经典面试题：环形链表的判断与定位

原创 nettee 面向大象编程 2020-08-02

收录于话题

#算法 4130 #编程 4772 #数据结构 1265 #经典面试题系列 2

作者：nettee

公众号：面向大象编程

大家好，我是 nettee。近期，我会跟大家分享一些「经典面试题」，既讲解题目的解法，也讲解面试中的一些套路。

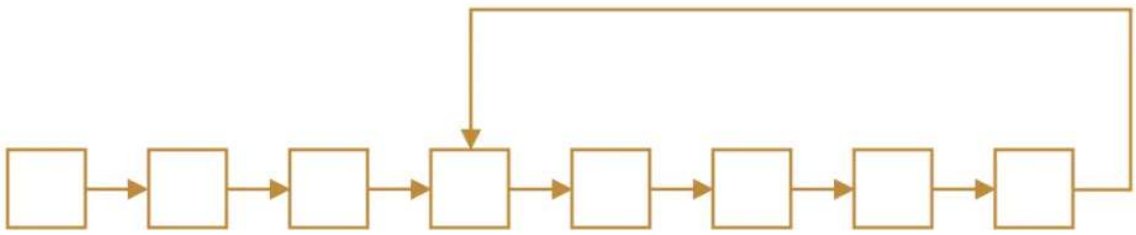
今天这篇文章要讲解的题目是「环形链表」的两道题目：

- **LeetCode 141. Linked List Cycle**（Easy）
- **LeetCode 142. Linked List Cycle II**（Medium）

两道题目都分别有「哈希表」的朴素解法，以及「双指针」的巧妙解法。非常适合大家掌握面试中的答题思路。想顺利通过面试的同学，不要错过这篇文章哟。

判断链表是否有环

链表是线性结构，怎么会出现环呢？实际上是这道题中的链表经过了特殊改造，链表尾部会额外增加一个指针，连接到链表中间的某个结点，如下图所示。

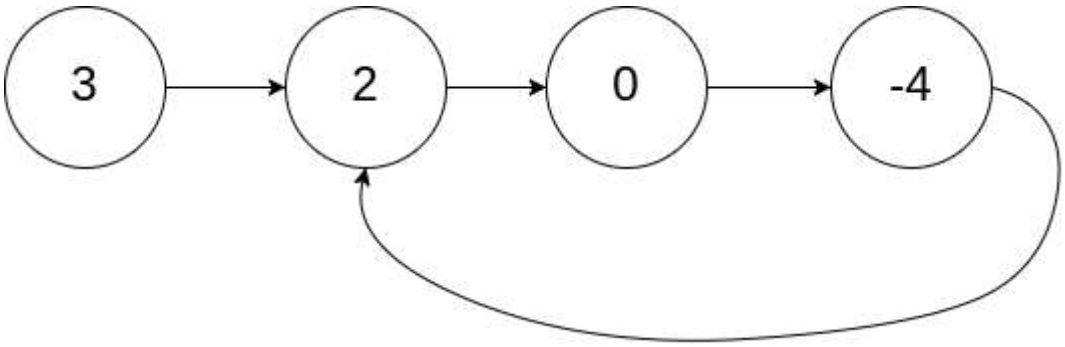


链表尾部额外增加一个指针，形成环路

我们今天要讲的这两道题，都是在这个结构的基础上出的题。

先看 141 题：

LeetCode 141. Linked List Cycle（Easy）

> 给定一个链表，判断链表中是否有环。
>
> 示例：
>
> 输入：
>
> 
>
>
> 输出：true
>
> 解释：链表中有一个环，其尾部连接到第二个结点。

这道题其实是一道非常新颖的题目，考察的是对链表知识的综合掌握。只靠背链表的遍历代码来解题的人，看到这道题会手足无措。因为链表加上环之后，普通的遍历代码根本就不管用了！

```
void traverse(ListNode head) {
    for (ListNode q = head; q != null; q = q.next) {
        // ...
    }
}
```

上面的这段遍历代码，在有环路的链表上，会陷入「无限循环」：指针 `q` 在环路中一直绕圈，永远到达不了结束条件 `q == null` 。

解题基本思路

其实看到前面的链表遍历代码，很多同学应该已经想出这道题的解题关键点了：

我只要顺着链表遍历，如果循环能正常退出（遇到 `q == null` 的情况），那么链表无环；如果循环永远不能退出，那么链表有环。

那么难点又来了，如果循环永远不能退出，我们的代码根本结束不了，不能返回「链表有环」的结果……

> 面试小贴士：
>
> 在面试中，我们经常会遇到以上的情况。自己有了一点思路，但是还不能确定思路对不对。这时候，我们要做的不是「闷头苦想」直到想出最终解法，而是应该及时把自己关键的思路说给面试官。
>
> 面试中最忌讳的是长时间的沉默。及时跟面试官反馈我们初步的思路，可以确定我们的思路是否正确。即使一开始一点思路都没有，也可以直接说出来，请面试官给一点提示。

此时，面试官会告诉我们，以上的思路是正确的。接下来，我们需要能够判断链表有环的情况，并及时退出循环。

那么究竟如何判断呢？可以使用朴素的「哈希表」方案，或者是巧妙的「双指针」方法，下面会分别介绍。

哈希表解法

哈希表解法的基本思路是：把访问过的结点记录下来，如果在遍历中遇到了访问过的结点，那么可以确定链表中存在环。记录访问过的结点，最常用的方法就是使用哈希表了。

有了这一点思路之后，我们很快可以写出相应的题解代码：

```
public boolean hasCycle(ListNode head) {
    // 记录已访问过的结点
    Set<ListNode> seen = new HashSet<>();
    for (ListNode q = head; q != null; q = q.next) {
        if (seen.contains(q)) {
            // 遇到已访问过的结点，确定链表存在环
            return true;
        }
        seen.add(q);
    }
    // 遍历循环正常退出，链表不存在环
    return false;
```

```
    }
```

这样，我们就成功地解决了这道题目。

但接下来面试官会让我们分析算法的时间、空间复杂度。发现空间复杂度是 $O(n)$ 之后，面试官会让我们再写一个空间复杂度更低的解法，也就是说，我们不能使用哈希表这样的额外存储空间了。

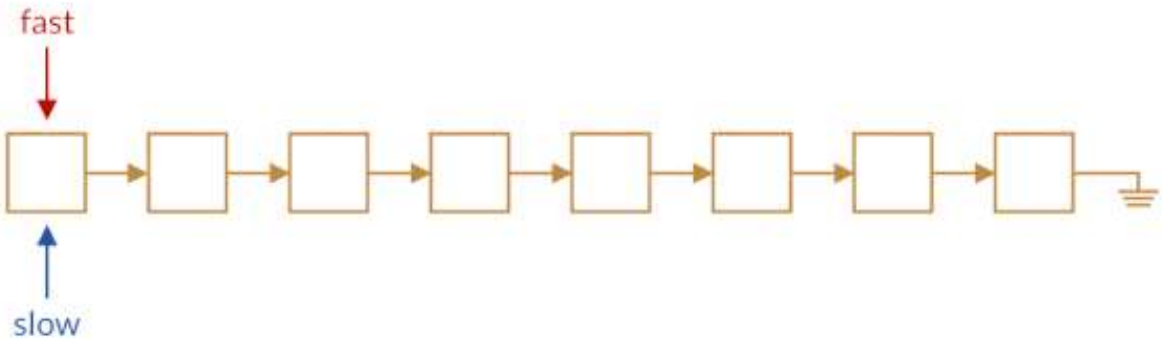
双指针解法

在面试中，做出上面的哈希表解法属于「基本达标」，可以打 70 分了，但是如果要让面试官青睐你，打到 90 分，还要能写出不使用额外存储空间的解法，也就是双指针解法。

实话说，要在面试中凭空想出双指针解法有点困难。因此，这要靠我们平时的积累，多见识不同的解题技巧，这样在面试中可以熟练运用。

链表中的双指针技巧也叫「快慢指针」，指的是用两个指针一快一慢同时遍历链表。

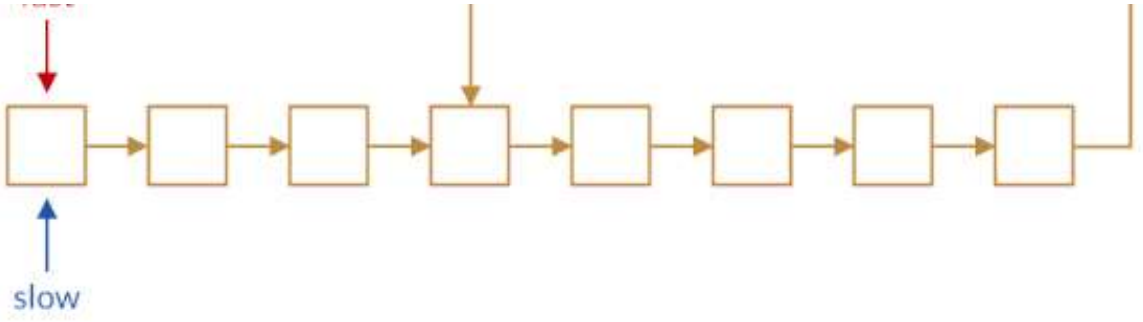
例如，寻找链表中点的操作。使用一快一慢两个指针。快指针一次前进两个结点，速度是慢指针的两倍。当快指针到达链表尾部时，慢指针正好到达的链表的中部，这样就找到的链表的中点，非常巧妙。遍历过程如下图所示。



使用快慢指针寻找链表中点（动图）

如果我们熟悉这个技巧，就可以在环形链表这道题中触类旁通，使用出快慢指针技巧。

我们让快指针一次前进两个结点，慢指针一次前进一个结点。当快慢指针都进入环路时，快指针会将慢指针「套圈」，从后面追上慢指针，如下图所示。



在环路中，快指针套圈慢指针（动图）

这样，如果快指针追上了慢指针，我们就可以判断链表中存在环路。而如果链表中不存在环的话，快指针会永远走在慢指针的前面，它们不会相遇。

依照这个思路，我们可以写出以下的题解代码。

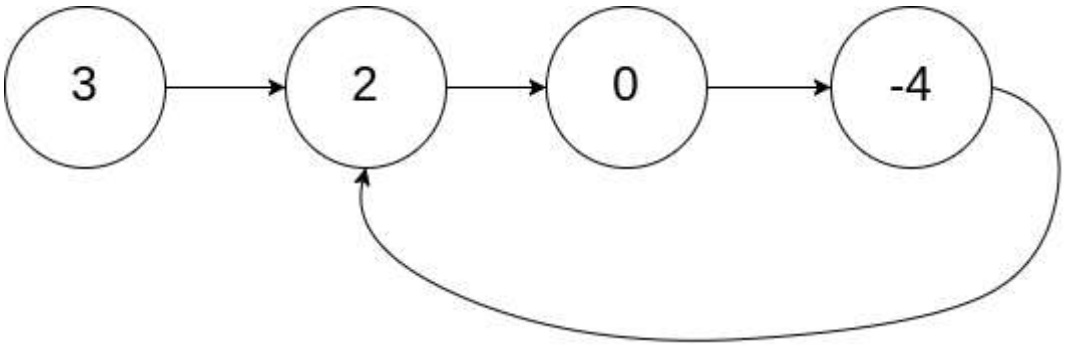
```
public boolean hasCycle(ListNode head) {
    ListNode fast = head;
    ListNode slow = head;
    while (fast != null && fast.next != null) {
        fast = fast.next.next;
        slow = slow.next;
        // fast 和 slow 指向同一个结点，说明存在“套圈”
        if (fast == slow) {
            return true;
        }
    }
    // fast 到达链表尾部，则不存在环
    return false;
}
```

代码简洁，解释清晰，那么这道面试算法题，你可以得 100 分。

寻找链表环的起点

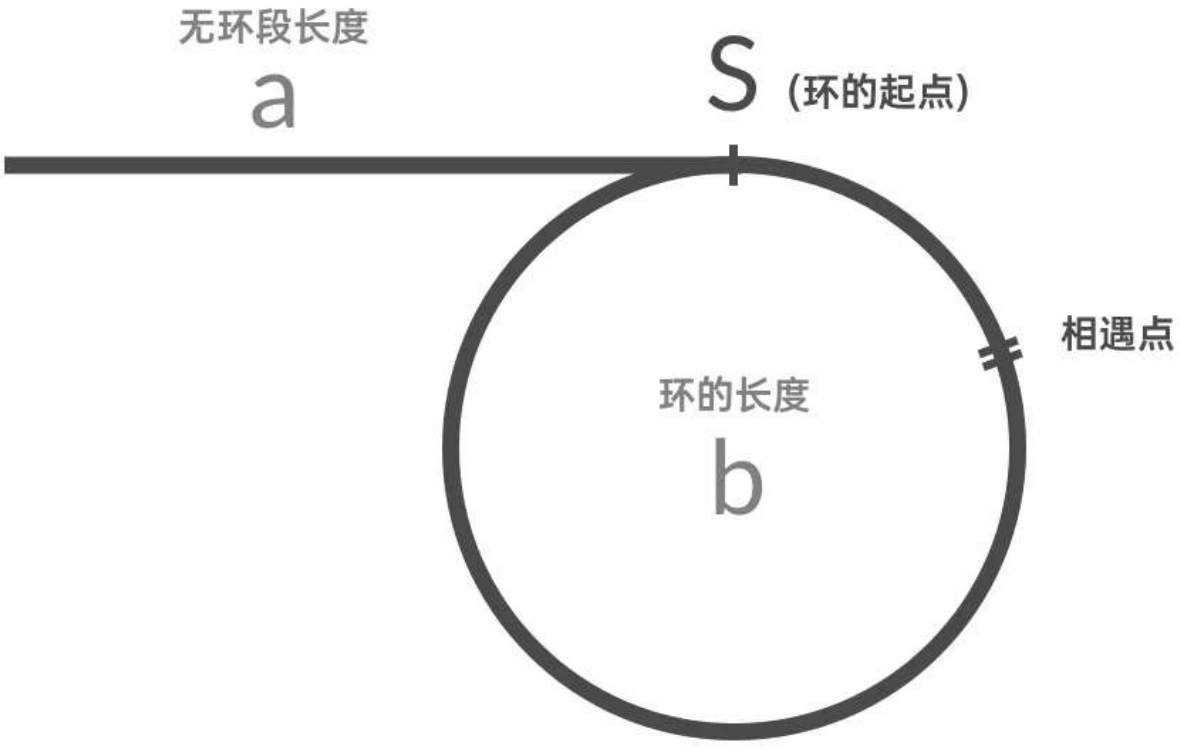
LeetCode 142 这道题，其实相当于 141 的「第二问」。当你用双指针解决了「判断环形链表」的题目之后，面试官看你答得不错，可能会继续追问，如何寻找环形链表的环的起点？如果你能答出这一问，可以得 120 分。

LeetCode 142. Linked List Cycle II（Medium）

> 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` 。不允许修改给定的链表。
>
> 示例：
>
> 输入：
>
> 
>
>
> 输出：结点 <2>
>
> 解释：链表中有一个环，其尾部连接到第二个结点。

如果用哈希表的方法，这道题其实和前一题是一样的，没什么难度。因此，面试官要求你用双指针的方法寻找链表环的起点。

乍一看来，这道题挺难的。我们可以用快慢指针的方法知道链表中是否存在环，但我们不知道两个指针相遇在什么位置，要找到环的起点就更难了。不过不要慌，我们先画个图看看两个指针相遇的过程：



两个指针相遇的过程

如上图所示，链表分为两段：无环段和有环段。我们设无环段的长度为 a，环的长度为 b。当快慢指针相遇时，我们设慢指针已经走了 x 步，那么快指针这时候已经走了 $2x$ 步。快指针套圈了慢指针，也就是比慢指针多走了若干圈。我们可以列出公式：

$$2x - x = k \cdot b$$

其中，k 可以是任意正整数，因为快指针可能套了慢指针不止一圈。将上式化简得到

$$x = k \cdot b$$

这个 x 恰好是慢指针走的步数。也就是说，慢指针目前前进的 x 步，正好是环的长度 b 的整数倍。

那么，慢指针再走 a 步，就可以正好到达环的起点（图中的 S 点）。这是因为，$x + a = a + k \cdot b$，正好够从链表头部出发，走一段无环段（a），再把有环段走 k 圈（$k \cdot b$）。

如果我们在此时再用一个指针 `q` 从链表头部出发。那么指针 `q` 和慢指针 `slow` 会在走了 a 步以后恰好在环的起点处相遇。这样，我们就可以知道环的起点了。

根据以上思路，我们可以写出下面的题解代码：

```
public ListNode detectCycle(ListNode head) {
    ListNode fast = head;
    ListNode slow = head;
    while (fast != null && fast.next != null) {
        fast = fast.next.next;
        slow = slow.next;
        // 快慢指针相遇，说明链表存在环
        if (fast == slow) {
            // 此时 slow 指针距离环的起点的距离恰好为 a
            ListNode q = head;
            while (q != slow) {
                slow = slow.next;
                q = q.next;
            }
            // slow 和 q 相遇的位置一定是环的起点
            return slow;
        }
    }
    // 链表不存在环，返回 null
    return null;
}
```

如果你能思考出以上内容并写出结果，那么面试官会非常认可你，这场面试基本就比较稳了。

面试套路总结

从以上的讲解过程中我们可以发现，这个环形链表的题目其实可以分成几个小问，难度层层递进：

1. 用哈希表的方法判断链表是否存在环
2. 用快慢指针的方法判断链表是否存在环
3. 用快慢指针的方法寻找环的起点

在面试中，面试官往往会先抛出比较简单的一两个小问进行「试探」。当发现你回答得还不错，则会继续提更难的问题，知道你不会为止。如果你一开始就回答得不好，那么面试官可能会认为你在这一块的掌握很一般，很快转而问其他的问题。

把握清楚面试中的这个「层层递进」的规律，我们要注意两点：

第一，在平时复习的时候要注意基础。也许你能做出很多高难度的题目，但如果在简单的题目上回答不上来，面试官可能就不会继续问了，你解决难题的能力可能也发挥不上来。

第二，面试中如果遇到很难的题目、一点也答不上来，不要气馁。这时候很可能是面试官在试探你能力的边界在哪里，侧面说明了你前面可能表现得不错。所以说在面试的全程中都要不慌不忙，会就正常回答，不会就直接说不知道。例如本文例题「寻找环的起点」，其实能在面试中做出来的人不多，能把「判断环是否存在」做好就已经很不错了。

本文的讲解就到此结束，希望能对你后面的面试有所帮助~

往期文章

- **LeetCode** 一题多解 | **53.** 最大子数组和：五种解法完全手册
- 时间复杂度分析快速入门：题型分类法
- **LeetCode** 例题精讲 | **18** 前缀和：空间换时间的技巧

我是 nettee，致力于分享面试算法的解题套路，让你真正掌握解题技巧，做到举一反三。我的《LeetCode 例题精讲》系列文章正在写作中，关注我的公众号，获取最新文章。

原创不易，欢迎分享、点赞和「在看」⇩

喜欢此内容的人还喜欢

随笔 | 美元霸权的瓦解与世界的本质

一个坏土豆

“男朋友说他禁欲了”

趣玩实验站