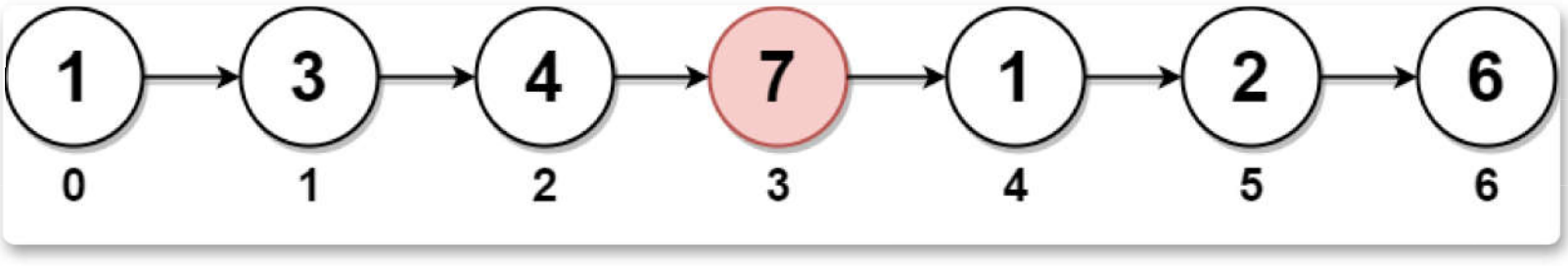
> PS：这道题其实挺有意思，面试中也遇见过，第一次碰见这个题的时候还真没什么好思路，直到我遇见了快慢指针，yyds。

题目介绍[1]

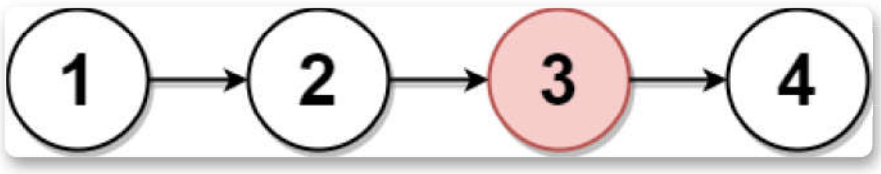
给你一个链表的头节点 head 。删除链表的中间节点 ，并返回修改后的链表的头节点 head 。 长度为 n 链表的中间节点是从头数起第 ⌊n / 2⌋ 个节点（下标从 0 开始），其中 ⌊x⌋ 表示小于或等于 x 的最大整数。 对于 n = 1、2、3、4 和 5 的情况，中间节点的下标分别是 0、1、1、2 和 2 。

示例 1



> 输入：head = [1,3,4,7,1,2,6]
>
> 输出：[1,3,4,1,2,6]
>
> 解释： 上图表示给出的链表。节点的下标分别标注在每个节点的下方。由于 n = 7 ， 值为 7 的节点 3 是中间节点，用红色标注。返回结果为移除节点后的新链表。

示例 2



> 输入：head = [1,2,3,4]
>
> 输出：[1,2,4]
>
> 解释： 上图表示给出的链表。对于 n = 4 ，值为 3 的节点 2 是中间节点，用红色标注。

题目解答

方法一：快慢指针

思路和算法

大家可以先看一道相似题目，**链表的中间结点(快慢指针)**，做完这道题目之后，我们就有解答本道题的思路了。 这道题目要求的是需要删除链表的中间节点，而不是获取链表的中间节点。

所以我们需要在链表的前面加一个 dummp 节点，dummp 节点也称之为哨兵节点。 这个节点看似没有任何用处，其实用处非常之大，可以省去很多操作。加上这个 dummp 节点之后，慢指针 slow 指向 dummp 节点，快指针 fast 指向原链表头节点，这样我们获取到的中间节点就是我们要删除的节点的前一个节点了。

代码实现

```
class Solution {
    public ListNode deleteMiddle(ListNode head) {
        if (head == null || head.next == null) {
            return null;
        }
        ListNode middle = middleNode(head);
        middle.next = middle.next.next;
        return head;
    }
    public ListNode middleNode(ListNode head) {
        ListNode dummp = new ListNode(0, head);
        ListNode slow = dummp;
        ListNode fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }
}
```

复杂度分析

- 时间复杂度：O(n)，fast 指针需要遍历一次链表，因此最多走 n 步；
- 空间复杂度：O(1)。

参考资料

[1] **原题链接:** https://leetcode-cn.com/problems/delete-the-middle-node-of-a-linked-list/