1.循环链表

1. 哈希表法

```
    public boolean hasCycle(ListNode head) {
        HashSet<ListNode> hashSet = new HashSet<ListNode>();
        while (head != null) {
            // hashSet.add(head) 如果里面没有该节点返回true, 并且将head节点添加到
hashset里边
            // 如果里面有该节点返回false
            if (!hashSet.add(head)) return true;
            //指针指向
            head = head.next;
        }
        return false;
    }
```

2.快慢指针法

```
    public boolean hasCycle(ListNode head) {
        if (head == null) return false;
        ListNode fast = head, slow = head;
        do {
            if (fast == null || fast.next == null) {
                return false;
            }
            slow = slow.next;
            fast = fast.next.next;
        } while (fast != slow);
        return true;
    }
```

2.循环链表||

1.哈希表

```
    public ListNode detectCycle(ListNode head) {
        HashSet<ListNode> hashSet = new HashSet<ListNode>();
        while (head != null) {
            if (!hashSet.add(head)) {
                return head;
            }
            head = head.next;
        }
        return null;
    }
```

2.快慢指针

```
    public ListNode detectCycle(ListNode head) {
      if (head==null )return null ;
```

```
        ListNode fast= head,slow=head;
        do {
            if (fast==null || fast.next==null) return null;
            fast=fast.next.next;
            slow=slow.next;
        }while (fast!=slow);
        // ListNode newNode=head;
          fast=head;
          while (fast!=slow){
              slow=slow.next;
              fast=fast.next;
          }
          return fast;
    }
```

3.快乐数

```
    public boolean isHappy(int n) {
        int fast = n, slow = n;
        do {
            fast = getNext(getNext(fast));
            slow = getNext(slow);
        } while (fast != slow && fast != 1);
        return fast == 1;
    }

    public int getNext(int n) {
        int sum = 0;
        while (n > 0) {
            // 15  5* 5
            sum += (n % 10) * (n % 10);
            n = n / 10;
        }
        return sum;
    }
```

4.反转链表

```
    public ListNode reverseList(ListNode head) {
        ListNode pre = null, curr = head, next = null;
        while (curr != null) {
            next = curr.next;
            curr.next = pre;
            pre = curr;
            curr = next;
        }
        return pre;
    }
```

5.反转链表II

```
public ListNode reverseBetween(ListNode head, int left, int right) {
        ListNode hair = new ListNode(0, head), con = hair, tail = null;
```

```
        int n = right - left + 1;
        while (left > 1) {
            con = con.next;
            left--;
        }
        con.next = reverse(con.next, n);
        return hair.next;
    }

    public ListNode reverse(ListNode head, int n) {
        ListNode pre = new ListNode(), curr = head, next = null;
        while (n > 0) {
            next = curr.next;
            curr.next = pre.next;
            pre.next = curr;
            curr = next;
            n--;
        }
        head.next = curr;
        return pre.next;
    }
```

6.K个一组反转链表

```
public ListNode reverseKGroup(ListNode head, int k) {
        ListNode hair = new ListNode(0, head), pre = hair, tail = null;
        while (head != null) {
            tail = pre;
            for (int i = 0; i < k; i++) {
                tail = tail.next;
                if (tail == null) {
                    return hair.next;
                }
            }
            ListNode[] reverse = reverse(head, tail);
            head = reverse[0];
            tail = reverse[1];
            pre.next = head;
            pre = tail;
            head = pre.next;
        }
        return hair.next;
    }

    public ListNode[] reverse(ListNode head, ListNode tail) {
        ListNode pre = tail.next, curr = head, next = null;
        while (pre != tail) {
            next = curr.next;
            curr.next = pre;
            pre = curr;
            curr = next;
        }
        return new ListNode[]{tail, head};
    }
```

7.旋转链表

```java
public ListNode rotateRight(ListNode head, int k) {
    if (head == null || head.next == null) return head;
    int length = 1;
    ListNode oldTail = head;
    while (oldTail.next != null) {
        oldTail = oldTail.next;
        length++;
    }
    oldTail.next = head;
    ListNode newTail = head;
    for (int i = 0; i < length - k % length - 1; i++) {
        newTail = newTail.next;
    }
    ListNode newHead = newTail.next;
    newTail.next = null;
    return newHead;
}
```

8.两两交换链表的节点

```java
public ListNode swapPairs(ListNode head) {
    ListNode hair = new ListNode(0, head), pre = hair;
    while (pre.next != null && pre.next.next != null) {
        ListNode one = pre.next;
        ListNode two = pre.next.next;
        one.next = two.next;
        two.next = one;
        pre.next = two;
        pre = one;
    }
    return hair.next;
}
```

9.删除链表的倒数第N个节点

```java
public ListNode removeNthFromEnd(ListNode head, int n) {
    ListNode hair = new ListNode(0, head), fast = head, slow = hair;
    while (n > 0) {
        fast = fast.next;
        n--;
    }
    while (fast != null) {
        slow = slow.next;
        fast = fast.next;
    }
    slow.next = slow.next.next;
    return hair.next;
}
```

10.删除排序链表中的重复元素

```java
public ListNode deleteDuplicates(ListNode head) {
    ListNode curr = head;
    while (curr != null && curr.next != null) {
        if (curr.val == curr.next.val) {
            curr.next = curr.next.next;
        } else {
            curr = curr.next;
        }
    }
    return head;
}
```

11.删除排序链表中的重复元素II

```java
public ListNode deleteDuplicates(ListNode head) {
    ListNode hair = new ListNode(0, head), pre = hair, curr = head;
    while (curr != null) {
        while (curr.next != null && curr.val == curr.next.val) {
            curr = curr.next;
        }
        if (pre.next == curr) {
            pre = pre.next;
        } else {
            pre.next = curr.next;
        }
        curr = curr.next;
    }
    return hair.next;
}
```

kaikeba
开课吧