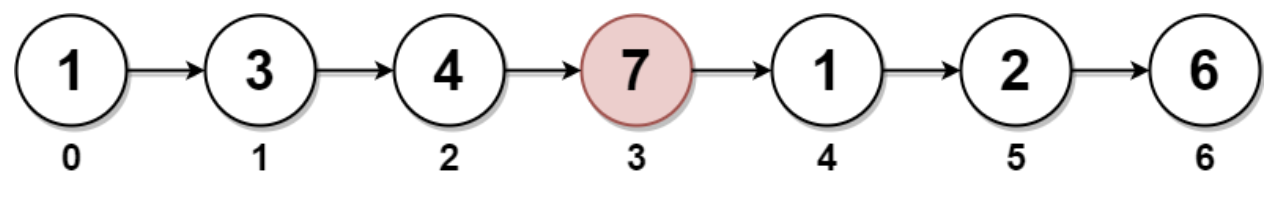
# 题目

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]

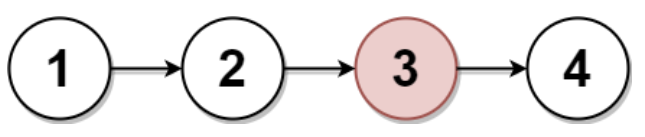
解释：

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]

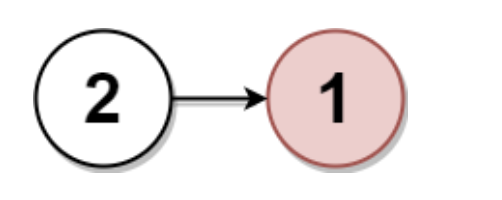
输出：[1,2,4]

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。

示例 3：



输入：head = [2,1]

输出：[2]

解释：

上图表示给出的链表。

对于 n = 2 ，值为 1 的节点 1 是中间节点，用红色标注。

值为 2 的节点 0 是移除节点 1 后剩下的唯一一个节点。

提示：

链表中节点的数目在范围 [1, 105] 内

1 <= Node.val <= 105

# 分析

## 方法一：双指针/快慢指针

分析：

由于链表不支持随机访问，因此常见的找出链表中间节点的方法是使用快慢指针：即我们使用两个指针fast和slow对链表进行遍历，其中快指针fast每次遍历两个元素，慢指针slow每次遍历一个元素。这样在快指针遍历完链表时，慢指针就恰好在链表的中间位置。

在本题中，我们还需要删除链表的中间节点，因此除了慢指针slow外，我们再使用一个指针pre时刻指向slow的前一个节点。这样我们就可以在遍历结束后，通过pre将slow删除了。

细节

当链表中只有一个节点时，我们会删除这个节点并返回空链表。但这个节点不存在前一个节点，即pre是没有意义的，因此对于这种情况，我们可以在遍历前进行特殊判断，直接返回空指针作为答案。

说明：对于链表的下标访问，比如中间节点，倒序以及间隔的节点，这种有两个方法：

1. 数组，存储节点然后按照数组下标访问
2. 快慢指针，这种一般适用于中间节点等

代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode() : val(0), next(nullptr) {}

\* ListNode(int x) : val(x), next(nullptr) {}

\* ListNode(int x, ListNode \*next) : val(x), next(next) {}

\* };

\*/

class Solution {

public:

ListNode\* deleteMiddle(ListNode\* head) {

if (head->next == nullptr) return nullptr;

ListNode \*slow = head;

ListNode \*fast = head;

ListNode \*prev = nullptr;

while(fast && fast->next) {

fast = fast->next->next;

prev = slow;

slow = slow->next;

}

prev->next = prev->next->next;

return head;

}

};

复杂度分析：

时间复杂度：O(n)。

空间复杂度：O(1)。