链表：环找到了，那入口呢？

原创 程序员Carl 代码随想录 2020-08-17 08:35

收录于话题
#数据结构与算法 354 #链表 15

关于代码的一切尽在「代码随想录」

> 找到有没有环已经很不容易了，还要让我找到环的入口？

第142题.环形链表II

题意：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。

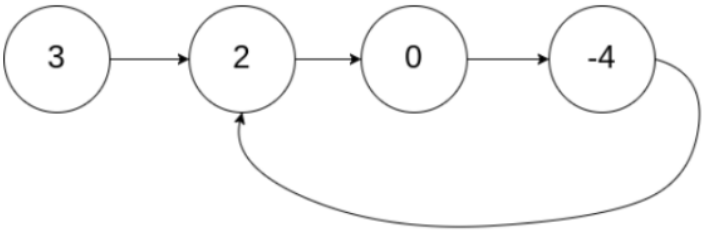
为了表示给定链表中的环，使用整数 pos 来表示链表尾连接到链表中的位置（索引从 0 开始）。

如果 pos 是 -1，则在该链表中没有环。

「说明」：不允许修改给定的链表。

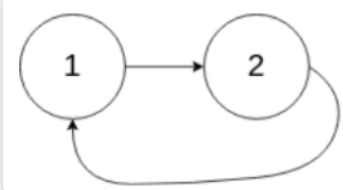
示例 1：

```
输入：head = [3,2,0,-4], pos = 1
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节点。
```



示例 2：

```
输入：head = [1,2], pos = 0
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节点。
```



思路

这道题目，不仅考察对链表的操作，而且还需要一些数学运算。

主要考察两知识点：

- 判断链表是否环
- 如果有环，如何找到这个环的入口

判断链表是否有环

可以使用快慢指针法， 分别定义 fast 和 slow指针，从头结点出发，fast指针每次移动两个节点，slow指针每次移动一个节点，如果 fast 和 slow指针在途中相遇 ，说明这个链表有环。

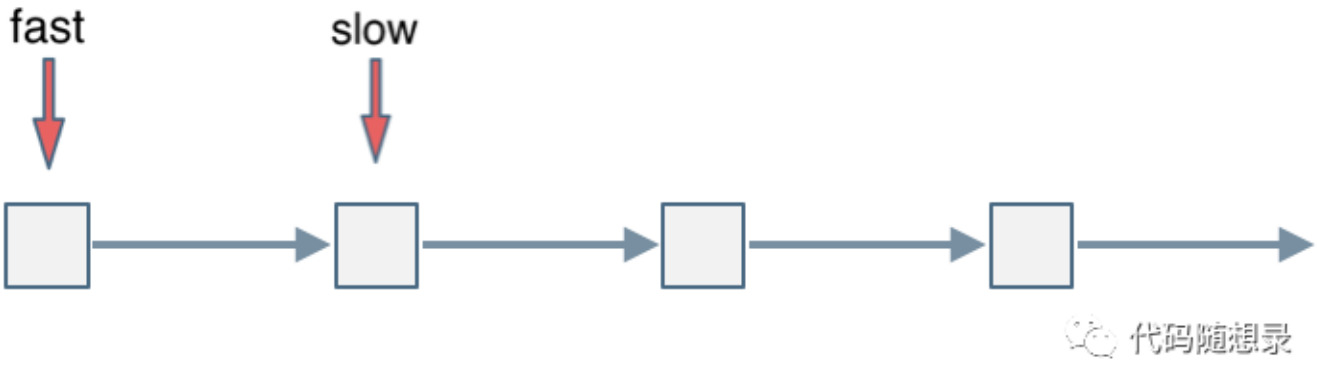
为什么fast 走两个节点，slow走一个节点，有环的话，一定会在环内相遇呢，而不是永远的错开呢

首先第一点：**「fast指针一定先进入环中，如果fast 指针和slow指针相遇的话，一定是在环中相遇，这是毋庸置疑的。」**

那么来看一下，**「为什么fast指针和slow指针一定会相遇呢？」**

可以画一个环，然后让 fast指针在任意一个节点开始追赶slow指针。

会发现最终都是这种情况， 如下图：



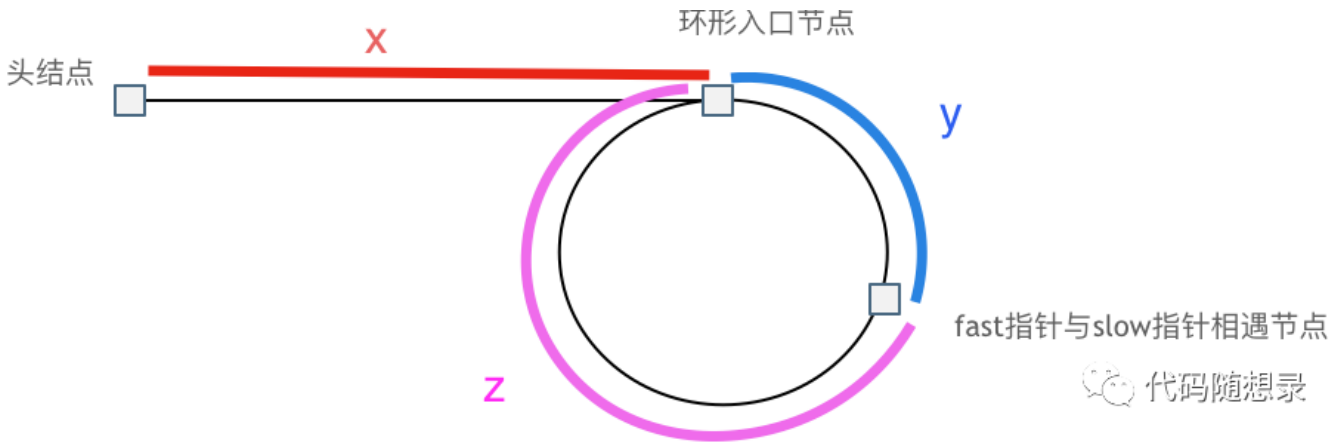
fast和slow各自再走一步， fast和slow就相遇了

这是因为fast是走两步，slow是走一步，**「其实相对于slow来说，fast是一个节点一个节点的靠近slow的」**，所以fast一定可以和slow重合。

如果有环，如何找到这个环的入口

「此时已经可以判断链表是否有环了，那么接下来要找这个环的入口了。」

假设从头结点到环形入口节点 的节点数为x。环形入口节点到 fast指针与slow指针相遇节点 节点数为y。从相遇节点 再到环形入口节点节点数为 z。如图所示：



那么相遇时：slow指针走过的节点数为：`x + y`， fast指针走过的节点数：`x + y + n (y + z)`，n为fast指针在环内走了n圈才遇到slow指针， (y+z) 为 一圈内节点的个数A。

因为fast指针是一步走两个节点，slow指针一步走一个节点， 所以 fast指针走过的节点数 = slow指针走过的节点数 * 2：

```
(x + y) * 2 = x + y + n (y + z)
```

两边消掉一个（x+y）：`x + y = n (y + z)`

因为要找环形的入口，那么要求的是x，因为x表示 头结点到 环形入口节点的的距离。

所以要求x ，将x单独放在左面：`x = n (y + z) - y` ，

再从n(y+z)中提出一个 (y+z) 来，整理公式之后为如下公式：`x = (n - 1) (y + z) + z`
注意这里n一定是大于等于1的，因为 fast指针至少要多走一圈才能相遇slow指针。

这个公式说明什么呢？

先拿n为1的情况来举例，意味着fast指针在环形里转了一圈之后，就遇到了 slow指针了。

当 n为1的时候，公式就化解为 `x = z`，

这就意味着，**「从头结点出发一个指针，从相遇节点 也出发一个指针，这两个指针每次只走一个节点， 那么当这两个指针相遇的时候就是 环形入口的节点」**。

也就是在相遇节点处，定义一个指针index1，在头结点处定一个指针index2。

让index1和index2同时移动，每次移动一个节点， 那么他们相遇的地方就是 环形入口的节点。

那么 n如果大于1是什么情况呢，就是fast指针在环形转n圈之后才遇到 slow指针。

其实这种情况和n为1的时候 效果是一样的，一样可以通过这个方法找到 环形的入口节点，只不过，index1 指针在环里 多转了(n-1)圈，然后再遇到index2，相遇点依然是环形的入口节点。

C++代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        ListNode* fast = head;
        ListNode* slow = head;
        while(fast != NULL && fast->next != NULL) {
            slow = slow->next;
            fast = fast->next->next;
            // 快慢指针相遇，此时从head 和 相遇点，同时查找直至相遇
            if (slow == fast) {
                ListNode* index1 = fast;
                ListNode* index2 = head;
                while (index1 != index2) {
                    index1 = index1->next;
                    index2 = index2->next;
                }
                return index2; // 返回环的入口
            }
        }
        return NULL;
    }
};
```

欢迎在评论区留言讨论！

我将算法学习相关的资料已经整理到了Github ： https://github.com/youngyangyang04/leetcode-master，里面还有leetcode刷题攻略、各个类型经典题目刷题顺序、思维导图看一看一定会有所收获！

-------end-------

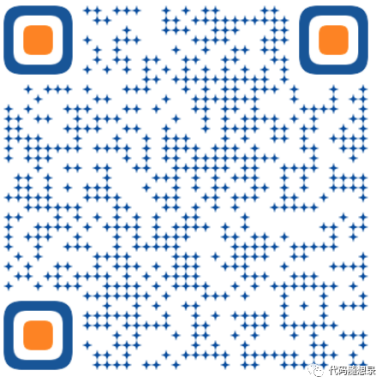
因为公众号改版，时间线被打乱，一些精彩文章大家可能错过了。如果感觉这里的文章对你有帮助，赶紧给「代码随想录」加一个星标吧，方便第一时间阅读文章。

链表知识回顾

链表：听说过两天反转链表又写不出来了？
链表：一道题目考察了常见的五个操作！
链表：听说用虚拟头节点会方便很多？
关于链表，你该了解这些！

「代码随想录」期待你的关注！

每天一道经典算法题目，助你轻松学习算法！



我就知道你[在看]

收录于话题 #数据结构与算法 354

< 上一篇
关于哈希表，你该了解这些！

下一篇 >
链表：听说过两天反转链表又写不出来了？