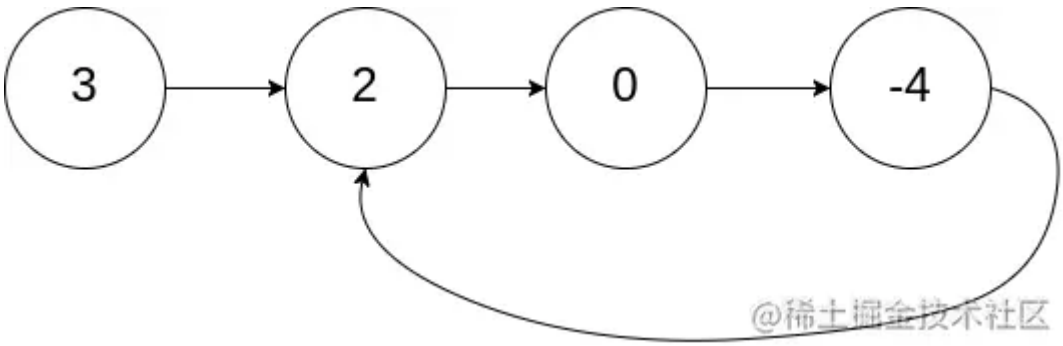
环形链表是链表中的一类特殊问题，它和链表反转一样，有着相对恒定的解题思路和适当的变体。如果你对它的特性和解法没有预先的了解和把握，那么前期的推导可能会花去你大量的时间。反过来看，只要我们能够掌握其核心思路，那么不管它怎么变化，大家都能在瞬间找到解题的“抓手”、进而给出正确的解答。

环形链表基本问题——如何判断链表是否成环?

> 真题描述：给定一个链表，判断链表中是否有环。

> 示例 1：

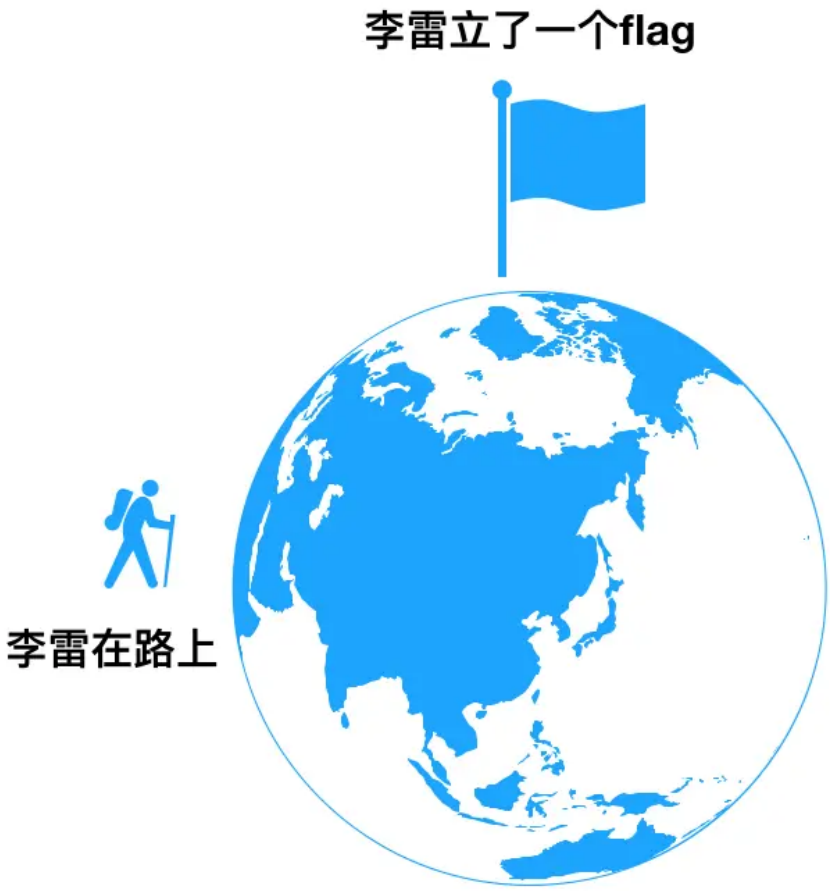
输入：[3,2,0,4]（链表结构如下图） 输出：true
解释：链表中存在一个环



思路解读

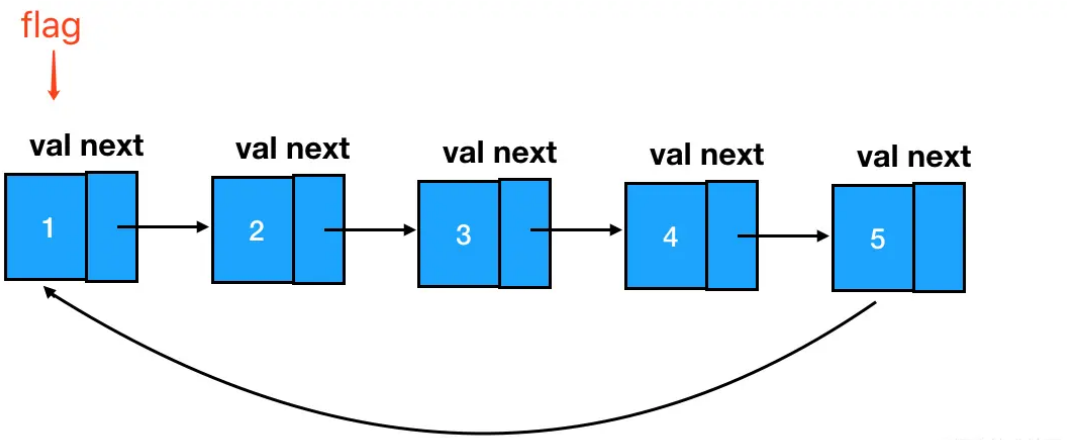
其实链表成环的特征非常明显，大家可以结合一个现实中的例子来理解：
假如现实中有一个长跑爱好者李雷，这货很狂，他立了一个 flag，说要徒步环游世界：

地球的周长围出来的这个圆，它就是一个“环”。李雷现在就想围着这个环跑上一圈，说他狂，他也没那么狂——他觉得自己最多跑一圈，为了防止自己跑过界，他决定在出发的地方立一个 flag：



这样，不管李雷走完这个环用了多少年，世事如何变迁，只要他的 flag 还没有倒，那么李雷就一定能回到自己梦开始的地方：）。

回到链表的世界里，也是一个道理。一个环形链表的基本修养，是能够让遍历它的游标回到原点：



从 flag 出发，只要我能够再回到 flag 处，那么就意味着，我正在遍历一个环形链表。

我们按照这个思路来做题：

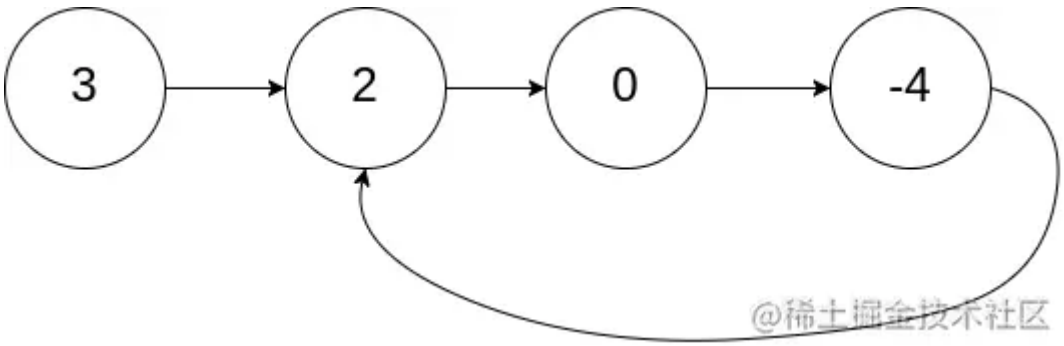
编码实现

```js
/**
 * @param {ListNode} head
 * @return {boolean}
 */
// 入参是头结点
const hasCycle = function(head) {
    // 只要结点存在，那么就继续遍历
    while(head){
        // 如果 flag 已经立过了，那么说明环存在
        if(head.flag){
            return true;
        }else{
            // 如果 flag 没立过，就立一个 flag 再往
            下走
            head.flag = true;
            head = head.next;
        }
    }
    return false;
};
```

> 真题描述：给定一个链表，返回链表开始入环的第一个结点。 如果链表无环，则返回 null。

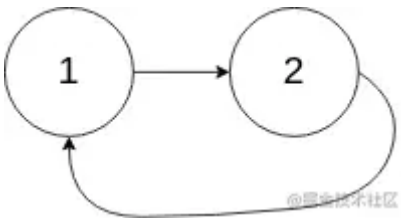
> 示例 1：

输入：head = [3,2,0,-4]（如下图） 输出：tail connects to node index 1 解释：链表中有一个环，其尾部连接到第二个结点。



> 示例 2：

输入：head = [1,2]（如下图）
输出：tail connects to node index 0

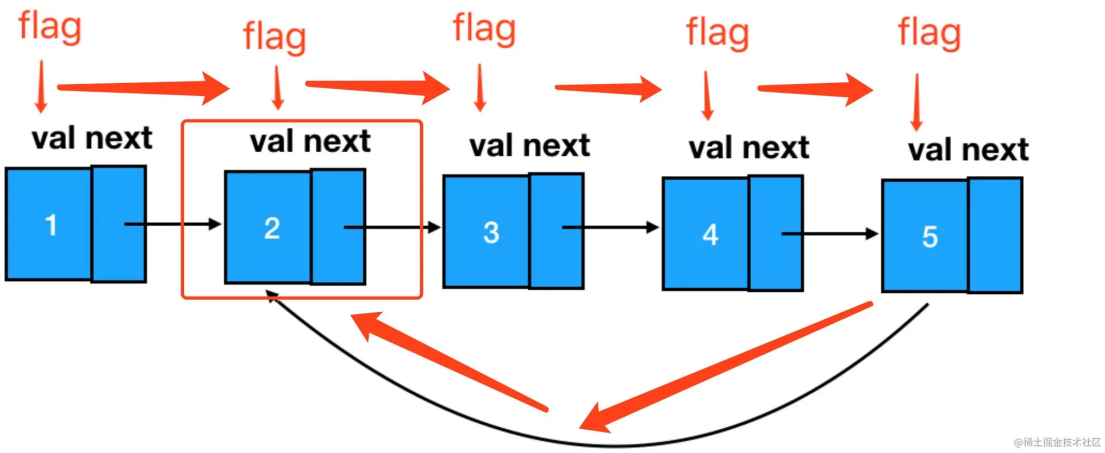


解释：链表中有一个环，其尾部连接到第一个结点。

> 示例 3：

输入：head = [1]（如下图）
输出：no cycle
解释：链表中没有环。

这道题在上道题的基础上，仅仅增加了一个“返回链表的成环起点”，其难度定义就从 easy 上升到了 medium。不过对于掌握了关键解题思路的各位来说，这道题仍然是 easy——因为如果一个结点是环形链表成环的起点，那么它一定是第一个被发现 flag 标志已存在的结点：



这一点不难理解，我们试想如果从头开始遍历一个链表，假如途中进入了一个环，那么首先被打上 flag 标签的其实就是环的起点。待我们遍历完这个环时，即便环上所有的结点都已经被立了 flag，但起点处的 flag 一定最先被我们定位到。因此，我们只需要在第一次发现 flag 已存在时，将对应的结点返回即可：

编码实现

```js
/**
 * @param {ListNode} head
 * @return {ListNode}
 */
const detectCycle = function(head) {
    while(head){
        if(head.flag){
            return head;
        }else{
            head.flag = true;
            head = head.next;
        }
    }
    return null;
};
```

快慢指针的思路

步。这样如果它们是在一个有环的链表里移动，一定有相遇的时刻。这个原理证明起来也比较简单：我们假设移动的次数为 t，slow 移动的路程就是 `t` ，fast 移动的路程为 `2t` ，假如环的长度为 `s` ，那么当下面这个条件：

```js
2t - t = s
```

也就是：

```js
t = s
```

满足时，slow 和 fast 就一定会相遇。反之，如果两者没有相遇，同时 fast 遍历到了链表的末尾，发现 next 指针指向 null，则链表中不存在环。

有兴趣的同学，可以尝试用双指针法实现一遍上面的判定。不过我更加推荐的仍然是“立flag”法，理解难度和编码难度上来说都更加友好，有利于大家实现题目的“秒杀”。

弦外之音

在这一节，大家会发现一个非常有趣的现象——做环形链表的系列题目，难点其实在于你怎么去想明白这个成环的过程、怎么把握成环后的特性。真正编码实现的时候，寥寥数行就可以搞定。这其实也是我想要向大家传达的一个重要的解题习惯——做算法题时，不要急于下手写代码，而应该先静下心来，稳住神、一步一步捋清楚你自己的思路。

之所以要把这点单独拎出来讲，是因为我知道很多同学平时写业务代码比较多。前端业务代码是什么特征？干就完了，对吧？反正就算代码有问题，也可以通过直观的视觉反馈及时发现、及时修复。在肉眼可见的反馈的指导下，你基本不会出什么方向性的问题。

做算法题就大不一样了，真正提交给 OJ 运行之前，除了你自己的逻辑判断之外、没有任何直观的线索能够帮你明确问题的所在。也就是说，如果你一开始压根没想清楚、脑子里本来就是一团乱麻，那么直接开干后往往是越写越乱、最后代码的修复成本也会变得非常高。

路感到不确定、不自信，这时候可以问对方要张纸、先线下梳理一下。真正面试的时候，我们对于自己敲在屏幕上的每一行代码，都应该抱有敬畏之心。

（阅读过程中有任何想法或疑问，或者单纯希望和笔者交个朋友啥的，欢迎大家添加我的微信xyalinode与我交流哈~）

留言

输入评论（Enter换行，⌘ + Enter发送）

发表评论

全部评论（47）

Mingo-233 Lv2　web前端　2月前

我们设置快慢两个指针，fast, slow fast一次前进两步，slow一次前进一步， 设a为第一个节点到入环节点的距离。 a=[0->2] 设b为入环口到相遇点的距离。b=[2->6] 设c为相遇点到入环口的距离。c=[6->2] 当fast，和slow相遇的时候，fast经过的节点是slow的两倍，设slow经过的节点数为S 根据上面的设置 可知 S=a+b ,2S=a+b+c+b，可知 a=c,此时让...

展开

点赞　回复

敲代码的小提琴手 Lv2　前端开发练习生 @ JDT　4月前

今天发现一个问题 142这道题不让改链表结点 也就是说 这个flag的方法用不了了 我写了个双指针的题解 配合其他大佬做得图解 很容易明白的~ leetcode-cn.com

3　回复

外flag的解题方法学习了~

点赞 2

敲代码的小... Lv2 4月前

根据修言大佬的思路写了个题解~ leetcode-cn.com 再次感叹 这个方法太bug了！跟141那道简单题代码几乎完全一样...

点赞 回复

敲代码的小... Lv2 回复 敲代码的小... 3月前

好吧 不能这样 有规定不让改变链表结点的属性害

“根据修言大佬的思路写了个题解~ leetcode-cn.com 再次感叹 这个... ”

点赞 回复

iJay 前端开发 6月前

哈撒给～

点赞 回复

不看完不改名_前... Lv1 前端 6月前

虽然大家都说【快慢指针】是原配，但我还是喜欢【立flag】，简单明了，就是不知道面试官喜欢不，哈哈哈哈哈 ——继续今日份的打卡

2 1

敲代码的小... Lv2 4月前

这个方法蛮新鲜的！学习了hh 之前都是无脑双指针解这种题的

点赞 回复

Joe爱学习66950 业务仔 7月前

如果在入环之前的长度大于环的长度，似乎就不是t=s了吧

2 1

ng_kp Lv2 6月前

应该是 2t-t = ns

1 回复

我是谁我在哪aaa 9月前

快慢指针找入口.

点赞 回复

ailgiP 10月前

😭 想了很久快为什么慢指针能相遇，还有相遇的地点

点赞 1

柠致 8月前

就像在田径场跑圈一样，跑得快的人 总会与 跑得慢的人再次相遇； 若无环，则fast指针先到null

4 回复

KaKaKa 11月前

这节的题终于能自己做出来了

点赞 3

颜酱 Lv3 10月前

哈哈哈 对对对

点赞 回复

KaKaKa 回复 颜酱 5月前

5个月后再看，都忘记了

“哈哈哈 对对对”

点赞 回复

查看更多回复

campcc Lv3 前端工程师 @ 腾讯 1年前

快慢指针

3 回复

点赞 回复

小子王 Lv2 前端 @ 广州 1年前

快慢指针

1 回复

潘小安 Lv2 @不务正业的程序员 @ ... 1年前

打卡

点赞 回复

---

negrochn Lv2 1年前

快慢指针法定位环的起点问题。

1 回复

Wneil 1年前

环形链表,给每个遍历过得元素做标记 然后下一个元素, 如果再次到标记位则为环形链表

点赞 回复

JustDemo Lv1 酒店试睡员 @ 阿里妈妈 1年前

打卡环形链表，感觉比快慢指针和多指针简单点哦

点赞 回复

一只菜鸟攻城狮啊 Lv1 前端攻城狮 @ 北京四环... 1年前

插眼

点赞 回复

金钟罩铁布衫 Lv2 前端 1年前

感觉这章，应该看到的是快慢指针的解法

4 1

2 回复

一位宇航员 前端工程师 @ 阿里巴巴 1年前

第二种快慢指针的证明方法有些问题， 应该是 2t - t = n * s；n属于大于等于1的正整数。而且作者老哥，这个方法最好还是证明一下怎么找到入环点哦。😄

6 回复

Benbinbin Lv3 1年前

「一开始压根没想清楚、脑子里本来就是一团乱麻，那么直接开干后往往是越写越乱」 说到心里去了😭

1 回复

查看全部 47 条回复