@TOC

环形链表

141.环形链表

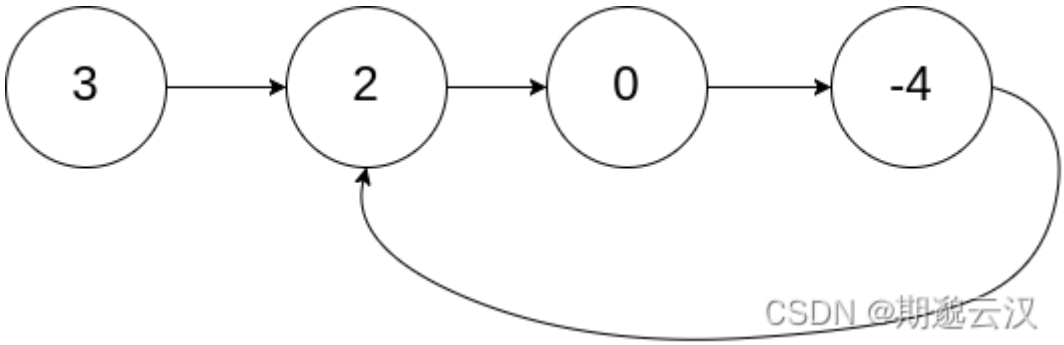
问题引入：

给你一个链表的头节点 head ，判断链表中是否有环。

如果链表中有某个节点，可以通过连续跟踪 next 指针再次到达，则链表中存在环。 为了表示给定链表中的环，评测系统内部使用整数 pos 来表示链表尾连接到链表中的位置（索引从 0 开始）。注意：pos 不作为参数进行传递 。仅仅是为了标识链表的实际情况。

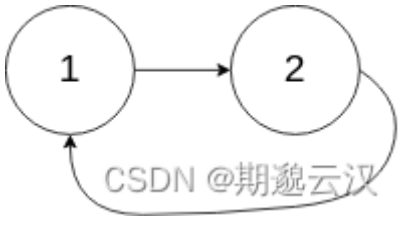
如果链表中存在环 ，则返回 true 。 否则，返回 false 。

- **示例1**：




```
输入：head = [3,2,0,-4], pos = 1
输出：true
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：true
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：false
解释：链表中没有环。
```

考虑快慢指针

快慢指针的思想就是引入两个节点指针flow和fast指针，在步长上慢指针flow一次走一个节点，而快指针一次可以走两个节点，或者更多。

即表现为：

```
flow = flow->next;
fast = fast->next->next;  //步长为慢指针的2倍
```

快慢指针在控制遍历单链表时很有用处，例如：876.链表的中间节点，找到这个中间节点时非常方便。

考虑解法

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
bool hasCycle(struct ListNode *head) {
    struct ListNode * flow = head;
    struct ListNode * fast = head;

    while(fast && fast->next){
        flow = flow->next;
        fast = fast->next->next;
        if(fast == flow){
            return true;
        }
    }
    return false;
}
```

其实将问题转化为追及相遇问题。

- **考虑若没有环路的时候，就是普通的单链表，那么当到达while循环结束条件时，那么肯定最终会返回false，即当前链表不存在环路。**
- **考虑快慢指针的追赶问题，flow走一步，fast走两步结果会不会相遇?**

  flow走一步，fast走3步结果会不会相遇?

  flow走一步，fast走4步结果会不会相遇?

  ……

- 使用快慢指针，当快指针进入环里的时候，相当于这要flow慢指针一旦进入环，快指针将会开始追赶，其实根据分析，flow走1步，fast走2步这样的步长是肯定可以追上的。考虑若慢指针flow刚入

环，环中有N个元素，那么每一次走步，快慢指针的间距就是 N + 1 - 2 = N - 1，最终N必然为0，两者相遇，说明有环存在。

- 当追上时，那肯定就是有环了，当退出while循环了，说明当前链表不存在环路。
- 其实不难考虑，flow走一步，fast走3步结果会不会相遇?

  flow走一步，fast走4步结果会不会相遇?

相当于是N + 1 - 3 = N - 2和N + 1 - 4 = N - 3，那么最终N能否为0，取决于N的大小

考虑当N为偶数时，那么fast一次走3步那么两者也是可以相遇。考虑N为奇数时，N最后变为N - 1，说明fast把flow反超了1位，那么设环的长度为C，即C个节点，此时两者再追赶，那么fast追上flow要相距C - 1长度，那么此时重新考虑，只要C - 1是偶数，那么就能追上；如果C - 1是奇数，显然永远追不上。因为每一次都错开了1位

考虑fast走4步，那么每次两者相距3位，那么，每次距离缩减3，若N不是3的倍数，那么情况最后会变为-1 或 -2，情况和走3步是类似的。

寻找pos环的入点

L：head到环的入点的长度
C：环的长度
X：flow慢指针的相距环入点的距离

两者相遇时快指针是慢指针路程的二倍

L + N*C + X = 2（L + X）

可得 N*C = L + X

L = N*C - X
L = （N - 1）C + `C - X`

分析代码：

```
bool hasCycle(struct ListNode *head) {
    struct ListNode * flow = head;
    struct ListNode * fast = head;

    while(fast && fast->next){
        flow = flow->next;
        fast = fast->next->next;
        //一个指针从相遇点开始走，一个指针从haed开始走，两者会在入口点相遇
        if(fast == flow){
            struct ListNode * meet = flow;
            while(head != meet){
                head = head->next;
                meet = meet->next;
            }
            return meet;
        }
    }
    return false;
}
```