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

```
    public ListNode partition(ListNode head, int x) {
        ListNode small = new ListNode(0);
        ListNode smallHair = small;
        ListNode large = new ListNode(0);
        ListNode largeHair = large;
        while (head != null) {
            if (head.val < x) {
                small.next = head;
                small = small.next;
            } else {
                large.next = head;
                large = large.next;
            }
            head = head.next;
        }
        large.next = null;
        small.next = largeHair.next;
        return smallHair.next;
    }
```

2.LeetCode 138 复制带随机指针的链表

```
    public Node copyRandomList(Node head) {
        if (head == null) return null;
        Node pointer = head;
        while (pointer != null) {
            Node newNode = new Node(pointer.val);
            newNode.next = pointer.next;
            pointer.next = newNode;
            pointer = newNode.next;
        }
        pointer = head;
        while (pointer != null) {
            pointer.next.random = (pointer.random != null) ? pointer.random.next
: null;
            pointer = pointer.next.next;
        }
        Node pointerOldList = head;
        Node pointerNewList = head.next;
        Node newHead = head.next;
        while (pointerOldList != null) {
            pointerOldList.next = pointerOldList.next.next;
```

```
            pointerNewList.next = (pointerNewList.next != null) ?
pointerNewList.next.next : null;
            pointerOldList = pointerOldList.next;
            pointerNewList = pointerNewList.next;
        }
        return newHead;
    }
```

3.LeetCode 622. 设计循环队列

方法一：使用数组

```
public class MyCircularQueue {
    int[] queue;
    int capacity, front, rear, count;

    public MyCircularQueue(int k) {
        queue = new int[k];
        capacity = k;
        front = 0;
        rear = 0;
        count = 0;
    }

    public boolean enQueue(int value) {
        if (isFull()) return false;
        queue[rear++] = value;
        rear %= capacity;
        count++;
        return true;
    }

    public boolean deQueue() {
        if (isEmpty()) return false;
        front = (front + 1) % capacity;
        count--;
        return true;
    }

    public int Front() {
        if (isEmpty()) return -1;
        return queue[front];
    }

    public int Rear() {
        if (isEmpty()) return -1;
        return queue[(rear - 1 + capacity) % capacity];
    }

    public boolean isEmpty() {
        return count == 0;
    }

    public boolean isFull() {
        return count == capacity;
```

```
    }
}
```

方法二：使用单向链表

```
public class MyCircularQueue {
    class Node {
        int value;
        Node nextNode;

        public Node(int value) {
            this.value = value;
        }
    }

    Node head, tail;
    int count, capacity;

    public MyCircularQueue(int k) {
        capacity = k;
    }

    public boolean enQueue(int value) {
        if (count == capacity) return false;
        Node newNode = new Node(value);
        if (count == 0) {
            head = tail = newNode;
        } else {
            tail.nextNode = newNode;
            tail = newNode;
        }
        count++;
        return true;
    }

    public boolean deQueue() {
        if (count == 0) return false;
        head = head.nextNode;
        count--;
        return true;
    }

    public int Front() {
        if (count == 0) return -1;
        return head.value;
    }

    public int Rear() {
        if (count == 0) return -1;
        return tail.value;
    }

    public boolean isEmpty() {
        return count == 0;
    }

    public boolean isFull() {
```

```
        return count == capacity;
    }
}
```

4.LeetCode 641. 设计循环双端队列

第一种方法

```
public class MyCircularDeque {
    int[] queue;
    int capacity, front, rear, count;

    /**
     * Initialize your data structure here. Set the size of the deque to be k.
     */
    public MyCircularDeque(int k) {
        capacity = k;
        front = 0;
        rear = 0;
        count = 0;
        queue = new int[k];
    }

    /**
     * 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;
        queue[front] = value;
        count++;
        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;
        queue[rear++] = value;
        rear %= capacity;
        count++;
        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 = (front + 1) % capacity;
```

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

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

    /**
     * Get the last item from the deque.
     */
    public int getRear() {
        if (isEmpty()) return -1;
        return queue[(rear - 1 + capacity) % capacity];

    }

    /**
     * Checks whether the circular deque is empty or not.
     */
    public boolean isEmpty() {
        return count == 0;
    }

    /**
     * Checks whether the circular deque is full or not.
     */
    public boolean isFull() {
        return count == capacity;
    }
}
```

第二种方法

```
class MyCircularDeque {
    private int capacity;
    private int[] arr;
    private int front;
    private int rear;

    public MyCircularDeque(int k) {
        capacity=k+1;
```

```
        arr=new int[capacity+1];
        front=0;
        rear=0;
    }


    public boolean insertFront(int value) {
        if(isFull()){
            return false;
        }
        front=(front-1+capacity)%capacity;
        arr[front]=value;
        return true;
    }


    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=(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=(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;
```

```
        }
        return arr[(rear - 1 + capacity) % capacity];
    }

    public boolean isEmpty() {
        return rear==front;
    }

    public boolean isFull() {
        return (rear+1) % capacity==front;
    }
}
```

5.LeetCode 1670. 设计前中后队列

第一种方法

```
public class FrontMiddleBackQueue {
    class Node {
        Node pre, next;
        int val;


        public Node() {
        }

        public Node(int val) {
            this.val = val;
        }

        public void insertPre(Node node) {
            node.pre = pre;
            node.next = this;
            if (this.pre != null) this.pre.next = node;
            this.pre = node;
        }

        public void insertNext(Node node) {
            node.next = this.next;
            node.pre = this;
            if (this.next != null) this.next.pre = node;
            this.next = node;
        }

        public void deletePre() {
            if (this.pre == null) return;
            Node pointer = this.pre;
            this.pre = pointer.pre;
            if (pointer.pre != null) pointer.pre.next = this;
        }

        public void deleteNext() {
            if (this.next == null) return;
            Node pointer = this.next;
            this.next = pointer.next;
```

```
            if (pointer.next != null) pointer.next.pre = this;
        }
    }

    class MyQueue {
        Node dummyHead = new Node(), dummyTail = new Node();
        int count;

        public MyQueue() {
            dummyHead.next = dummyTail;
            dummyHead.pre = null;
            dummyTail.pre = dummyHead;
            dummyTail.next = null;
            count = 0;
        }

        public void pushFront(int value) {
            dummyHead.insertNext(new Node(value));
            count++;
        }

        public void pushBack(int value) {
            dummyTail.insertPre(new Node(value));
            count++;
        }

        public int popBack() {
            if (isEmpty()) return -1;
            int val = dummyTail.pre.val;
            dummyTail.deletePre();
            ;
            count--;
            return val;
        }

        public int popFront() {
            if (isEmpty()) return -1;
            int val = dummyHead.next.val;
            dummyHead.deleteNext();
            count--;
            return val;
        }

        public boolean isEmpty() {
            return dummyHead.next == dummyTail;
        }

        public int size() {
            return count;
        }
    }

    MyQueue left = new MyQueue(), right = new MyQueue();

    public FrontMiddleBackQueue() {

    }

```

```
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

```java
class FrontMiddleBackQueue {

        Deque<Integer> left;
        Deque<Integer> right;

        public FrontMiddleBackQueue() {
            left = new LinkedList<>();
            right = new LinkedList<>();
        }

        public void pushFront(int val) {
            left.addFirst(val);
            reBalance();
        }

        public void pushMiddle(int val) {
            if (left.size() == right.size()) {
                right.addFirst(val);
            } else {
                left.addLast(val);
            }
        }

        public void pushBack(int val) {
            right.addLast(val);
            reBalance();
        }

        public int popFront() {
            Integer integer = left.pollFirst();
            if (integer == null) {
                integer = right.pollFirst();
                return integer == null ? -1 : integer;
            } else {
                reBalance();
                return integer;
            }
        }

        public int popMiddle() {
            if (left.size() == right.size()) {
                Integer integer = left.pollLast();
                return integer == null ? -1 : integer;
            } else {
                return right.pollFirst();
            }
        }

        public int popBack() {
            Integer integer = right.pollLast();
            reBalance();
            return integer == null ? -1 : integer;
        }

        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
```