反转链表

alt text

解法一，迭代法：

维护迭代指针，通过**prev**实现当前节点的翻转。

```cpp
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
    ListNode *prev=NULL,*temp = head;
    while(temp)
    {
        ListNode *next = temp->next;
        temp->next = prev;
        prev = temp ;
        temp = next;
    }
    return prev;
    }
};
```

解法二，头插法思想：

```cpp
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
    ListNode new_head, *p =head ,*q;
    new_head.next = NULL;
    while(p)
    {
        q = p->next;
        p->next = new_head.next;
        new_head.next = p;
        p = q;
    }
    return new_head.next;
    }
};
```

解法三，递归法：

```cpp
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
    if(head == NULL || head->next == NULL) return head;
    ListNode* tail = head->next;
    ListNode* newhead = reverseList(head->next);   //先reverseList
    head->next = tail->next;       把head->next先指向反转后的末端的下一个
```

```
    tail->next = head;      □□newhead
    return newhead;
    }
};
```
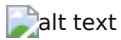
□□□□□□□

□□□□□

- □□□□□□□□□
- □□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□false
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□true

□□□□□□□□□□□□

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        ListNode *p = head , *q = head;
        while(q && q->next)    //□□□□q->next□□□null□□□□
        {                           □□□□□□□□□□□□□□□□□
            p = p->next;
            q = q->next->next;
            if(p == q) return true;
        }
        return false;
    }
};
```

□□□

□□□□□□□□□□□□□□□□□

alt text

- □□□1□□□□1□□□□
- □□□2□□□□□□□□□□□□□□□□□□1 □□□□□□□□□□□□□□□□□□□□

□□□

```
class Solution {
public:
    int get(int n)
    {
        int y=0,d;
        while(n)
        {
        d = n % 10;
        y += d * d;
        n = n / 10;
        }
        return y;
```

```
    }
    bool isHappy(int n) {
        int p = n,q = n;
        while(q!=1)  // q□□□□□□□□□q□□□□□□□□□q□□□□□□
        {                1□□p□□□□□□□□□1□
            p=get(p);
            q=get(get(q));
            if(p==q && q!=1) // p□q□□□□□□□□1□□□□□
            return false;
        }
        return true;
    }
};
```

□□□□□□□□□

□□□□

alt text

□□□□□

```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if(head == NULL) return head;
    int len = 0;        //□□□□
    ListNode *p = head ;
    ListNode *q = head,*temp,*newhead;
    while (q)
    {
        q = q->next;
        len+=1;
    }
    int x = k % len;
    if(x == 0) return head;
    for(int i = 1;i < len-x ; i++)//□□□□□□
        p = p->next;            //□□□□□□□□□
    temp = p->next;         //□□□□□□□□□□
    newhead = p->next;
    p->next = NULL;         //□□□□□□□□□null
    while(temp->next != NULL) //□□□□□□□□□□
    temp = temp->next;
    temp->next = head;
    return newhead;
    }
};
```

□□□□□□□ □□□□□□□

```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if(head == NULL)    return head;
        ListNode *p = head , *q = head,*temp = head;
        int len = 0;
        while(head)
        {
            head = head->next;
            len+=1;
        }
        k %= len;
        if(k==0) return temp;
        for(int i = 0;i <= k ;i++)
        q = q->next;
        while(q)
        {
            p = p->next;
            q = q->next;
        }
        q = p->next;
        p->next = NULL;
        p = q;
        while(p->next)
        p = p->next;
        p->next = temp;
        return q;
    }
};
```

删除链表的倒数第N个节点

alt text 如下图所示

```
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {

    ListNode *p = head,*q = head;
    ListNode new_head;
    new_head.next = head;
    int len = 0;
    while(q)
    {
        q = q->next;
        len +=1;
    }
    if(n == len)
    {
    new_head.next = head->next;
```

```
    return new_head.next;
    }
    for(int i = 1;i < len-n ;i++)
        p = p->next;
        p->next = p->next->next;
        return head;
    }
};
```

在这里还是用到了之前我们说过的技巧即**return head**，然后我们发现提交的时候**return**部分错了，这是因为

这里我们要注意几个问题，第一个问题

- 首先我们的虚拟头节点只能是定义一个节点，而不是指针
- 其次我们要循环走n+1步
- 最后我们走到的这个节点如果走到最后是null，那么说明这里的循环操作要进行判断一下

```
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode new_head , *p = &new_head , *q = p;
        new_head.next = head;
        for(int i = 1; i <= n+1;i++)
        q = q->next;
        while (q)
        {
            p = p->next;
            q = q->next;
        }
        p->next = p->next->next;
        return new_head.next;
    }
};
```

总结：

1. 虚拟头节点只能定义成一个节点，而不是指针
2. 利用快慢指针

环形链表II

 alt text 给定一个链表的头节点 ，返回链表开始入环的第一个节点。

- 通过快慢指针来进行判断，也就是我们之前说到的，快指针走两步，慢指针走一步，相遇（a）
- 此时我们再来看数学公式
- 此时需要我们开始注意的是，此时我们需要看此时我们两个链表走的距离
  - 如果链表走x步，此时我们可以知道，从头节点开始到环（a）步长
  - 如果我们此时知道了这是x-a
  - 那么我们此时可以知道是x-a步
  - 此时我们可以知道这是x-a，那么此时我们可以知道这是x-（x-a）=a步长了

- 如上图所示,可以计算出相遇点。alt text

- 从起点和相遇点,各设一个指针,每次走一步,两个指针必定相遇,且相遇的第一个点为环入口点。

【代码】:

```cpp
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        ListNode *q =head , *p = head;
        while(p && p->next)
        {
            q = q->next;
            p = p->next->next;
            if(p==q)
            break;
        }
        if (p == NULL || p->next == NULL)
        return NULL;
        q = head;
        while(p != q)
        {
            p = p->next;
            q = q->next;
        }
        return q;
    }
};
```