- ## 143. Reorder List

Hint:
1.修改完ListNode后要将next修改为空
2.以arraylist记录每一个node的信息

```
class Solution {
    public void reorderList(ListNode head) {
        if(head == null || head.next == null)
            return;
        List<ListNode> list = new ArrayList<>();
        while(head != null){
            list.add(head);
            head = head.next;
        }
        int start = 0;
        int end = list.size() - 1;
        ListNode first = list.get(start);
        ListNode second = list.get(end);
        ListNode last = null;
        while(start < end){
            first = list.get(start);
            second = list.get(end);
            first.next = second;
            second.next = null;
            if(last != null){
                last.next = first;
            }
            last = second;
            start++;
            end--;
        }
        if(start == end ){
            last.next = list.get(start);
            last.next.next = null;
        }
    }
}
```

## 19. Remove Nth Node From End of List

**Hint：**
**1.用map记录每一个node的index**
**2.移除的node next要修改为空**

```
class Solution {
    public ListNode removeNthFromEnd(ListNode head, int n) {
        Map<Integer, ListNode> map = new HashMap<>();
        int max = 0;
        int index = 1;
        ListNode cur = head;
        while(cur != null){
            map.put(index, cur);
            cur = cur.next;
            if(index > max){
                max = index;
            }
            index++;
        }
        int target = max - n + 1;
        if(target == 1){
            cur = head.next;
            head.next = null;
            return cur;
        }
        map.get(target - 1).next = map.get(target + 1);
        map.get(target).next = null;
        return head;
    }
}
```

- ## 430. Flatten a Multilevel Doubly Linked List

Hint:
1.Recursion
2.需要先判断原node的next是否为null，因为需要修改node.next.prev，否则会造成null pointer
**3.Recursion返回当前linkedlist的最后一个node**

```
class Solution {
    public Node flatten(Node head) {
        if(head != null){
            dfs(head);
        }
        return head;
    }
    private Node dfs(Node head){
        while(head != null ){
            if(head.child != null){
                Node newnext = head.next;
                Node temp = dfs(head.child);
                head.next = head.child;
                head.next.prev = head;
                head.child = null;
                temp.next = newnext;
                if(newnext != null){
                    newnext.prev = temp;
                }
            }
            if(head.next == null){
                return head;
            }
            head = head.next;
        }
        return head;
    }
}
```