环形链表 II

给定一个链表，返回链表开始入环的第一个节点。 如果链表无环，则返回 `null`。

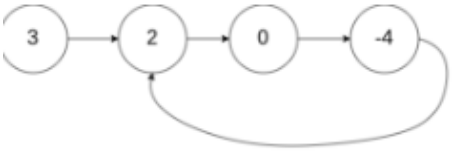
为了表示给定链表中的环，我们使用整数 `pos` 来表示链表尾连接到链表中的位置（索引从 0 开始）。 如果 `pos` 是 `-1`，则在该链表中没有环。**注意，pos 仅仅是用于标识环的情况，并不会作为参数传递到函数中。**

说明：不允许修改给定的链表。

进阶：

- 你是否可以使用 `O(1)` 空间解决此题？

示例 1：

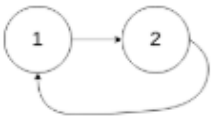


输入：`head = [3,2,0,-4], pos = 1`

输出：返回索引为 `1` 的链表节点

解释：链表中有一个环，其尾部连接到第二个节点。

示例 2：



输入：`head = [1,2], pos = 0`

输出：返回索引为 `0` 的链表节点

解释：链表中有一个环，其尾部连接到第一个节点。

示例 3：

输入：`head = [1], pos = -1`

输出：返回 null

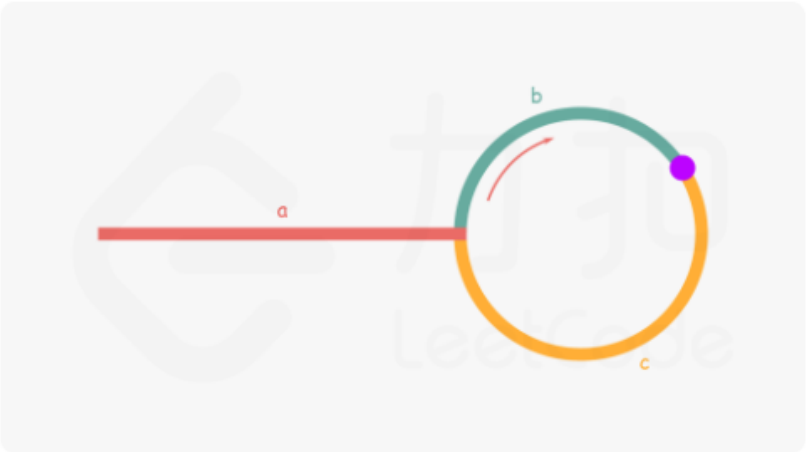
解释：链表中没有环。

快慢指针

思路与算法

我们使用两个指针，*fast* 与 *slow*。它们起始都位于链表的头部。随后，*slow* 指针每次向后移动一个位置，而 *fast* 指针向后移动两个位置。如果链表中存在环，则 *fast* 指针最终将再次与 *slow* 指针在环中相遇。

如下图所示，设链表中环外部分的长度为 a。*slow* 指针进入环后，又走了 b 的距离与 *fast* 相遇。此时，*fast* 指针已经走完了环的 n 圈，因此它走过的总距离为 $a+n(b+c)+b=a+(n+1)b+nc$。



根据题意，任意时刻，*fast* 指针走过的距离都为 *slow* 指针的 2 倍。因此，我们有

$$a+(n+1)b+nc=2(a+b) \implies a=c+(n-1)(b+c)$$

有了 $a=c+(n-1)(b+c)$ 的等量关系，我们会发现：从相遇点到入环点的距离加上 $n-1$ 圈的环长，恰好等于从链表头部到入环点的距离。

因此，当发现 *slow* 与 *fast* 相遇时，我们再额外使用一个指针 *ptr*。起始，它指向链表头部；随后，它和 *slow* 每次向后移动一个位置。最终，它们会在入环点相遇。

代码

C++ Java C JavaScript Golang

```c
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode *detectCycle(struct ListNode *head) {
    if(head==NULL)
    {
        return NULL;
    }
    int flag=0;
    struct ListNode *slow=head;
    struct ListNode *fast=head;
```

```
    while(fast!=NULL&&fast->next!=NULL)
    {
        slow=slow->next;
        fast=fast->next->next;
        if(fast==slow)
        {
            struct ListNode *ptr=head;
            while(ptr!=slow)
            {
                ptr=ptr->next;
                slow=slow->next;
            }
            return slow;
        }
    }
    return NULL;
}
```