1. 链表

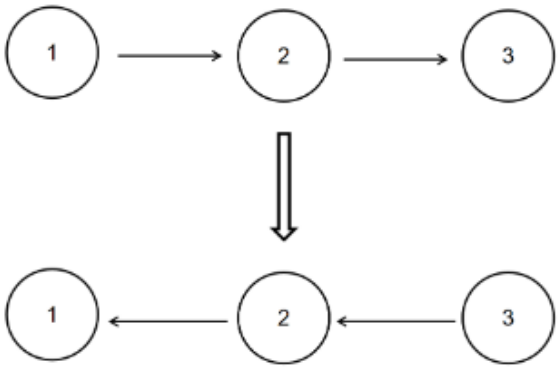
1. 反转链表

题目描述:

给定一个单链表的头结点pHead(该头节点是有值的，比如在下图，它的val是1)，长度为n，反转该链表后，返回新链表的表头。

数据范围： $0 \leq n \leq 1000$
要求：空间复杂度 $O(1)$ ，时间复杂度 $O(n)$ 。

如当输入链表{1,2,3}时，
经反转后，原链表变为{3,2,1}，所以对应的输出为{3,2,1}。
以上转换过程如下图所示：



示例1

```
输入：  {1,2,3}
返回值：{3,2,1}
```

示例2

```
输入：  {}
返回值：{}
说明：  空链表则输出空
```

解题代码:

```java
import java.util.*;

/*
 * public class ListNode {
 *   int val;
 *   ListNode next = null;
 *   public ListNode(int val) {
 *     this.val = val;
 *   }
 * }
 */

public class Solution {
    /**
     * 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
     *
     *
     * @param head ListNode类
     * @return ListNode类
     */
    public ListNode ReverseList (ListNode head) {
        // write code here
        if (head == null || head.next == null) {
            return head; // 如果链表为空或只有一个节点，直接返回原链表
        }
        ListNode prev = null;
        ListNode current = head;
        ListNode nextNode;

        while (current != null) {
            nextNode = current.next; // 保存下一个节点的引用
            current.next = prev; // 当前节点指向前一个节点
            prev = current; // 前一个节点更新为当前节点
            current = nextNode; // 当前节点更新为下一个节点
        }

        return prev; // prev现在是反转后的链表头结点
    }
}
```

```
/**
 * struct ListNode {
 *  int val;
 *  struct ListNode *next;
 *  ListNode(int x) : val(x), next(nullptr) {}
 * };
 */
class Solution {
public:
    /**
     * 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
     *
     *
     * @param head ListNode类
     * @return ListNode类
     */
    ListNode* ReverseList(ListNode* head) {
        // write code here
        if (head == nullptr || head->next == nullptr ) {
            return head;
        }
        ListNode* prev_node = nullptr;
        ListNode* current = head;
        ListNode* nextNode;
        while (current != nullptr) {
            nextNode = current->next;
            current->next = prev_node;
            prev_node = current;
            current = nextNode;
        }

        return prev_node;
    }
};
```

2. 链表内指定区间反转

题目描述：

描述

将一个节点数为 size 链表 m 位置到 n 位置之间的区间反转，要求时间复杂度 $O(n)$，空间复杂度 $O(1)$。
例如：
给出的链表为 $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow NULL, m = 2, n = 4$,
返回 $1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow NULL$.

数据范围： 链表长度 $0 < size \leq 1000$，$0 < m \leq n \leq size$，链表中每个节点的值满足 $|val| \leq 1000$
要求：时间复杂度 $O(n)$ ，空间复杂度 $O(n)$
进阶：时间复杂度 $O(n)$，空间复杂度 $O(1)$

示例1

输入： `{1,2,3,4,5},2,4` 复制

返回值：`{1,4,3,2,5}` 复制

示例2

输入： `{5},1,1` 复制

返回值：`{5}` 复制

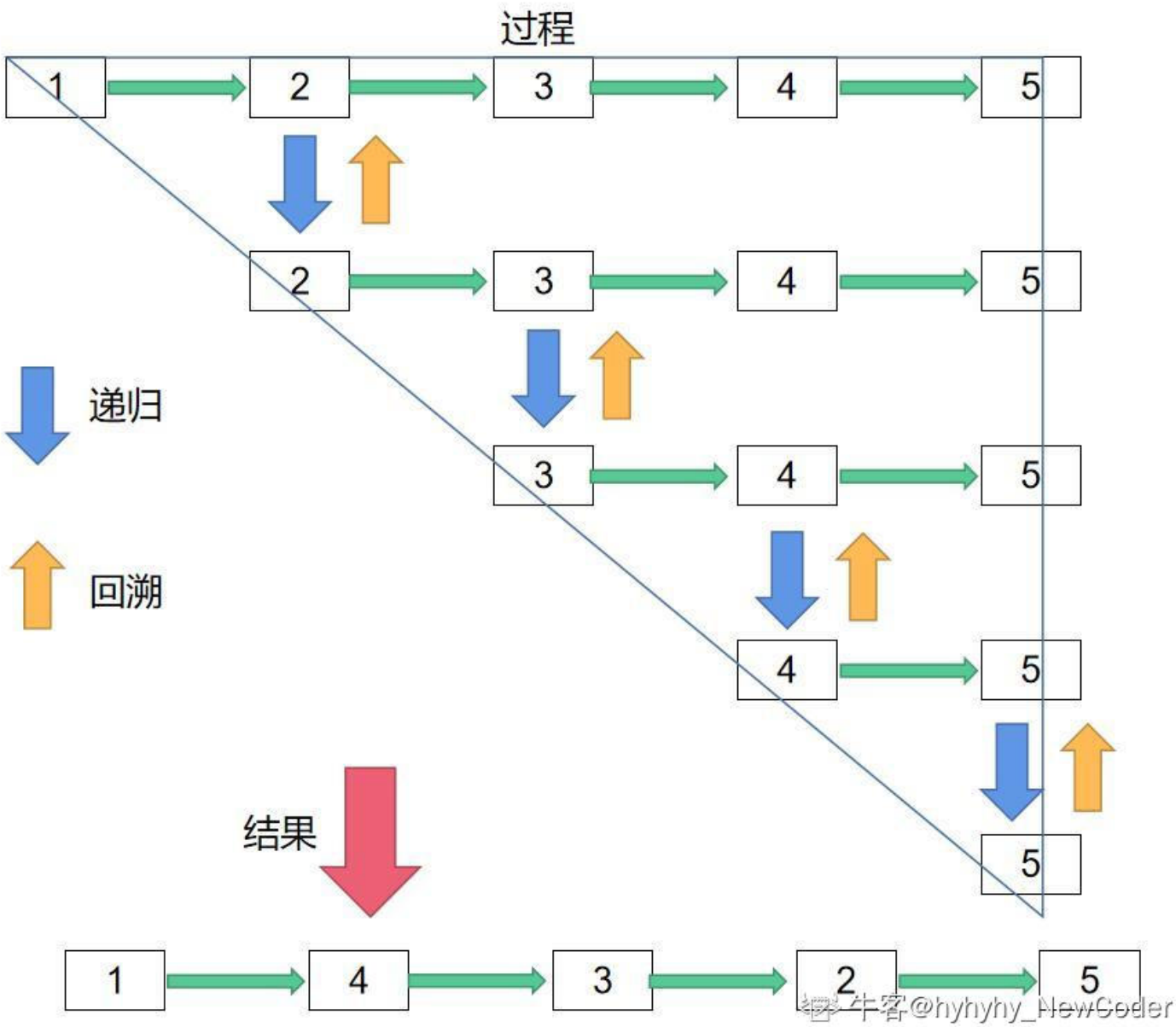
解题代码:

```java
import java.util.*;

/*
 * public class ListNode {
 *   int val;
 *   ListNode next = null;
 *   public ListNode(int val) {
 *     this.val = val;
 *   }
 * }
 */

public class Solution {
    /**
     * 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
     *
     *
     * @param head ListNode类
     * @param m int整型
     * @param n int整型
     * @return ListNode类
     */
    public ListNode reverseBetween (ListNode head, int m, int n) {
        // write code here
        //我们可以在链表前加一个表头，后续返回时去掉就好了，因为如果要从链表头的位置开始反转，在多了一个
        ListNode HEAD = new ListNode(0);
        HEAD.next = head;
        ListNode prev = null;
        ListNode current = HEAD;
        ListNode nextNode;
        for(int i = 0;i < m;i++){
            nextNode = current.next;
            prev = current;
            current = nextNode;
        }
        ListNode after = current;
        for(int i = 0;i < n - m;i++){
            after = after.next;
        }
        nextNode = after.next;
        after.next = null;
        after = nextNode;
        prev.next = ReverseList(current);
```

```
        while(prev.next!=null){
            prev = prev.next;
        }
        prev.next = after;

        return HEAD.next;
    }
}
```

大佬思路:

对于反转链表，我们使用递归的思想，将大问题转换为小问题，然后进行相应的求解即可。对于递归过程，判断n的数值，当n为1时返回head指针，否则进行递归，并且反转链表，最后进行拼接，返回拼接之后的链表。（代码简单，但是空间复杂度：其最坏递归空间复杂度为O(N).）



```
class Solution {
public:
    ListNode* tmp = NULL;
    ListNode* reverseBetween(ListNode* head, int m,int n){
        if( m == 1) return reverse(head, n);
        ListNode* p = reverseBetween(head->next, m-1,n-1); // m,n在子问题中要减1
        head->next = p; //拼接反转之后的部分
        return head;
    }
    ListNode* reverse(ListNode* head,int n)
    {//反转链表，递归的思想，参考贵州大学的漫漫云天自翱翔的思路
        if(n == 1)
        {
            tmp = head->next;
            return head;
        }
        ListNode* new_head = reverse(head->next,n-1);  //子问题，位置要减1
        head->next->next = head; //反转
        head->next = tmp;  //拼接尾部
        return new_head;
    }
};
```

3. 链表中的节点每k个一组翻转

题目描述：

描述

将给出的链表中的节点每 k 个一组翻转，返回翻转后的链表
如果链表中的节点数不是 k 的倍数，将最后剩下的节点保持原样
你不能更改节点中的值，只能更改节点本身。

数据范围： $0 \le n \le 2000$ ， $1 \le k \le 2000$ ，链表中每个元素都满足 $0 \le val \le 1000$
要求空间复杂度 $O(1)$，时间复杂度 $O(n)$
例如：
给定的链表是 $1 \to 2 \to 3 \to 4 \to 5$
对于 $k = 2$, 你应该返回 $2 \to 1 \to 4 \to 3 \to 5$
对于 $k = 3$, 你应该返回 $3 \to 2 \to 1 \to 4 \to 5$

示例1

输入： {1,2,3,4,5},2 复制

返回值：{2,1,4,3,5} 复制

示例2

输入： {},1 复制

返回值：{} 复制

解题代码:

```
public ListNode reverseKGroup (ListNode head, int k) {
    // write code here
    if (head == null || k == 1) {
        return head;
    }
    ListNode current = head;
    int num = 0;
    while(current!=null){
        current = current.next;
        num++;
    }
    int i = 1;
    for(;(i+k -1) <= num;i+=k){
        head = reverseBetween(head ,i,i+k-1);
    }

    return head;
}
```

大佬思路:

又快又小

```
/**
 * struct ListNode {
 *  int val;
 *  struct ListNode *next;
 * };
 */

class Solution {
public:
    /**
     *
     * @param head ListNode类
     * @param k int整型
     * @return ListNode类
     */
    ListNode* reverseKGroup(ListNode* head, int k) {
        // write code here
        ListNode *p=head;
        ListNode *new_head=head;
        int kk=k;
        ListNode *ne=NULL;
        ListNode *tail=NULL;
        while(p)
        {
            if(kk==1)
            {
                ne=p->next;
                p->next=NULL;
                ListNode *temp=reverse(head);//反转链表输入的head最后指向原链表的最后一个节点
                if(new_head==head)
                    new_head=temp;
                if(tail)
                    tail->next=p;
                tail=head;

                head=ne;
                p=ne;
                kk=k;
                continue;
            }
            kk--;
            p=p->next;
        }
```

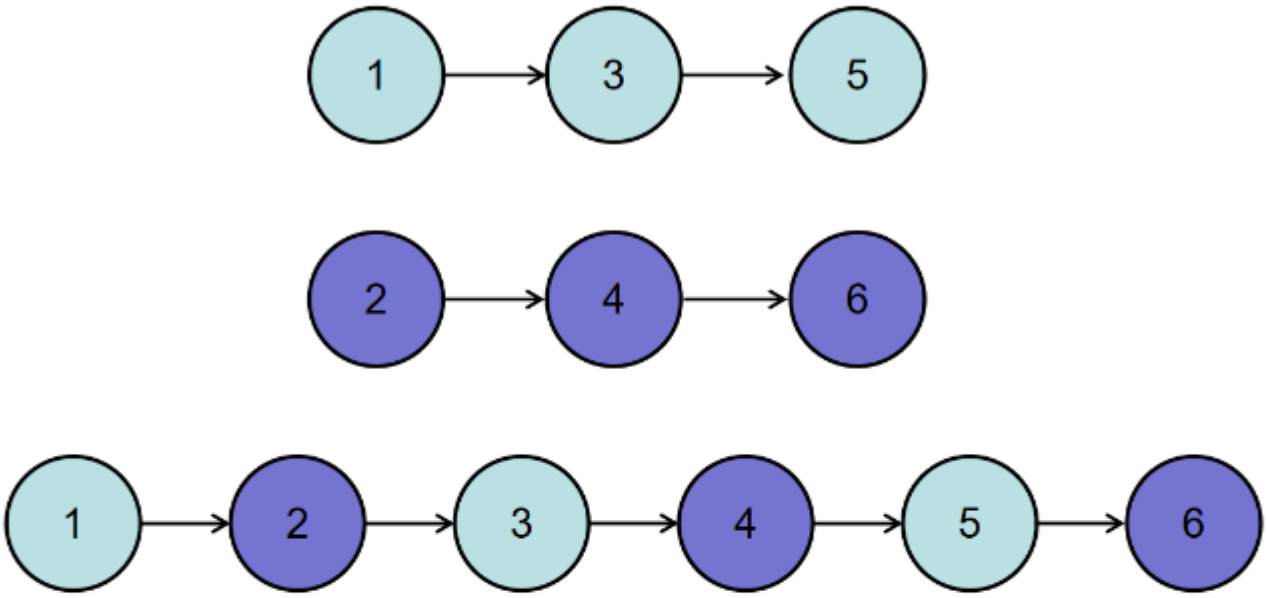
```
        if(tail)
            tail->next=head;
        return new_head;
    }
    ListNode *reverse(ListNode *start)
    {
        ListNode * p=nullptr;
        while(start)
        {
            ListNode *temp=start->next;
            start->next=p;
            p=start;
            start=temp;
        }
        return p;
    }
};
```

4.合并两个排序的链表

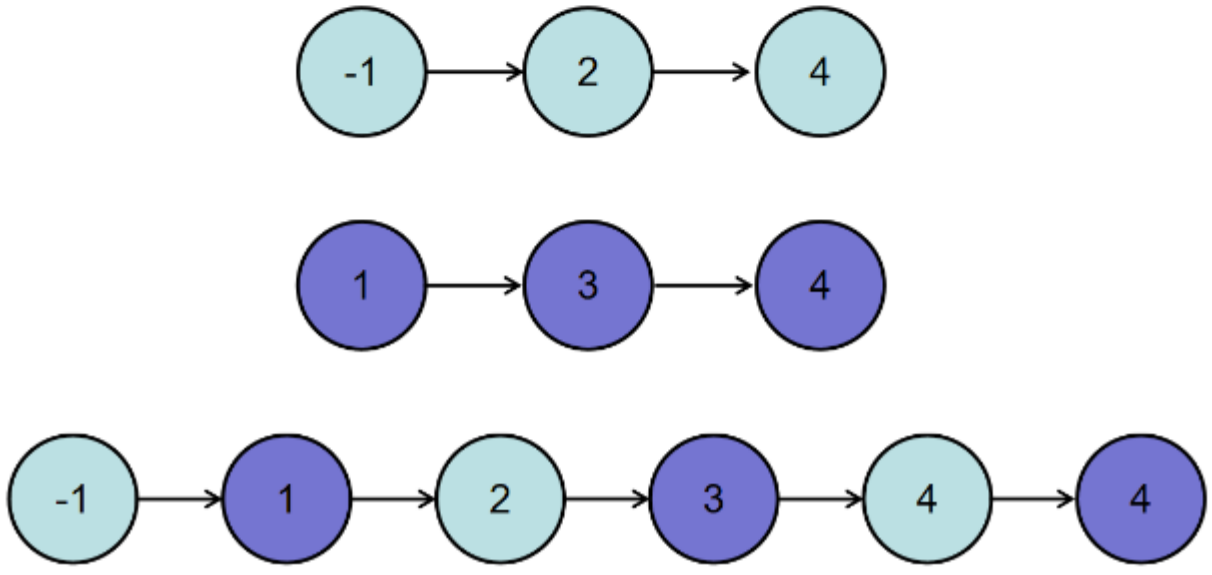
题目描述：

输入两个递增的链表，单个链表的长度为n，合并这两个链表并使新链表中的节点仍然是递增排序的。
数据范围：$0 \leq n \leq 1000$，$-1000 \leq$ 节点值 ≤ 1000
要求：空间复杂度 $O(1)$，时间复杂度 $O(n)$

如输入{1,3,5},{2,4,6}时，合并后的链表为{1,2,3,4,5,6}，所以对应的输出为{1,2,3,4,5,6}，转换过程如下图所示：



或输入{-1,2,4},{1,3,4}时，合并后的链表为{-1,1,2,3,4,4}，所以对应的输出为{-1,1,2,3,4,4}，转换过程如下图所示：



示例1

输入： {1,3,5},{2,4,6} 复制

返回值：{1,2,3,4,5,6} 复制

示例2

输入： {},{} 复制

返回值：{} 复制

示例3

输入： {-1,2,4},{1,3,4} 复制

返回值：{-1,1,2,3,4,4} 复制

```java
import java.util.*;

/*
 * public class ListNode {
 *   int val;
 *   ListNode next = null;
 *   public ListNode(int val) {
 *     this.val = val;
 *   }
 * }
 */

public class Solution {
    /**
     * 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
     *
     *
     * @param pHead1 ListNode类
     * @param pHead2 ListNode类
     * @return ListNode类
     */
    public ListNode Merge (ListNode pHead1, ListNode pHead2) {
        // write code here
        if(pHead1==null || pHead2==null){
            return (pHead1!=null)?pHead1:pHead2;
        }
        ListNode current1 = pHead1;
        ListNode current2 = pHead2;
        ListNode new_head = new ListNode(0);
        ListNode nextNode;
        ListNode tail = new_head;

        while(current1!=null && current2!=null){
            if(current1.val <= current2.val){
                nextNode = current1;
                tail.next = nextNode;
                tail = tail.next;
                current1 = current1.next;
            }else{
                nextNode = current2;
                tail.next = nextNode;
                tail = tail.next;
                current2 = current2.next;
```

```
            }
        }
        if(current1!=null){
            tail.next = current1;
        }else{
            tail.next = current2;
        }
        return new_head.next;
    }
}
```

大佬思路:

其实一样

```
/*
struct ListNode {
    int val;
    struct ListNode *next;
    ListNode(int x) :
            val(x), next(NULL) {
    }
};*/
class Solution {
public:
    ListNode* Merge(ListNode* pHead1, ListNode* pHead2) {
        if(pHead1 == nullptr){return pHead2;}
        if(pHead2 == nullptr){return pHead1;}

        ListNode* dummy = new ListNode(0);//这儿必须初始化
        ListNode* cur = dummy;

        while(pHead1 && pHead2){
            if(pHead1->val <= pHead2->val){
                cur->next = pHead1;
                pHead1 = pHead1->next;
            }
            else{
                cur->next = pHead2;
                pHead2 = pHead2->next;
            }
            cur = cur->next;
        }
        if(pHead1){cur->next = pHead1;}// 可以学学这儿怎么if
        else{cur->next = pHead2;}

        return dummy->next;
    }
};
```

5. 合并k个已排序的链表

题目描述：

描述

合并 k 个升序的链表并将结果作为一个升序的链表返回其头节点。

数据范围：节点总数 $0 \leq n \leq 5000$，每个节点的val满足 $|val| <= 1000$
要求：时间复杂度 $O(nlogn)$

示例1

输入：[{1,2,3},{4,5,6,7}] 复制

返回值：{1,2,3,4,5,6,7} 复制

示例2

输入：[{1,2},{1,4,5},{6}] 复制

返回值：{1,1,2,4,5,6} 复制

解题代码:

```
public ListNode mergeKLists (ArrayList<ListNode> lists) {
    // write code here
    if(lists.isEmpty()){
        return null;
    }
    ListNode new_head = lists.get(0);
    lists.remove(0);
    for(ListNode node:lists){
        new_head = Merge(new_head,node);
    }
    return new_head;
}
```

```
ListNode* mergeKLists(vector<ListNode*>& lists) {
    // write code here
    if(lists.empty()){
        return nullptr;
    }
    ListNode* new_head = lists[0];
    lists.erase(lists.begin());
    for(ListNode* node:lists){
        new_head = Merge(new_head,node);
    }
    return new_head;
}
```

大佬思路:

使用sort,

```
class Solution {
public:
    ListNode *mergeKLists(vector<ListNode *> &lists) {
        vector<int> v;
        for(int i=0;i<lists.size();i++){
            ListNode* temp=lists[i];
            while(temp){
                v.push_back(temp->val);
                temp=temp->next;
            }

        }
        sort(v.begin(),v.end());
        ListNode* head=new ListNode(0);
        ListNode* cur=head;
        for(vector<int>::iterator it=v.begin();it!=v.end();it++){
//              cur->val=*it;
            int a=*it;
            ListNode * temp=new ListNode(a);
            cur->next=temp;
            cur=cur->next;
        }
        return head->next;
    }
};
```

6. 判断链表中是否有环

题目描述：

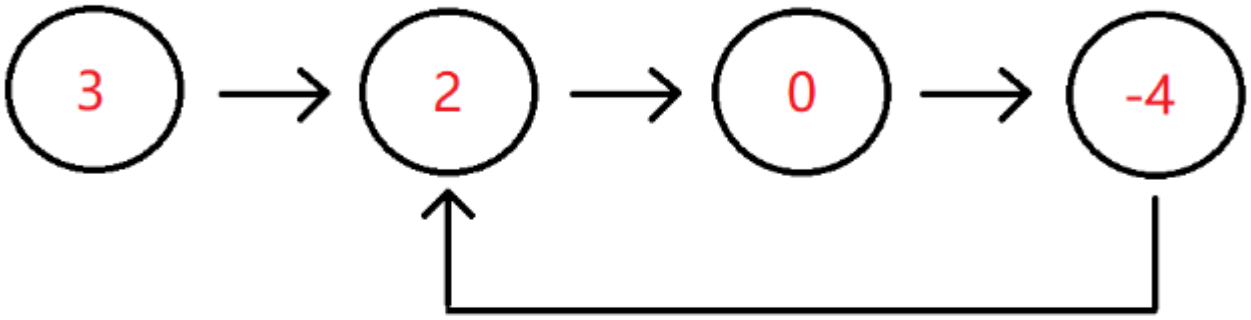
描述

判断给定的链表中是否有环。如果有环则返回true，否则返回false。

数据范围：链表长度 $0 \leq n \leq 10000$，链表中任意节点的值满足 $|val| <= 100000$
要求：空间复杂度 $O(1)$，时间复杂度 $O(n)$

输入分为两部分，第一部分为链表，第二部分代表是否有环，然后将组成的head头结点传入到函数里面。-1代表无环，其它的数字代表有环，这些参数解释仅仅是为了方便读者自测调试。实际在编程时读入的是链表的头节点。

例如输入{3,2,0,-4},1时，对应的链表结构如下图所示：



可以看出环的入口结点为从头结点开始的第1个结点（注：头结点为第0个结点），所以输出true。

示例1

输入：`{3,2,0,-4},1` 复制

返回值：`true` 复制

说明：第一部分`{3,2,0,-4}`代表一个链表，第二部分的`1`表示，`-4`到位置`1`（注：头结点为位置`0`），即`-4->2`存在一个链接，组成传入的`head`为一个带环的链表，返回`true`

示例2

输入：`{1},-1` 复制

返回值：`false` 复制

说明：第一部分`{1}`代表一个链表，`-1`代表无环，组成传入`head`为一个无环的单链表，返回`false`

示例3

输入： {-1,-7,7,-4,19,6,-9,-5,-2,-5},6 复制

返回值：true 复制

解题代码：

```
import java.util.*;
/**
 * Definition for singly-linked list.
 * class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public boolean hasCycle(ListNode head) {
        if(head == null){
            return false;
        }
        HashSet<ListNode> hashMap = new HashSet<>();

        while(head!=null){
            if(hashMap.contains(head)){
                return true;
            }
            hashMap.add(head);
            head = head.next;
        }
        return false;
    }
}
```

大佬思路：

一种是我的哈希表，一种是快慢指针

```python
class Solution:
    def hasCycle(self , head ):
        # write code here
        if not head:
            return head
        # 双指针  快慢指针
        slow = head
        fast = head
        while slow and fast:
            slow = slow.next
            if fast.next:
                fast = fast.next.next
            else:
                return False
            # 当双指针相遇 即表示指针有环
            if slow == fast:
                return True
        return False
```

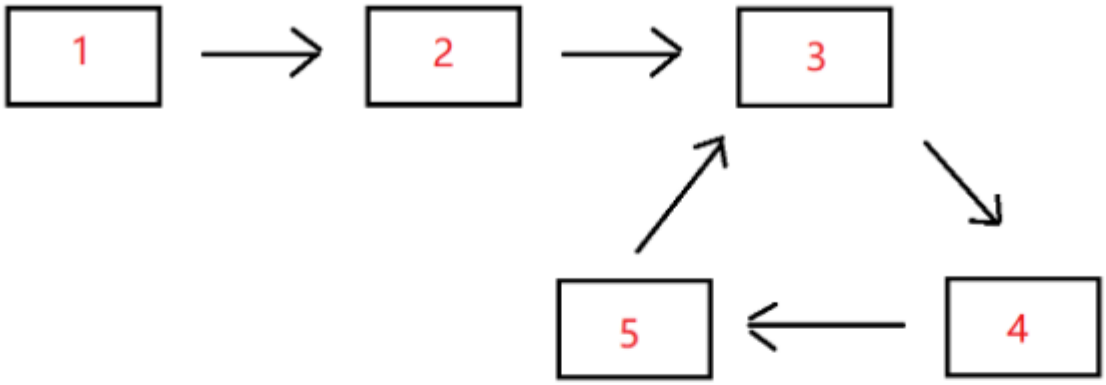
7. 链表中环的入口结点

题目描述：

给一个长度为n链表，若其中包含环，请找出该链表的环的入口结点，否则，返回null。

数据范围：$n \leq 10000$，$1 <= 结点值 <= 10000$
要求：空间复杂度 $O(1)$，时间复杂度 $O(n)$

例如，输入{1,2},{3,4,5}时，对应的环形链表如下图所示：



可以看到环的入口结点的结点值为3，所以返回结点值为3的结点。

输入描述：

输入分为2段，第一段是入环前的链表部分，第二段是链表环的部分，后台会根据第二段是否为空将这两段组装成一个无环或者有环单链表

返回值描述：

返回链表的环的入口结点即可，我们后台程序会打印这个结点对应的结点值；若没有，则返回对应编程语言的空结点即可。

示例1

输入：`{1,2},{3,4,5}` 复制

返回值：`3` 复制

说明：返回环形链表入口结点，我们后台程序会打印该环形链表入口结点对应的结点值，即3

示例2

输入：`{1},{}` 复制

返回值：`"null"` 复制

说明：没有环，返回对应编程语言的空结点，后台程序会打印"null"

示例3

输入： {},{2} 复制

返回值： 2 复制

说明： 环的部分只有一个结点，所以返回该环形链表入口结点，后台程序打印该结点对应的结点值，即2

解题代码:

```
public class Solution {

    public ListNode EntryNodeOfLoop(ListNode pHead) {
        if(pHead == null){
            return pHead;
        }
        HashSet<ListNode> hashMap = new HashSet<>();

        while(pHead!=null){
            if(hashMap.contains(pHead)){
                return pHead;
            }
            hashMap.add(pHead);
            pHead = pHead.next;
        }
        return null;
    }
}
```

大佬思路:

```java
public class Solution {

    public ListNode EntryNodeOfLoop(ListNode pHead) {
        if (pHead==null || pHead.next==null) return null;
        ListNode fast = pHead;
        ListNode slow = pHead;
        while (fast!=null && fast.next!=null) {
            fast=fast.next.next;
            slow=slow.next;
            // 快慢相遇，说明成环
            if (fast==slow) {
                ListNode tmp = pHead;
                while (tmp!=slow) {
                    tmp=tmp.next;
                    slow=slow.next;
                }
                return tmp;
            }
        }
        return null;
    }
}
```

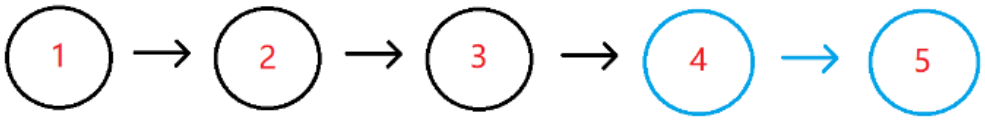
8. 链表中倒数最后k个结点

题目描述：

输入一个长度为 n 的链表，设链表中的元素的值为 a_i ，返回该链表中倒数第k个节点。
如果该链表长度小于k，请返回一个长度为 0 的链表。

数据范围：$0 \le n \le 10^5$，$0 \le a_i \le 10^9$，$0 \le k \le 10^9$
要求：空间复杂度 $O(n)$，时间复杂度 $O(n)$
进阶：空间复杂度 $O(1)$，时间复杂度 $O(n)$

例如输入{1,2,3,4,5},2时，对应的链表结构如下图所示：



其中蓝色部分为该链表的最后2个结点，所以返回倒数第2个结点（也即结点值为4的结点）即可，系统会打印后面所有的节点来比较。

示例1

输入：{1,2,3,4,5},2 复制
返回值：{4,5} 复制
说明：返回倒数第2个节点4，系统会打印后面所有的节点来比较。

示例2

输入：{2},8 复制
返回值：{} 复制

解题代码:

```java
public class Solution {
    /**
     * 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
     *
     *
     * @param pHead ListNode类
     * @param k int整型
     * @return ListNode类
     */
    public ListNode FindKthToTail (ListNode pHead, int k) {
        // write code here
        if(k == 0 || pHead==null){
            return null;
        }

        ListNode new_head = pHead;
        while(k > 1){
            new_head = new_head.next;
            if(new_head ==null){
                return null;
            }
            k--;
        }
        while(new_head.next != null){
            pHead = pHead.next;
            new_head = new_head.next;
        }
        return pHead;
    }
}
```

大佬思路:

哥就是大佬思路（快慢指针）

```
class Solution {
public:
    ListNode* FindKthToTail(ListNode* pHead, int k) {
        ListNode* fast = pHead;
        ListNode* slow = pHead;
        //快指针先行k步
        for(int i = 0; i < k; i++){
            if(fast != NULL)
                fast = fast->next;
            //达不到k步说明链表过短，没有倒数k
            else
                return slow = NULL;
        }
        //快慢指针同步，快指针先到底，慢指针指向倒数第k个
        while(fast != NULL){
            fast = fast->next;
            slow = slow->next;
        }
        return slow;
    }
};
```

9. 删除链表的倒数第n个节点

题目描述：

给定一个链表，删除链表的倒数第 n 个节点并返回链表的头指针
例如，
给出的链表为: $1\to 2\to 3\to 4\to 5, n=2$.
删除了链表的倒数第 n 个节点之后,链表变为$1\to 2\to 3\to 5$.

数据范围： 链表长度 $0\le n\le 1000$，链表中任意节点的值满足 $0\le val\le 100$
要求：空间复杂度 $O(1)$，时间复杂度 $O(n)$
备注：
题目保证 n 一定是有效的

示例1

输入： {1,2},2 复制
返回值：{2} 复制

解题代码：

```java
public ListNode removeNthFromEnd (ListNode head, int n) {
    // write code here
    if(n == 0 || head == null){
        return head;
    }

    ListNode new_head = head;
    ListNode prev = null;
    ListNode current = head;
    while(n > 1){
        new_head = new_head.next;
        n--;
    }
    if(new_head.next == null){
        return head.next;
    }
    while(new_head.next != null){
        prev = current;
        current = current.next;
        new_head = new_head.next;
    }
    prev.next = current.next;
    return head;
}
```

大佬思路:

```
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        //添加表头
        ListNode* res = new ListNode(-1);
        res->next = head;
        //当前节点
        ListNode* cur = head;
        //前序节点
        ListNode* pre = res;
        ListNode* fast = head;
        //快指针先行n步
        while(n--)
            fast = fast->next;
        //快慢指针同步，快指针到达末尾，慢指针就到了倒数第n个位置
        while(fast != NULL){
            fast = fast->next;
            pre = cur;
            cur = cur->next;
        }
        //删除该位置的节点
        pre->next = cur->next;
        //返回去掉头
        return res->next;
    }
};
```

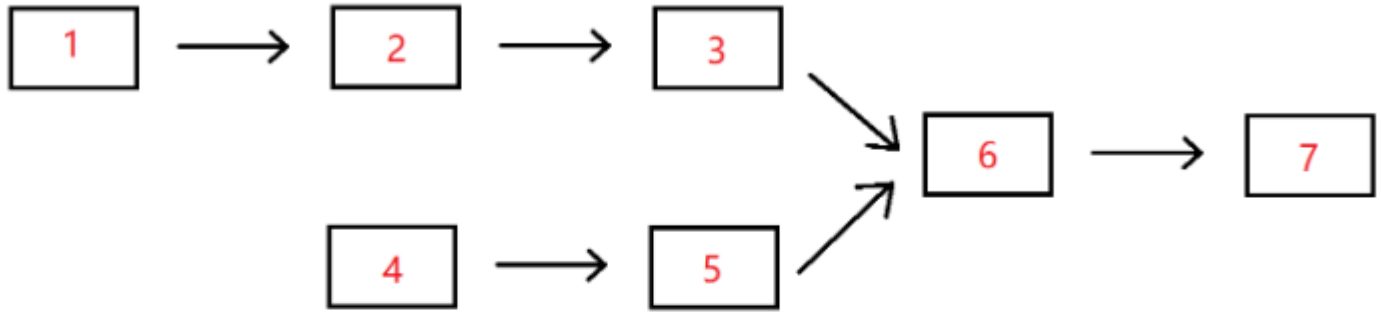
10. 两个链表的第一个公共结点

题目描述：

输入两个无环的单向链表，找出它们的第一个公共结点，如果没有公共节点则返回空。（注意因为传入数据是链表，所以错误测试数据的提示是用其他方式显示的，保证传入数据是正确的）

数据范围：$n \leq 1000$
要求：空间复杂度 $O(1)$，时间复杂度 $O(n)$

例如，输入{1,2,3},{4,5},{6,7}时，两个无环的单向链表的结构如下图所示：



可以看到它们的第一个公共结点的结点值为6，所以返回结点值为6的结点。

输入描述：

输入分为是3段，第一段是第一个链表的非公共部分，第二段是第二个链表的非公共部分，第三段是第一个链表和第二个链表的公共部分。 后台会将这3个参数组装为两个链表，并将这两个链表对应的头节点传入到函数FindFirstCommonNode里面，用户得到的输入只有pHead1和pHead2。

返回值描述：

返回传入的pHead1和pHead2的第一个公共结点，后台会打印以该节点为头节点的链表。

示例1

输入：`{1,2,3},{4,5},{6,7}` 复制

返回值：`{6,7}` 复制

说明：第一个参数`{1,2,3}`代表是第一个链表非公共部分，第二个参数`{4,5}`代表是第二个链表非公共部分，最后的`{6,7}`表示的是2个链表的公共部分
这3个参数最后在后台会组装成为2个两个无环的单链表，且是有公共节点的

示例2

输入：`{1},{2,3},{}` 复制

返回值：`{}` 复制

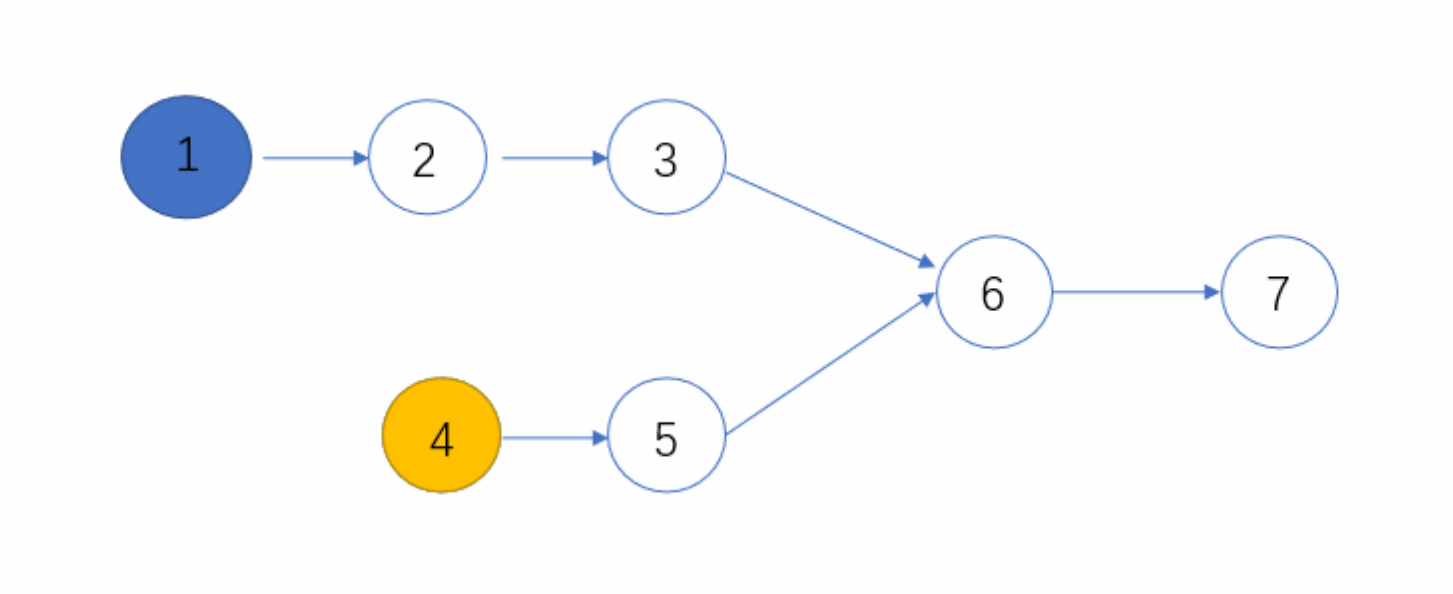
说明：2个链表没有公共节点 ,返回null，后台打印`{}`

解题代码:

```
public ListNode FindFirstCommonNode(ListNode pHead1, ListNode pHead2) {
    HashSet<ListNode> hashMap = new HashSet<>();
    while(pHead1!=null){
        hashMap.add(pHead1);
        pHead1 = pHead1.next;
    }
    while(pHead2!=null){
        if(hashMap.contains(pHead2)){
            return pHead2;
        }
        pHead2 = pHead2.next;
    }
    return null;
}
```

大佬思路:

因为两个指针，同样的速度，走完同样长度（链表1+链表2），不管两条链表有无相同节点，都能够到达同时到达终点。



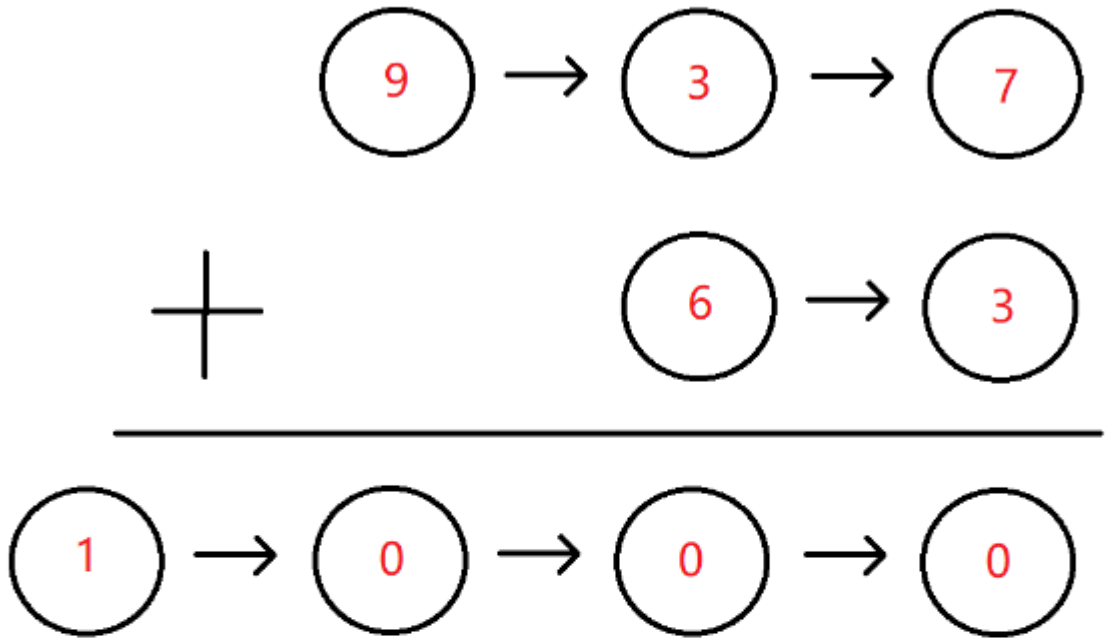
```
public ListNode FindFirstCommonNode(ListNode pHead1, ListNode pHead2) {
    ListNode l1 = pHead1, l2 = pHead2;
    while(l1 != l2){
        l1 = (l1==null)?pHead2:l1.next;
        l2 = (l2==null)?pHead1:l2.next;
    }
    return l1;
}
```

11. 链表相加(二)

题目描述：

假设链表中每一个节点的值都在 0 - 9 之间，那么链表整体就可以代表一个整数。
给定两个这种链表，请生成代表两个整数相加值的结果链表。
数据范围：$0 \leq n, m \leq 1000000$，链表任意值 $0 \leq val \leq 9$
要求：空间复杂度 $O(n)$，时间复杂度 $O(n)$

例如：链表 1 为 9->3->7，链表 2 为 6->3，最后生成新的结果链表为 1->0->0->0。



示例1

输入：[9,3,7],[6,3]

返回值：{1,0,0,0}

说明：如题面解释

示例2

输入：[0],[6,3]

返回值：{6,3}

备注:

$1 \leq n, m \leq 10^6$
$0 \leq a_i, b_i \leq 9$

解题代码:

```
public ListNode addInList (ListNode head1, ListNode head2) {
    // write code here
    ListNode Tmp1 = ReverseList(head1);
    ListNode Tmp2 = ReverseList(head2);

    ListNode head = new ListNode(0);
    ListNode current = head;
    int carry = 0;
    int value;
    while(Tmp1!=null && Tmp2!=null){
        value = (Tmp1.val+Tmp2.val + carry)%10;
        carry = (Tmp1.val+Tmp2.val + carry)/10;
        current.next = new ListNode(value);
        current = current.next;
        Tmp1 = Tmp1.next;
        Tmp2 = Tmp2.next;
    }
    while(Tmp1!=null){
        value = (Tmp1.val+ carry)%10;
        carry = (Tmp1.val+ carry)/10;
        current.next = new ListNode(value);
        current = current.next;
        Tmp1 = Tmp1.next;
    }
    while(Tmp2!=null){
        value = (Tmp2.val+ carry)%10;
        carry = (Tmp2.val+ carry)/10;
        current.next = new ListNode(value);
        current = current.next;
        Tmp2 = Tmp2.next;
    }
    if(carry == 1){
        current.next = new ListNode(1);
        current = current.next;
    }
    // head.next = null;
    return ReverseList(head.next);
}
```

大佬思路:

我的方法是先反转链表，然后相加，最后再反转回来，这样做的话，空间复杂度为O(N),时间复杂度为O(N),但是我们可以不用反转链表，直接使用栈，这样空间复杂度为O(1),时间复杂度为O(N).

```java
public class Solution {
    /**
     *
     * @param head1 ListNode类
     * @param head2 ListNode类
     * @return ListNode类
     */
    public ListNode addInList (ListNode head1, ListNode head2) {
        // write code here
        if(head1 == null)
            return head2;
        if(head2 == null){
            return head1;
        }
        // 使用两个辅助栈，利用栈先进后出，相当于反转了链表
        Stack<ListNode> stack1 = new Stack<>();
        Stack<ListNode> stack2 = new Stack<>();
        ListNode p1=head1;
        ListNode p2=head2;
        // 将两个链表的结点入栈
        while(p1!=null){
            stack1.push(p1);
            p1=p1.next;
        }
        while(p2!=null){
            stack2.push(p2);
            p2=p2.next;
        }
        // 进位
        int tmp = 0;
        // 创建新的链表头节点
        ListNode head = new ListNode(-1);
        ListNode nHead = head.next;
        while(!stack1.isEmpty()||!stack2.isEmpty()){
            // val用来累加此时的数值（加数+加数+上一位的进位=当前总的数值）
            int val = tmp;
            // 栈1不为空的时候，弹出结点并累加值
            if (!stack1.isEmpty()) {
                val += stack1.pop().val;
            }
            // 栈2不为空的时候，弹出结点并累加值
            if (!stack2.isEmpty()) {
                val += stack2.pop().val;
```

```
            }
            // 求出进位
            tmp = val/10;
            // 进位后剩下的数值即为当前节点的数值
            ListNode node = new ListNode(val%10);
            // 将结点插在头部
            node.next = nHead;
            nHead = node;
        }
        if(tmp > 0){
            // 头插
            ListNode node = new ListNode(tmp);
            node.next = nHead;
            nHead = node;
        }
        return nHead;
    }
}
```

12. 单链表的排序

题目描述：

描述

给定一个节点数为n的无序单链表，对其按升序排序。

数据范围：$0 < n \leq 100000$，保证节点权值在$[-10^9, 10^9]$之内。
要求：空间复杂度 $O(n)$，时间复杂度 $O(nlogn)$

示例1

输入：[1,3,2,4,5]

返回值：{1,2,3,4,5}

示例2

输入：[-1,0,-2]

返回值：{-2,-1,0}

解题代码:

运行超时了，但是思路是对的

```
public ListNode sortInList (ListNode head) {
    // write code here
    if(head == null){
        return head;
    }
    ListNode new_head = new ListNode(head.val);
    head = head.next;
    ListNode nextNode;
    while(head!=null){
        nextNode = new ListNode(head.val);
        new_head = Merge(new_head,nextNode);
        head=head.next;
    }
    return new_head;
}
```

大佬思路:

```java
import java.util.*;

/*
 * public class ListNode {
 *   int val;
 *   ListNode next = null;
 * }
 */

public class Solution {
    /**
     *
     * @param head ListNode类 the head node
     * @return ListNode类
     */
    public ListNode sortInList (ListNode head) {
        // write code here
        if(head ==null || head.next ==null){
            return head;
        }
        ListNode left = head;
        ListNode mid = head.next;
        ListNode right = head.next.next;

        while(right!=null && right.next !=null){
            left = left.next;
            mid = mid.next;
            right = right.next.next;
        }
        left.next = null;
        return merge(sortInList(head),sortInList(mid));
    }
    //两个有序链表排序
    public ListNode merge(ListNode head1,ListNode head2){
        if(head1==null){
            return head2;
        }
        if(head2==null){
            return head1;
        }
        ListNode res = new ListNode(0);
        ListNode cur = res;
        while(head1!=null && head2!=null){
```

```
            if(head1.val<head2.val){
                cur.next = head1;
                head1 = head1.next;
            }else{
                cur.next = head2;
                head2 = head2.next;
            }
            cur = cur.next;
        }
        if(head1!=null){
            cur.next = head1;
        }
        if(head2!=null){
            cur.next = head2;
        }
        return res.next;
    }
}
```

13. 判断一个链表是否为回文结构

题目描述：

给定一个链表，请判断该链表是否为回文结构。
回文是指该字符串正序逆序完全一致。
数据范围：链表节点数 $0 \leq n \leq 10^5$，链表中每个节点的值满足 $|val| \leq 10^7$

示例1

输入：{1} 复制

返回值：true 复制

示例2

输入：{2,1} 复制

返回值：false 复制

说明：2->1

示例3

输入：{1,2,2,1} 复制

返回值：true 复制

说明：1->2->2->1

解题代码:

```
public boolean isPail (ListNode head) {
    // write code here
    if(head == null || head.next == null){
        return true;
    }
    ListNode new_head = new ListNode(head.val);
    ListNode current = head.next;
    ListNode nextNode;
    while(current!=null){
        nextNode = new ListNode(current.val);
        nextNode.next = new_head;
        new_head = nextNode;
        current = current.next;
    }
    while(head!=null){
        if(head.val != new_head.val){
            return false;
        }
        head = head.next;
        new_head = new_head.next;
    }
    return true;
}
```

大佬思路:

```java
import java.util.*;

/*
 * public class ListNode {
 *   int val;
 *   ListNode next = null;
 * }
 */

public class Solution {
    /**
     *
     * @param head ListNode类 the head
     * @return bool布尔型
     */
    public boolean isPail (ListNode head) {
        // write code here
        //快慢指针找到中点 -> 反转后半部分 -> 双指针同时遍历
        if(head == null || head.next == null) return true;
        ListNode slow = head, fast = head.next;
        while(fast != null && fast.next != null){
            slow = slow.next;
            fast = fast.next.next;
        }
        //反转slow结点往后的部分
        fast = slow.next;
        while(fast != null && fast.next != null){
            ListNode next = fast.next;
            fast.next = next.next;
            next.next = slow.next;
            slow.next = next;
        }
        fast = slow.next;
        slow = head;
        while(fast != null){
            if(slow.val != fast.val) return false;
            slow = slow.next;
            fast = fast.next;
        }
        return true;
    }
}
```

14. 链表的奇偶重排

题目描述：

给定一个单链表，请设定一个函数，将链表的奇数位节点和偶数位节点分别放在一起，重排后输出。
注意是节点的编号而非节点的数值。

数据范围：节点数量满足 $0 \leq n \leq 10^5$，节点中的值都满足 $0 \leq val \leq 1000$
要求：空间复杂度 $O(n)$，时间复杂度 $O(n)$

示例1

输入：{1,2,3,4,5,6}

返回值：{1,3,5,2,4,6}

说明：1->2->3->4->5->6->NULL
重排后为
1->3->5->2->4->6->NULL

示例2

输入：{1,4,6,3,7}

返回值：{1,6,7,4,3}

说明：1->4->6->3->7->NULL
重排后为
1->6->7->4->3->NULL
奇数位节点有1,6,7，偶数位节点有4,3。重排后为1,6,7,4,3

解题代码：

```
public ListNode oddEvenList (ListNode head) {
    // write code here
    if(head == null || head.next == null){
        return head;
    }
    ListNode odd = head;
    ListNode even = head.next;
    ListNode even_head = even;
    while(even!=null && even.next!=null){
        odd.next = even.next;
        odd = odd.next;
        even.next = odd.next;
        even = even.next;
    }
    odd.next = even_head;
    return head;
}
```

已经是大佬思路了

15.删除有序链表中重复的元素-I

题目描述：

删除给出链表中的重复元素（链表中元素从小到大有序），使链表中的所有元素都只出现一次
例如：
给出的链表为$1 \to 1 \to 2$,返回$1 \to 2$.
给出的链表为$1 \to 1 \to 2 \to 3 \to 3$,返回$1 \to 2 \to 3$.

数据范围：链表长度满足$0 \le n \le 100$，链表中任意节点的值满足$|val| \le 100$
进阶：空间复杂度$O(1)$，时间复杂度$O(n)$

示例1

输入：{1,1,2}

返回值：{1,2}

示例2

输入：{}

返回值：{}

解题代码:

```java
public ListNode deleteDuplicates (ListNode head) {
    // write code here
    if(head == null || head.next == null){
        return head;
    }
    int val = head.val;
    ListNode cur = head;
    ListNode tail = head;
    while(cur.next!=null){
        cur = cur.next;
        if(cur.val == val){
            continue;
        }
        tail.next = cur;
        tail = cur;
        val = cur.val;
    }
    if(tail.val == cur.val){
        tail.next = null;
    }
    return head;
}
```

大佬思路:

```
public ListNode deleteDuplicates (ListNode head) {
    // write code here
    if (head == null || head.next == null){
        return head;
    }
    ListNode q = head;
    while(q != null) {
        int cache = q.val;
        ListNode p = q.next;
        while(p != null && p.val == cache) {
            p = p.next;
        }
        q.next = p;
        q = p;
    }
    return head;
}
```

16.删除有序链表中重复的元素-II

题目描述：

给出一个升序排序的链表，删除链表中的所有重复出现的元素，只保留原链表中只出现一次的元素。
例如：
给出的链表为$1 \to 2 \to 3 \to 3 \to 4 \to 4 \to 5$, 返回$1 \to 2 \to 5$.
给出的链表为$1 \to 1 \to 1 \to 2 \to 3$, 返回$2 \to 3$.

数据范围：链表长度 $0 \le n \le 10000$，链表中的值满足 $|val| \le 1000$
要求：空间复杂度 $O(n)$，时间复杂度 $O(n)$
进阶：空间复杂度 $O(1)$，时间复杂度 $O(n)$

示例1

输入：{1,2,2}

返回值：{1}

示例2

输入：{}

返回值：{}

解题代码:

```
public ListNode deleteDuplicates (ListNode head) {
    // write code here
    if (head == null || head.next == null){
        return head;
    }
    ListNode new_head = new ListNode(0);
    new_head.next = head;
    ListNode cur = new_head;
    while(head != null) {
        int i = 0;
        int cache = head.val;
        ListNode p = head.next;
        while(p != null && p.val == cache) {
            i++;
            p = p.next;
        }
        if(i == 0){
            cur.next = head;
            cur = cur.next;
            head = head.next;
            cur.next = null;
        }else{
            head = p;
        }
    }
    if(new_head.next.next!=null && new_head.next.val == new_head.next.next.val){
        new_head.next = null;
    }
    return new_head.next;
}
```

大佬思路:

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