontMiddleBackQueue() {
94
95 }
96
```



```
97     public void reBalance() {
98         if (left.size() < right.size()) {
99             left.pushBack(right.popFront());
100         }
101         if (left.size() == right.size() + 2) {
102             right.pushFront(left.popBack());
103         }
104     }
105
106     public void pushFront(int val) {
107         left.pushFront(val);
108         reBalance();
109     }
110
111     public void pushMiddle(int val) {
112         if (left.size() > right.size()) {
113             right.pushFront(left.popBack());
114         }
115         left.pushBack(val);
116     }
117
118     public void pushBack(int val) {
119         right.pushBack(val);
120         reBalance();
121     }
122
123     public int popFront() {
124         if (isEmpty()) return -1;
125         int val = left.popFront();
126         reBalance();
127         return val;
128     }
129
130     public int popMiddle() {
131         if (isEmpty()) return -1;
132         int val = left.popBack();
133         reBalance();
134         return val;
135     }
136
137     public int popBack() {
138         if (isEmpty()) return -1;
139         int val;
140         if (right.isEmpty()) {
141             val = left.popBack();
142         } else {
143             val = right.popBack();
144         }
145         reBalance();
146         return val;
147     }
148
149     public boolean isEmpty() {
150         return left.size() == 0;
151     }
152 }
153
```

## 第二种方法

```
1  class FrontMiddleBackQueue {
2
3      Deque<Integer> left;
4      Deque<Integer> right;
5
6      public FrontMiddleBackQueue() {
7          left = new LinkedList<>();
8          right = new LinkedList<>();
9      }
10
11     public void pushFront(int val) {
12         left.addFirst(val);
13         reBalance();
14     }
15
16     public void pushMiddle(int val) {
17         if (left.size() == right.size()) {
18             right.addFirst(val);
19         } else {
20             left.addLast(val);
21         }
22     }
23
24     public void pushBack(int val) {
25         right.addLast(val);
26         reBalance();
27     }
28
29     public int popFront() {
30         Integer integer = left.pollFirst();
31         if (integer == null) {
32             integer = right.pollFirst();
33             return integer == null ? -1 : integer;
34         } else {
35             reBalance();
36             return integer;
37         }
38     }
39
40     public int popMiddle() {
41         if (left.size() == right.size()) {
42             Integer integer = left.pollLast();
43             return integer == null ? -1 : integer;
44         } else {
45             return right.pollFirst();
46         }
47     }
48
49     public int popBack() {
50         Integer integer = right.pollLast();
51         reBalance();
52         return integer == null ? -1 : integer;
53     }
54
55     public void reBalance() {
```

```

56         if (left.size() > right.size()) {
57             right.addFirst(left.pollLast());
58         } else if (right.size() == left.size() + 2) {
59             left.addLast(right.pollFirst());
60         }
61     }
62 }
63

```

## 6. LeetCode 933. 最近的请求次数

```

1 public class RecentCounter {
2     LinkedList<Integer> queue = new LinkedList<Integer>();
3
4     public RecentCounter() {
5     }
6
7     public int ping(int t) {
8         queue.add(t);
9         while (queue.peek() < t - 3000) {
10             queue.poll();
11         }
12         return queue.size();
13     }
14 }

```

## 7. 面试题 17.09. 第 k 个数

```

1 public int getKthMagicNumber(int k) {
2
3     int[] numbers = new int[k];
4     numbers[0] = 1;
5     int p3 = 0, p5 = 0, p7 = 0;
6     for (int i = 1; i < k; i++) {
7         numbers[i] = Math.min(Math.min(3 * numbers[p3], 5 * numbers[p5]), 7
8 * numbers[p7]);
9         if (numbers[i] == 3 * numbers[p3]) p3++;
10        if (numbers[i] == 5 * numbers[p5]) p5++;
11        if (numbers[i] == 7 * numbers[p7]) p7++;
12    }
13    return numbers[k - 1];
14 }

```

## 8. LeetCode 859. 亲密字符串

```

1 public boolean specialJudge(String a, String b) {
2     int[] count = new int[26];
3     for (int i = 0; i < a.length(); i++) {
4         count[a.charAt(i) - 'a']++;
5         if (count[a.charAt(i) - 'a'] > 1) return true;
6     }
7     return false;
8 }

```

```

9
10 public boolean buddyStrings(String a, String b) {
11     if (a.length() != b.length()) return false;
12     if (a.equals(b)) return specilJudge(a, b);
13     int first = -1, second = -1;
14     for (int i = 0; i < a.length(); i++) {
15         if (a.charAt(i) != b.charAt(i)) {
16             if (first == -1) first = i;
17             else if (second == -1) second = i;
18             else return false;
19         }
20     }
21     return second != -1 && a.charAt(first) == b.charAt(second) &&
a.charAt(second) == b.charAt(first);
22 }

```

## 9.LeetCode 860. 柠檬水找零

```

1 public boolean lemonadeChange(int[] bills) {
2     int five = 0, ten = 0;
3     for (int bill : bills) {
4         if (bill == 5) {
5             five++;
6         } else if (bill == 10) {
7             if (five == 0) return false;
8             five--;
9             ten++;
10        } else {
11            if (five > 0 && ten > 0) {
12                five--;
13                ten--;
14            } else if (five >= 3) {
15                five -= 3;
16            } else {
17                return false;
18            }
19        }
20    }
21    return true;
22 }

```

## 10.LeetCode 969. 煎饼排序

```

1 public List<Integer> pancakeSort(int[] arr) {
2     ArrayList<Integer> result = new ArrayList<>();
3     for (int i = arr.length; i > 0; i--) {
4         int index = 0;
5         while (arr[index] != i) {
6             index++;
7         }
8         if (index == i - 1) {
9             continue;
10        } else if (index == 0) {
11            reverse(arr, i);

```

```

12         result.add(i);
13     } else {
14         reverse(arr, index + 1);
15         result.add(index + 1);
16         reverse(arr, i);
17         result.add(i);
18     }
19 }
20 return result;
21 }
22
23 public void reverse(int[] arr, int index) {
24     int start = 0, end = index - 1;
25     while (start < end) {
26         int temp = arr[end];
27         arr[end] = arr[start];
28         arr[start] = temp;
29         start++;
30         end--;
31     }
32 }

```

## 11. LeetCode 621. 任务调度器

```

1 public int leastInterval(char[] tasks, int n) {
2     int[] count = new int[26];
3     for (int i = 0; i < tasks.length; i++) {
4         count[tasks[i] - 'A']++;
5     }
6     Arrays.sort(count);
7     int maxTimes = count[25];
8     int maxCount = 1;
9     // for (int i = 24; i > 0; i--) {
10    //     if (maxTimes == count[i]) {
11    //         maxCount++;
12    //     } else break;
13    // }
14    // for (int i = 25; i >= 1; i--) {
15    //     if (count[i] == count[i - 1]) {
16    //         maxCount++;
17    //     } else {
18    //         break;
19    //     }
20    // }
21    for (int i = 25; i >= 1 && count[i] == count[i - 1]; i--, maxCount++) ;
22    return Math.max((maxTimes - 1) * (n + 1) + maxCount, tasks.length);
23 }

```

