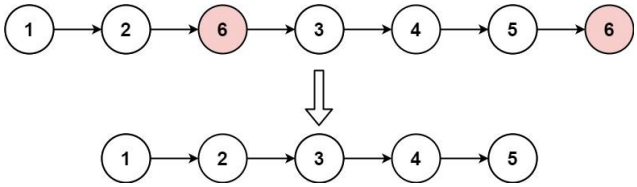
给你一个链表的头节点 head 和一个整数 val ，请你删除链表中所有满足 Node.val == val 的节点，并返回 新的头节点 。



好像很简单？

（困难 扩展）给你一个链表数组，每个链表都已经按升序排列。
请你将所有链表合并到一个升序链表中，返回合并后的链表。
示例 1:
输入：lists = [[1,4,5],[1,3,4],[2,6]]
输出：[1,1,2,3,4,4,5,6]
解释：链表数组如下：
[
  1->4->5,
  1->3->4,
  2->6
]
将它们合并到一个有序链表中得到。
1->1->2->3->4->4->5->6

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* mergeKLists(struct ListNode** lists, int listsSize)
{
    if(listsSize==0)
    {
        return NULL;
    }
    if(listsSize==1)
    {
        return lists[0];
    }
    int k=0 , i=0 , arr[listsSize];
    memset(arr,0,listsSize*sizeof(int));
```

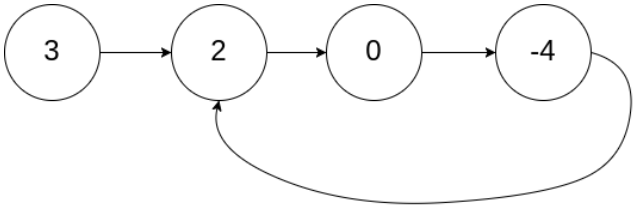
```
    for(i=0;i<listsSize;i++)
    {
        if(lists[i]!=NULL)
        {
            arr[k++]=i;
        }
    }
    struct ListNode *_head1=lists[arr[0]] , *_head2=lists[arr[1]];
    for(i=0;i<k-1;i++)
    {
        struct  ListNode  *head1=NULL  ,  *head2=NULL  ,  *node1=_head1  ,
*node2=_head2;
        while(node1 && node2)
        {



if(node1->val > node2->val)
            {

node2=(node2->next&&node2->next->val<node1->val)?node2->next:(head2=node2->n
ext,node2->next=node1,head2);
            }
            else if(node1->val <= node2->val)
            {


node1=(node1->next&&node1->next->val<=node2->val)?node1->next:(head1=node1->
next,node1->next=node2,head1);
            }
        }
        _head1=_head1->val<=_head2->val?_head1:_head2;
        if(i+2==k)
        {
            return _head1;
        }
        _head2=lists[arr[i+2]];
    }
    return _head1;
}
```

环形链表

给你一个链表的头节点 head ，判断链表中是否有环。

如果链表中有某个节点，可以通过连续跟踪 next 指针再次到达，则链表中存在环。 为了表示给定链表中的环，评测系统内部使用整数 pos 来表示链表尾连接到链表中的位置（索引从 0 开始）。如果 pos 是 -1，则在该链表中没有环。注意：pos 不作为参数进行传递，仅仅是为了标识链表的实际情况。

如果链表中存在环，则返回 true 。 否则，返回 false 。



输入：head = [3,2,0,-4], pos = 1

输出：true

解释：链表中有一个环，其尾部连接到第二个节点。

解题思路
思路 1.设置两个指针，一个每次前进两格，一个每次前进一格
2.若两个指针指向相同位置，则存在循环；否则，无循环

代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
```

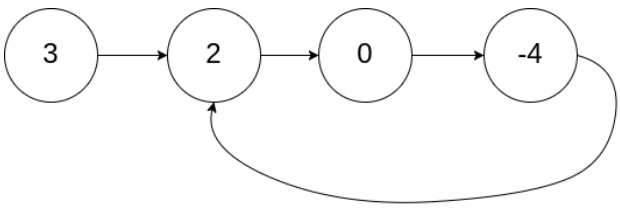
```
bool hasCycle(struct ListNode *head)
{
    struct ListNode *h;
    struct ListNode *n;
```

```
    if(!head)
    {
        return false;
    }

    h=head->next;
    n=head->next;

    while(h&&h->next&&h->next->next)
    {
        h=h->next->next;
        n=n->next;
        if(h->next==n->next)
        {
            return true;

        }

    }
    return false;
}
```

环形链表 II

给定一个链表，返回链表开始入环的第一个节点。 如果链表无环，则返回 null。

如果链表中有某个节点，可以通过连续跟踪 next 指针再次到达，则链表中存在环。 为了表示给定链表中的环，评测系统内部使用整数 pos 来表示链表尾连接到链表中的位置（索引从 0 开始）。如果 pos 是 -1，则在该链表中没有环。注意：pos 不作为参数进行传递，仅仅是为了标识链表的实际情况。



输入：`head = [3,2,0,-4], pos = 1`

输出：返回索引为 `1` 的链表节点

解释：链表中有一个环，其尾部连接到第二个节点。

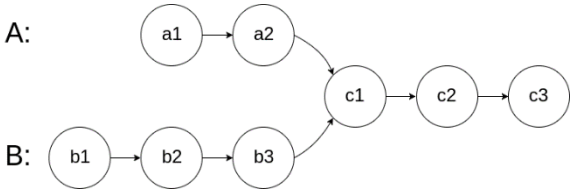
快慢指针
先用上题的方法判断是否有环，
如果有环，记录下快慢指针相遇的地方；
用一个指针遍历环，再次到此节点时可以得出环的节点数 n；
然后再用快慢指针，这次快指针先前进 n 个节点，此时快慢指针间隔 n 个节点；
再同时向前移动，直至两指针再次相遇点即为开始入环的第一个节点；
此时快指针入环已遍历一圈，慢指针刚入环.

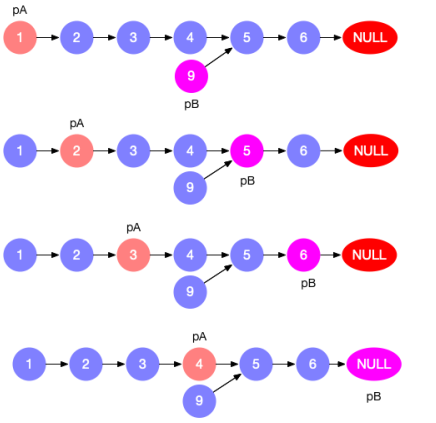
```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *    int val;
 *    struct ListNode *next;
 * };
 */
struct ListNode *detectCycle(struct ListNode *head) {
    if(head == NULL || head -> next == NULL)
        return NULL;
    struct ListNode* slow = head;
    struct ListNode* fast = head -> next;
    while(slow != fast){
        if(fast -> next == NULL || fast -> next -> next == NULL)
            return NULL;
        slow = slow -> next;
        fast = fast -> next -> next;
    }
    fast = fast -> next;
    int n = 1;
    while(fast != slow){
        fast = fast -> next;
        n++;
    }
    slow = head,fast = head;
    for(int i = 0;i < n;i++)
        fast = fast -> next;
    while(fast != slow){
        fast = fast -> next;
        slow = slow -> next;
    }
    return fast;
}
```

相交链表（相交是指地址相同 而不只是 data 一样）（很秀!!）

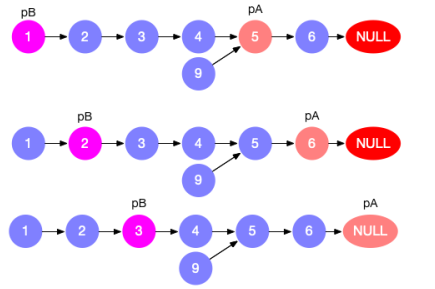
给你两个单链表的头节点 headA 和 headB ，请你找出并返回两个单链表相交的起始节点。如果两个链表不存在相交节点，返回 null 。

图示两个链表在节点 c1 开始相交：

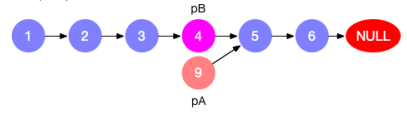




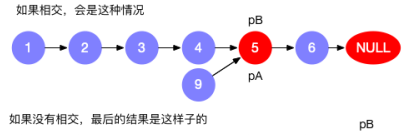
pB到了B链表末尾，指向链表A的头部，此时A和B的长度差为B链表的长度也就是3



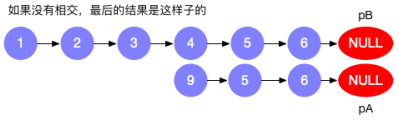
pA到了末尾，移动到B链表的头部
此时pA和pB到达末尾长度一样，会同时到达尾部



如果相交，会是这种情况



如果没有相交，最后的结果是这样子的



两个指针分别从 A、B 两个链表开始同时向后遍历，遍历所有结点后，两个指针再分别从另一个链表头结点出发遍历链表，两个指针会在两个链表交点相遇。
要注意链表为空的情况。

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode *getIntersectionNode(struct ListNode *headA, struct ListNode *headB) {
    if(headA == NULL || headB == NULL)         //链表为空
        return NULL;

    struct ListNode *qA = headA, *qB = headB;   //指针 qA 从 A 链表出发，qB 从 B 链表出发

    while(qA != qB){                            //指针 qA 和 qB 同时向后遍历
        if(qA != NULL)
            qA = qA->next;
        else
            qA = headB;                         //qA 指向 B 链表头结点
        if(qB != NULL)
            qB = qB->next;
        else
            qB = headA;                         //qB 指向 A 链表头结点
    }

    return qA;
}
```

删除中间结点

给你一个链表的头节点 head 。删除 链表的 中间节点 ，并返回修改后的链表的头节点 head 。

长度为 n 链表的中间节点是从头数起第 ⌊n / 2⌋ 个节点（下标从 0 开始），其中 ⌊x⌋ 表示小于或等于 x 的最大整数。

对于 n = 1、2、3、4 和 5 的情况，中间节点的下标分别是 0、1、1、2 和 2 。

解题思路（与前面题有类似）
基本思路设计两个指针，fast 一次走两步，slow 一次走一步。当 fast 或 fast->next 为空时，slow 必定指向中间节点

代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */


struct ListNode* deleteMiddle(struct ListNode* head){
    if(!head->next)
    {
        return NULL;
    }
    struct ListNode*fast=head;
    struct ListNode*slow=head;
    struct ListNode*pre;
    while(fast&&fast->next)
    {
        pre=slow;
        fast=fast->next->next;
        slow=slow->next;
    }
    pre->next=slow->next;
    free(slow);
    return head;
}
```