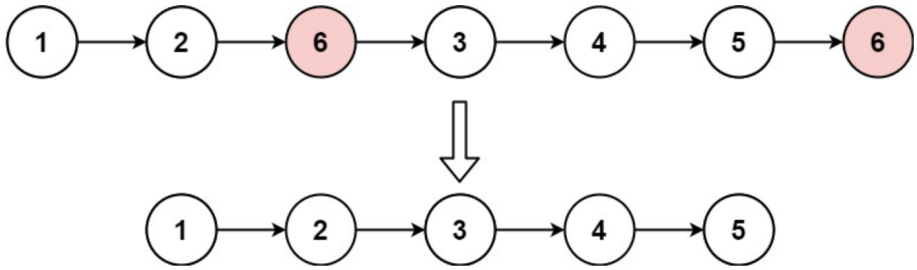
链表

```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
```

移除链表元素




```python
class Solution:
    def removeElements(self, head: Optional[ListNode], val: int) ->
Optional[ListNode]:
        dummy = ListNode(next = head)
        cur = dummy

        while cur.next:
            if cur.next.val == val:
                cur.next = cur.next.next
            else:
                cur = cur.next

        return dummy.next
```

设计链表[背]

> *self.dummy 的index为 -1*
>
> *self.dummy.next，也就是head 的index为 0*

```python
cur = dummy
for i in range(index+1):
    cur = cur.next
return cur.val
```

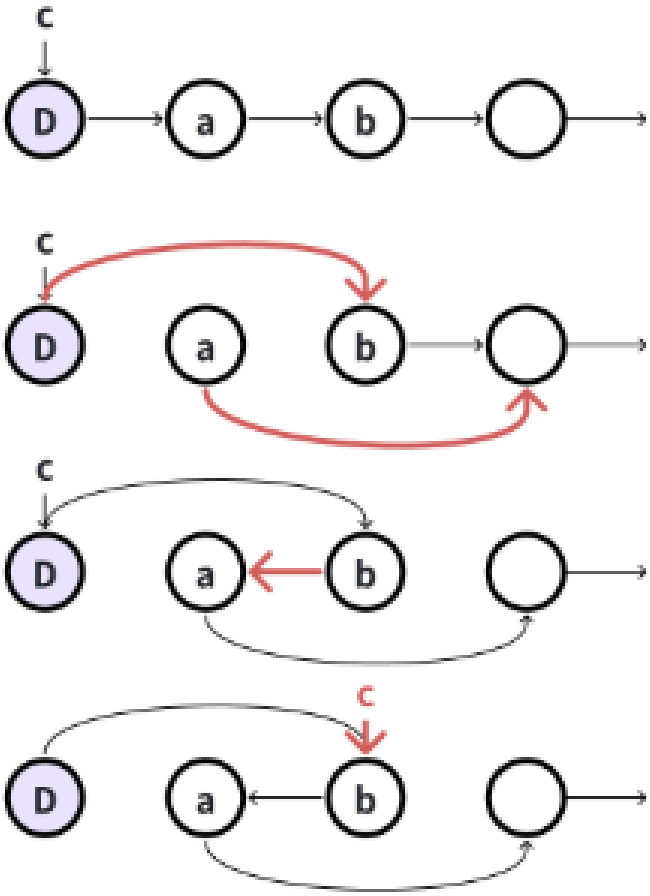
```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class MyLinkedList:
    def __init__(self):
        self.dummy = ListNode()
        self.size = 0

    def get(self, index: int) -> int:
        if index < 0 or index >= self.size:
            return -1

        cur = self.dummy.next
        for i in range(index):
            cur = cur.next

        return cur.val

    # 接链表头
    def addAtHead(self, val: int) -> None:
        self.dummy.next = ListNode(val, self.dummy.next)
        self.size += 1

    # 接链表尾巴
    def addAtTail(self, val: int) -> None:
        cur = self.dummy
        while cur.next:
            cur = cur.next
        cur.next = ListNode(val)
        self.size += 1

    def addAtIndex(self, index: int, val: int) -> None:
        if index < 0 or index > self.size:
            return

        cur = self.dummy
        for i in range(index):
            cur = cur.next
        cur.next = ListNode(val, cur.next)
        self.size += 1

    def deleteAtIndex(self, index: int) -> None:
        if index < 0 or index >= self.size:
            return

        cur = self.dummy
        for i in range(index):

```

```
            cur = cur.next
        cur.next = cur.next.next
        self.size -= 1
```

反转链表[无dummy]

```
class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        cur, pre = head, None
        while cur:
            tmp = cur.next # 暂存后继节点 cur.next
            cur.next = pre # 修改 next 引用指向
            pre = cur      # pre 暂存 cur
            cur = tmp      # cur 访问下一节点
        return pre
```

两两交换链表中的节点



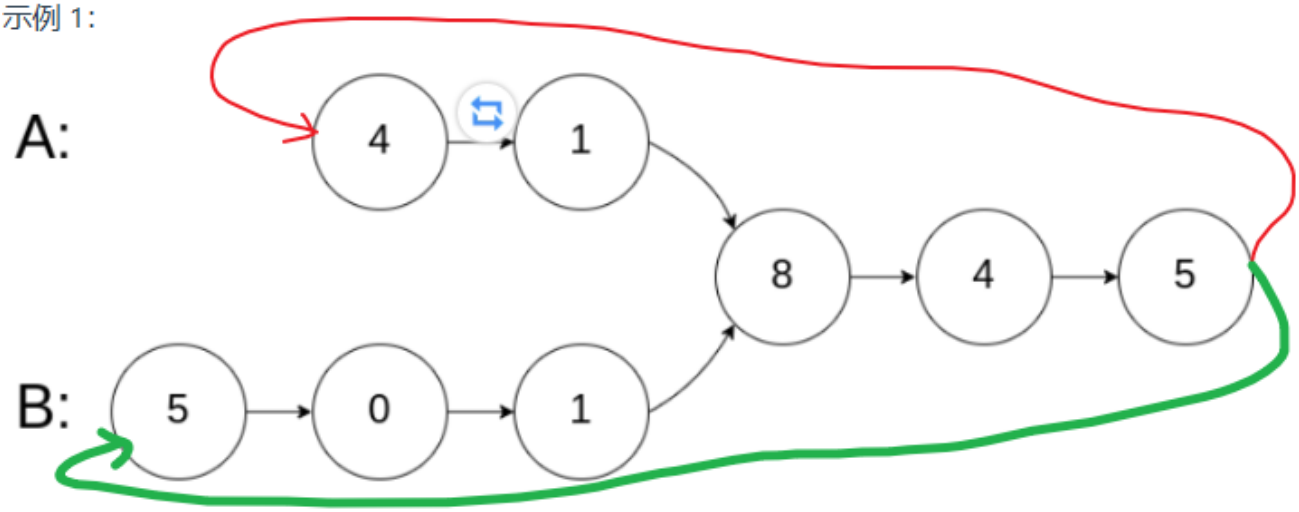
```python
class Solution:
    def swapPairs(self, head: ListNode) -> ListNode:
        dummy = ListNode(next=head)
        c = dummy
        while c.next and c.next.next:
            a, b = c.next, c.next.next
            c.next, a.next = b, b.next
            b.next = a
            c = c.next.next
        return dummy.next
```

删除链表的倒数第 N 个结点

```python
class Solution:
    def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:
        dummy = ListNode(0, head)

        slow = fast = dummy

        for i in range(n+1): # n+1 步
            fast = fast.next

        while fast:
            slow = slow.next
            fast = fast.next

        slow.next = slow.next.next

        return dummy.next
```

链表相交

示例 1：



```python
class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> ListNode:
        if not headA or not headB:
            return None

        A, B = headA, headB
        # AB若有交点，返回交点；没有交点，刚好链尾返回None
        while A != B:
            A = A.next if A else headB
            B = B.next if B else headA
        return A
```

环形链表[背]

```python
class Solution:
    def hasCycle(self, head):
        slow = fast = head
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next
            if slow == fast:
                return True
        return False
```

环形链表 II

```python
class Solution:
    def detectCycle(self, head: ListNode) -> ListNode:
        f=s=head
        while f and f.next:
            f=f.next.next
            s=s.next
            if f==s:    # 将上一题改写
                point=head
                while point != s:
                    point=point.next
                    s=s.next
                return point
        return None
```