本周题目如下

1. https://leetcode-cn.com/problems/reverse-linked-list/
2. https://leetcode-cn.com/problems/swap-nodes-in-pairs/
3. https://leetcode-cn.com/problems/linked-list-cycle/
4. https://leetcode-cn.com/problems/linked-list-cycle-ii/
5. https://leetcode-cn.com/problems/reverse-nodes-in-k-group/

主要分为两大类，一类是环状链表，另一类是链表结点的交换。

环状链表

- https://leetcode-cn.com/problems/linked-list-cycle/ 查看链表是否有环

```python
def hasCycle(self, head: ListNode) -> bool:
    if not head:
        return False
    slow = fast = head
    while slow and fast and fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow is fast:
            return True
    return False
```

这是一道easy题，定义两个指针分别为快慢指针，快指针走两步，慢指针走一步，如果有环，必相遇，否则无环。

时间复杂度：O(n)

空间复杂度: O(1)

- https://leetcode-cn.com/problems/linked-list-cycle-ii/ 查找链表的入环结点

```python
def detectCycle(self, head):
    node_dict = {}
    current_node = head
    index = 0
    while current_node:
        if current_node in node_dict:
            return current_node
        node_dict[current_node] = index
        current_node = current_node.next
        index += 1
    return None
```

对于寻找入环结点的题，入环结点的特点在于有两个结点的下一个结点是同一个结点，这个结点就是入环结点，因此在遍历结点时，入环结点必然出现在已经遍历过的结点里，只需要保存遍历过的结点进行判断就行了。
时间复杂度: O(n)
空间复杂度: O(n)

链表结点交换

主要有三道题目，后面两个题目其实是前面题目的升级版，只要第一道会做话，全都是一样的啦。

- https://leetcode-cn.com/problems/reverse-linked-list/ 反转链表

```
1
2  def reverseList(self, head):
3      if not head:
4          return head
5      pre_node = None
6      current_node = head
7      while current_node:
8          next_node = current_node.next
9          current_node.next = pre_node
10         pre_node = current_node
11         current_node = next_node
12     return pre_node
```

定义两个引用分别指向前后两个结点，依次遍历每个结点，每次将当前结点的next 转向，同时保存当前结点的下一个结点以防丢失引用。
时间复杂度: O(n)
空间复杂度: O(1)

- https://leetcode-cn.com/problems/swap-nodes-in-pairs/ 两两反转链表

```
1  def swapPairs(self, head):
2      new_head = ListNode()
3      new_head.next = head
4      current_head = new_head
5
6      while head:
7          next_node = head.next
8          if not next_node:
9              return new_head.next
10         next_next_node = next_node.next
```

```
        current_head.next = next_node
        next_node.next = head
        head.next = next_next_node
        current_head = head
        head = next_next_node
    return new_head.next
```

上面那道题是每遍历一个结点就进行反转，这道题目就是每遍历两个结点再进行反转，考虑到后面可能凑不够两个结点，因此没有将全链表进行反转，而是新建一个头结点来指向head结点，每次取两个结点进行反转，每次进行反转前保存好下一个结点的引用，以防丢失。

时间复杂度: O(n)

空间复杂度: O(1)

- https://leetcode-cn.com/problems/reverse-nodes-in-k-group/ 按K个结点反转链表

```
def reverseKGroup(self, head, k):
    if k <= 1:
        return head
    # create a new head node
    new_head = ListNode()
    new_head.next = head
    current_head = new_head

    while head:
        # record the current node
        current_node = head
        for i in range(k - 1):
            head = head.next
            if not head:
                return new_head.next
        # record the next node
        next_node = head.next
        head.next = None
        temp_head, temp_last_node = self.reverseList(current_node)
        current_head.next = temp_head
        current_head = temp_last_node
        head = next_node
        temp_last_node.next = head
    return new_head.next

def reverseList(self, head):
```

```
    pre_node = None
    last_node = head
    current_node = head
    while current_node:
        next_node = current_node.next
        current_node.next = pre_node
        pre_node = current_node
        current_node = next_node
    return pre_node, last_node
```

这道题跟上面那道是非常相似的，只是从每2个结点反转变成每k个结点反转，那么解法也就基本一样了，这边是每次取k个结点进行反转，每次反转前保存好当前的头结点和下一个结点，将取出的k个结点进行反转然后返回头结点和尾结点，这时就可以将这个子链表合并上去了。

时间复杂度: O(n)

空间复杂度: O(1)