1.面试题 02.02. 返回倒数第 k 个节点

```java
public int kthToLast(ListNode head, int k) {
    ListNode first = head, second = head;
    for (; k > 0; first = first.next, k--) ;
    for (; first != null; first = first.next, second = second.next) ;
    return second.val;
}
```

2.剑指 Offer 22. 链表中倒数第k个节点

```java
    public int kthToLast(ListNode head, int k) {
        ListNode first = head, second = head;
        for (; k > 0; first = first.next, k--) ;
        for (; first != null; first = first.next, second = second.next) ;
        return second.val;
    }
```

3.剑指 Offer 35. 复杂链表的复制

```java
    public Node copyRandomList(Node head) {
        if (head == null) return null;
        // 定义一个指针，来处理新旧节点的混合链表
        Node pointer = head;
        while (pointer != null) {
            // 复制前一个节点的值
            Node newNode = new Node(pointer.val);
            // 将新节点插入原节点的后面
            newNode.next = pointer.next;
            pointer.next = newNode;
            // pointer指针向后挪
            pointer = newNode.next;
        }
        // 执行到这里 的时候说明混合链表已经完成，现在开始处理随机指针
        // 指针重新指向head
        pointer = head;
        // 开始处理新建节点的随机指针
        // 遍历混合链表，将原节点的随机指针，赋值给新节点的随机指针
        while (pointer != null) {
            //如果原节点的随机指针不为null，新节点的随机指针，指向原节点的随机节点的下一个节点
            pointer.next.random = (pointer.random != null) ? pointer.random.next : null;
            // pointer向后走两步，因为要跨过新建的节点
            pointer = pointer.next.next;
        }
        // 拆解混合链表
        // 比如 A->A'->B->B'->C->C' 将被拆分为 A->B->C 和 A'->B'->C'
        // 定义两个指针分别接收原链表和新链表
        Node pointerOldList = head; // A->B->C
        Node pointerNewList = head.next; // A'->B'->C'
        // 标记新链表的头节点
        Node headNew = head.next;
        while (pointerOldList != null) {
            // 改变指针的下一个节点，之后指针向后移动
            pointerOldList.next = pointerOldList.next.next;
            pointerNewList.next = (pointerNewList.next != null) ? pointerNewList.next.next : null;
            pointerOldList = pointerOldList.next;
            pointerNewList = pointerNewList.next;
        }
        return headNew;
    }
```

4.面试题 02.03. 删除中间节点

```
    public void deleteNode(ListNode node) {
        node.val=node.next.val;
        node.next=node.next.next;
    }
```

5.445. 两数相加 II

```
public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
    Deque<Integer> stack1 = new LinkedList<Integer>();
    Deque<Integer> stack2 = new LinkedList<Integer>();
    while (l1 != null) {
        stack1.push(l1.val);
        l1 = l1.next;
    }
    while (l2 != null) {
        stack2.push(l2.val);
        l2 = l2.next;
    }
    // 进位
    int temp = 0;
    // 新节点
    ListNode result = null;
    //两个栈不为空，或者进位不为空
    while (!stack1.isEmpty() || !stack2.isEmpty() || temp != 0) {
        int a = stack1.isEmpty() ? 0 : stack1.pop();
        int b = stack2.isEmpty() ? 0 : stack2.pop();
        int cur = a + b + temp;
        temp = cur / 10;
        cur %= 10;
        ListNode curnode = new ListNode(cur);
        curnode.next = result;
        result = curnode;
    }
    return result;
}
```

6.143. 重排链表

```java
    public void reorderList(ListNode head) {
        if (head == null) {
            return;
        }
        List<ListNode> list = new ArrayList<ListNode>();
        ListNode node = head;
        while (node != null) {
            list.add(node);
            node = node.next;
        }
        int i = 0, j = list.size() - 1;
        while (i < j) {
            list.get(i).next = list.get(j);
            i++;
            if (i == j) {
                break;
            }
            list.get(j).next = list.get(i);
            j--;
        }
        list.get(i).next = null;
    }

```

7.面试题 02.08. 环路检测

```java
public ListNode detectCycle(ListNode head) {
    if (head == null) return null;
    // 快慢指针同时指向head节点
    ListNode fast = head, slow = head;
    do {
        // 如果快指针遍历到头了，说明没环
        if (fast == null || fast.next == null) {
            return null;
        }
        fast = fast.next.next;
        slow = slow.next;
    } while (fast != slow);
    //能执行到这里说明快慢指针相遇了，让快指针从头开始遍历
    fast = head;
    //让快慢指针向后走，直到相遇
    while (fast != slow) {
        fast = fast.next;
        slow = slow.next;
    }
    return fast;
}
```

8.707. 设计链表

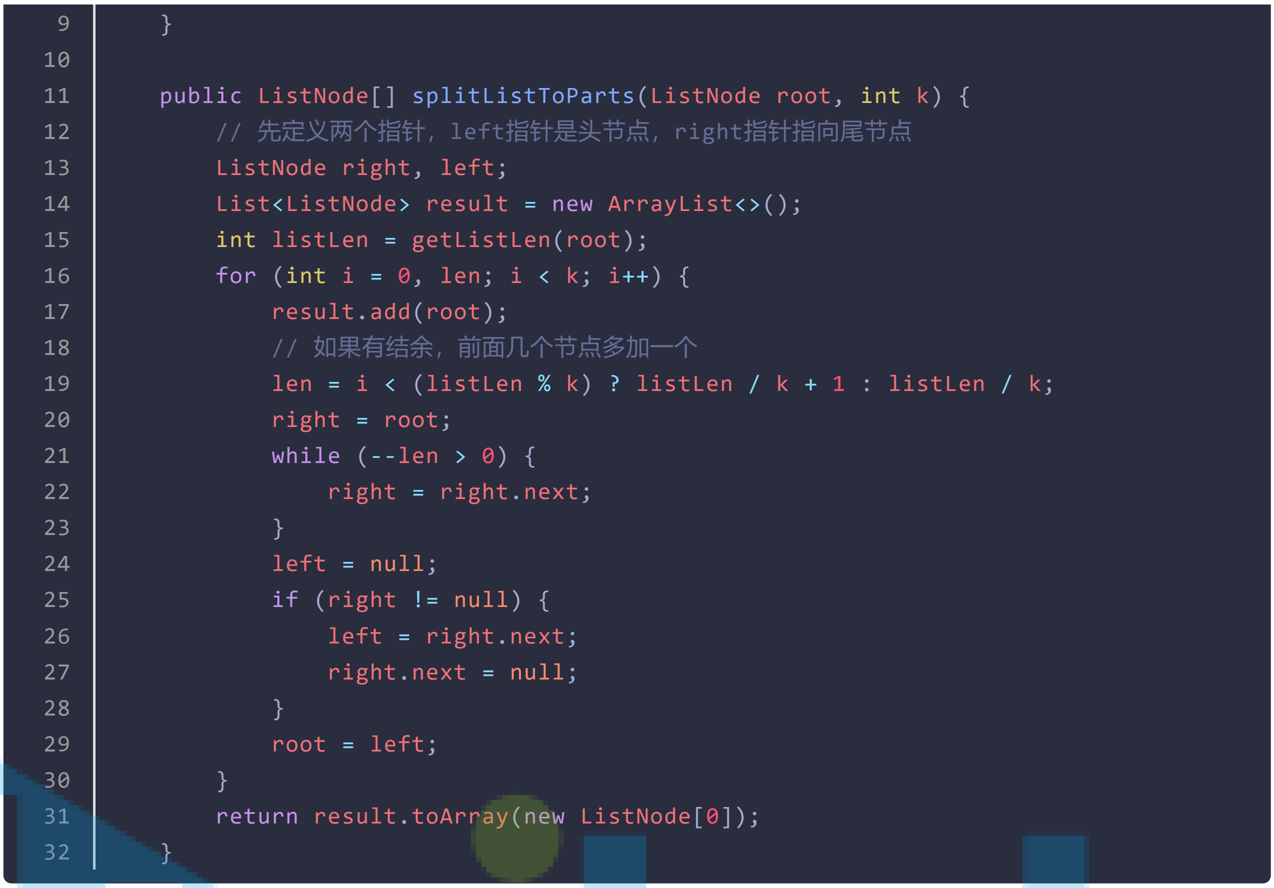
```java
public class MyLinkedList {
    class Node {
        Node prev, next;
        int val;
        public Node() {
        }

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

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

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

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

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

    Node dummyHead = new Node(-1), dummyTail = new Node(-1);
    int count;

    public MyLinkedList() {
        dummyHead.next = dummyTail;
        dummyTail.prev = dummyHead;

    }

    public int get(int index) {
        if (index < 0 || index >= count) return -1;
        Node pointer = dummyHead.next;
        while (index-- > 0) pointer = pointer.next;
        return pointer.val;
    }


    public void addAtHead(int val) {
        dummyHead.insertNext(new Node(val));
        count++;
    }


    public void addAtTail(int val) {
        dummyTail.insertPre(new Node(val));
        count++;
    }


    public void addAtIndex(int index, int val) {
        if (index > count) return;
        Node pointer = dummyHead;
        while (index-- > 0) pointer = pointer.next;
        pointer.insertNext(new Node(val));
        count++;
    }


    public void deleteAtIndex(int index) {
        if (index < 0 || index >= count) return;
        Node pointer = dummyHead;
        while (index-- > 0) pointer = pointer.next;
        pointer.deleteNext();
        count--;

    }
}
```

9.剑指 Offer 18. 删除链表的节点

```java
    public ListNode deleteNode(ListNode head, int val) {
        ListNode hair = new ListNode(0), pointer = hair;
        hair.next = head;
        while (pointer.next != null) {
            if (pointer.next.val != val) pointer = pointer.next;
            else pointer.next = pointer.next.next;
        }
        return hair.next;
    }
```

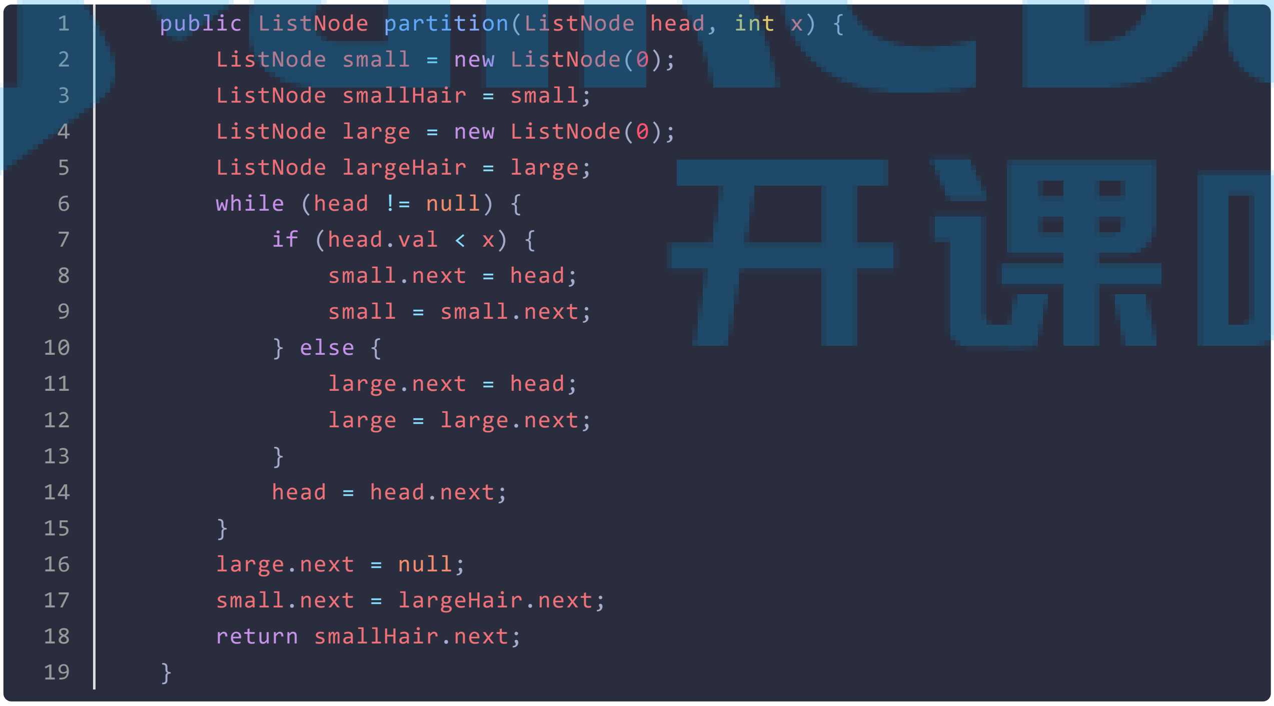
10.725. 分隔链表

```java
    // 获取链表长度
    public int getListLen(ListNode root) {
        int len = 0;
        while (root != null) {
            ++len;
            root = root.next;
        }
        return len;
```



```java
    }

    public ListNode[] splitListToParts(ListNode root, int k) {
        // 先定义两个指针，left指针是头节点，right指针指向尾节点
        ListNode right, left;
        List<ListNode> result = new ArrayList<>();
        int listLen = getListLen(root);
        for (int i = 0, len; i < k; i++) {
            result.add(root);
            // 如果有结余，前面几个节点多加一个
            len = i < (listLen % k) ? listLen / k + 1 : listLen / k;
            right = root;
            while (--len > 0) {
                right = right.next;
            }
            left = null;
            if (right != null) {
                left = right.next;
                right.next = null;
            }
            root = left;
        }
        return result.toArray(new ListNode[0]);
    }
```

11.面试题 02.04. 分割链表



```java
    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;
    }
```

12.779. 第K个语法符号

K在奇数位时，与N-1, (K+1)/2 位置的值相同 K在偶数位时，与N-1, K/2 位置的值相反

```java
public int kthGrammar(int N, int K) {
    if (N == 0) return 0;
    if (K % 2 == 1) return kthGrammar(N - 1, (K + 1) / 2);
    else return Math.abs(kthGrammar(N - 1, K / 2) - 1);
}
```

13.剑指 Offer 10- I. 斐波那契数列

迭代

```java
public int fib(int n) {
    int a = 0, b = 1, sum;
    for (int i = 0; i < n; i++) {
        sum = (a + b) % 1000000007;
        a = b;
        b = sum;
    }
    return a;
}
```

加入map的递归

```java
HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
public  int fib(int n) {
    if (n < 2) return n;
    if (map.containsKey(n)) return map.get(n);
    int ret = (fib(n - 1) + fib(n - 2)) % 1000000007;
    map.put(n, ret);//把计算过的放在map中，若遇到计算过的就直接取
    return ret;
}

```

矩阵

```java
public int fib(int n) {
    long[] initialState = new long[]{1, 0, 0, 0};
    long[] transformation = new long[]{1, 1, 1, 0};
    long[] buffer = new long[]{0, 0, 0, 0};
    matrixPowxer(transformation, n, buffer);
    long[] finalState = new long[]{0, 0, 0, 0};
    matrixMultiply(buffer, initialState, finalState);
    return (int) finalState[2];
}

private void matrixMultiply(long[] left, long[] right, long[] result) {
    result[0] = (left[0] * right[0] + left[1] * right[2]) % 1000000007;
    result[1] = (left[0] * right[1] + left[1] * right[3]) % 1000000007;
    result[2] = (left[2] * right[0] + left[3] * right[2]) % 1000000007;
    result[3] = (left[2] * right[1] + left[3] * right[3]) % 1000000007;
}

private void matrixPowxer(long[] base, long exponent, long[] result) {
    matrixCopy(result, new long[]{1, 0, 0, 1});
    long[] currentBase = new long[]{0, 0, 0, 0};
    matrixCopy(currentBase, base);
```

```
        long[] buffer = new long[]{0, 0, 0, 0};
        while (exponent != 0) {
            if (exponent % 2 != 0) {
                matrixMultiply(currentBase, result, buffer);
                matrixCopy(result, buffer);
            }
            matrixMultiply(base, base, currentBase);
            matrixCopy(base, currentBase);
            exponent /= 2;
        }
    }

    private void matrixCopy(long[] destination, long[] source) {
        for (int i = 0; i < 4; i++) {
            destination[i] = source[i];
        }
    }
```

最后一题的矩阵解法 考虑到大家应该线性代数可能都忘了，所以给大家找了几个视频资料 https://www.bilibili.com/video/BV1ys411472E 感兴趣的同学先看一下这个的00~04 https://www.bilibili.com/video/BV1at411d79w 以及这个的01