null，则返回旧的头节点。

全部过程请参看如下代码中的 reversePart 方法。

```
public Node reversePart(Node head, int from, int to) {
        int len = 0;
        Node node1 = head;
        Node fPre = null;
        Node tPos = null;
        while (node1 != null) {
                len++;
                fPre = len == from - 1 ? node1 : fPre;
                tPos = len == to + 1 ? node1 : tPos;
                node1 = node1.next;
        }
        if (from > to || from < 1 || to > len) {
                return head;
        }
        node1 = fPre == null ? head : fPre.next;
        Node node2 = node1.next;
        node1.next = tPos;
        Node next = null;
        while (node2 != tPos) {
                next = node2.next;
                node2.next = node1;
                node1 = node2;
                node2 = next;
        }
        if (fPre != null) {
                fPre.next = node1;
                return head;
        }
        return node1;
}
```

环形单链表的约瑟夫问题

【题目】

据说著名犹太历史学家 Josephus 有过以下故事：在罗马人占领乔塔帕特后，39 个犹太人与 Josephus 及他的朋友躲到一个洞中，39 个犹太人决定宁愿死也不要被敌人抓到，于是决定了一个自杀方式，41 个人排成一个圆圈，由第 1 个人开始报数，报数到 3 的人就自杀，然后再由下一个人重新报 1，报数到 3 的人再自杀，这样依次下去，直到剩下最后一个人时，那个人可以自由选择自己的命运。这就是著名的约瑟夫问题。现在请用单向环形链表描述该结构并呈现整个自杀过程。

输入：一个环形单向链表的头节点 head 和报数的值 m。

返回：最后生存下来的节点，且这个节点自己组成环形单向链表，其他节点都删掉。

进阶：

如果链表节点数为 N，想在时间复杂度为 $O(N)$时完成原问题的要求，该怎么实现？

【难度】

原问题：士 ★☆☆☆

进阶：校 ★★★☆

【解答】

先来看看普通解法是如何实现的，其实非常简单，方法如下：

1. 如果链表为空或者链表节点数为 1，或者 m 的值小于 1，则不用调整就直接返回。
2. 在环形链表中遍历每个节点，不断转圈，不断让每个节点报数。
3. 当报数到达 m 时，就删除当前报数的节点。
4. 删除节点后，别忘了还要把剩下的节点继续连成环状，继续转圈报数，继续删除。
5. 不停地删除，直到环形链表中只剩一个节点，过程结束。

普通的解法就像题目描述的过程一样，具体实现请参看如下代码中的 josephusKill1 方法。

```
public class Node {
	public int value;
	public Node next;

	public Node(int data) {
		this.value = data;
	}
}

public Node josephusKill1(Node head, int m) {
	if (head == null || head.next == head || m < 1) {
		return head;
	}
	Node last = head;
	while (last.next != head) {
		last = last.next;
	}
	int count = 0;
	while (head != last) {
		if (++count == m) {
			last.next = head.next;
			count = 0;
		} else {
			last = last.next;
```

```
            }
            head = last.next;
        }
        return head;
    }
```

普通的解法在实现上不难，就是考查面试者基本的代码实现技巧，做到不出错即可。很明显的是，每删除掉一个节点，都需要遍历 m 次，一共需要删除的节点数为 n-1，所以普通解法的时间复杂度为 $O(n \times m)$，这明显是不符合进阶要求的。

下面介绍进阶的解法。原问题之所以花费的时间多，是因为我们一开始不知道到底哪一个节点最后会活下来。所以依靠不断地删除来淘汰节点，当只剩下一个节点的时候，才知道是这个节点。如果不通过一直删除方式，有没有办法直接确定最后活下来的节点是哪一个呢？这就是进阶解法的实质。

举个例子，环形链表为：1->2->3->4->5->1，这个链表节点数为 n=5，m=3。

通过不断删除的方式，最后节点 4 会活下来。但我们可以不用一直删除的方式，而是用进阶的方法，根据 n 与 m 的值，直接算出是第 4 个节点最终会活下来，接下来找到节点 4 即可。

那到底怎么直接算出来呢？首先，如果环形链表节点数为 n，我们做如下定义：从这个环形链表的头节点开始编号，头节点编号为 1，头节点的下一个节点编号为 2，……，最后一个节点编号为 n。然后考虑如下问题：

最后只剩下一个节点，这个幸存节点在只由自己组成的环中编号为 1，记为 Num(1) = 1；

在由两个节点组成的环中，这个幸存节点的编号是多少呢？假设编号是 Num(2)；

……

在由 i-1 个节点组成的环中，这个幸存节点的编号是多少呢？假设编号是 Num(i-1)；

在由 i 个节点组成的环中，这个幸存节点的编号是多少呢？假设编号是 Num(i)；

……

在由 n 个节点组成的环中，这个幸存节点的编号是多少呢？假设编号是 Num(n)。

我们已经知道 Num(1) = 1，如果再确定 Num(i-1)和 Num(i)到底是什么关系，就可以通过递归过程求出 Num(n)。Num(i-1)和 Num(i)的关系分析如下：

1．假设现在圈中一共有 i 个节点，从头节点开始报数，报 1 的是编号 1 的节点，报 2 的是编号 2 的节点，假设报 A 的是编号 B 的节点，则 A 和 B 的对应关系如下。

A	B
1	1

2	2
...	...
i	i
i+1	1
i+2	2
...	...
2i	i
2i+1	1
2i+2	2
…	…

举个例子，环形链表有 3 个节点，报 1 的是编号 1，报 2 的是编号 2，报 3 的是编号 3，报 4 的是编号 1，报 5 的是编号 2，报 6 的是编号 3，报 7 的是编号 1，报 8 的是编号 2，报 9 的是编号 3，报 10 的是编号 1……

如上 A 和 B 的关系用数学表达式来表示可以写成：B=(A-1)%i+1。这个表达式不一定是唯一的，读者只要能写出准确概括 A 和 B 关系的式子就可以。总之，要找到报数（A）和编号节点（B）之间的关系。

2．如果编号为 s 的节点被删除，环的节点数自然从 i 变成了 i-1。那么原来在大小为 i 的环中，每个节点的编号会发生什么变化呢？变化如下：

环大小为 i 的每个节点编号	删掉编号 s 的节点后，环大小为 i-1 的每个节点编号
...	…
s-2	i-2
s-1	i-1
s	—（无编号是因为被删掉了）
s+1	1
s+2	2
...	…

新的环只有 i-1 个节点，因为有一个节点已经删掉。编号为 s 的节点往后，编号为 s+1、s+2、s+3 的节点就变成了新环中的编号为 1、2、3 的节点；编号为 s 的节点的前一个节点，也就是编号 s-1 的节点，就成了新环中的最后一个节点，也就是编号为 i-1 的节点。

假设环大小为 i 的节点编号记为 old，环大小为 i-1 的每个节点编号记为 new，则 old 与 new 关系的数学表达式为：old=(new+s-1)%i+1。表达式同样不止一种，写出一种满足的

即可。

3．因为每次都是报数到 m 的节点被杀，所以根据步骤 1 的表达式 B=(A-1)%i+1，A=m。被杀的节点编号为(m-1)%i+1，即 s=(m-1)%i+1，带入到步骤 2 的表达式 old=(new+s-1)%i+1 中，经过化简为 old=(new+m-1)%i+1。至此，我们终于得到了 Num(i-1)—new 和 Num(i)—old 的关系，且这个关系只和 m 与 i 的值有关。

整个进阶解法的过程总结为：

1．遍历链表，求链表的节点个数记为 n，时间复杂度为 $O(N)$。

2．根据 n 和 m 的值，还有上文分析的 Num(i-1)和 Num(i)的关系，递归求生存节点的编号；这一步的具体过程请参看如下代码中的 getLive 方法，getLive 方法为单决策的递归函数，且递归为 N 层，所以时间复杂度为 $O(N)$。

3．最后根据生存节点的编号，遍历链表找到该节点，时间复杂度为 $O(N)$。

4．整个过程结束，总的时间复杂度为 $O(N)$。

进阶解法的全部过程请参看如下代码中的 josephusKill2 方法。

```
public Node josephusKill2(Node head, int m) {
	if (head == null || head.next == head || m < 1) {
		return head;
	}
	Node cur = head.next;
	int tmp = 1; // tmp -> list size
	while (cur != head) {
		tmp++;
		cur = cur.next;
	}
	tmp = getLive(tmp, m); // tmp -> service node position
	while (--tmp != 0) {
		head = head.next;
	}
	head.next = head;
	return head;
}

public int getLive(int i, int m) {
	if (i == 1) {
		return 1;
	}
	return (getLive(i - 1, m) + m - 1) % i + 1;
}
```