142. 环形链表 II

地址： 环形链表 II

题目：

- English：

142. Linked List Cycle II

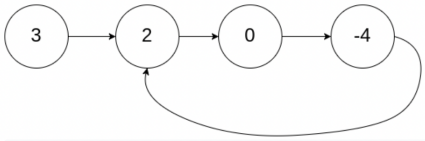
Medium 4591 329 Add to List Share

Given a linked list, return the node where the cycle begins. If there is no cycle, return `null`.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter**.

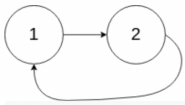
Notice that you **should not modify** the linked list.

Example 1:




```
Input: head = [3,2,0,-4], pos = 1
Output: tail connects to node index 1
Explanation: There is a cycle in the linked list, where
tail connects to the second node.
```

Example 2:




```
Input: head = [1,2], pos = 0
Output: tail connects to node index 0
Explanation: There is a cycle in the linked list, where
tail connects to the first node.
```

Example 3:


```
Input: head = [1], pos = -1
Output: no cycle
Explanation: There is no cycle in the linked list.
```

Constraints:

- The number of the nodes in the list is in the range `[0, 10^4]`.
- `-10^5 <= Node.val <= 10^5`
- `pos` is `-1` or a **valid index** in the linked-list.

- 中文：

142. 环形链表 II

难度 中等 1060

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`。

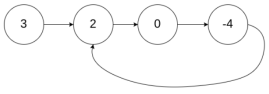
为了表示给定链表中的环，我们使用整数 `pos` 来表示链表尾连接到链表中的位置（索引从 0 开始）。 如果 `pos` 是 `-1`，则在该链表中没有环。**注意，`pos` 仅仅是用于标识环的情况，并不会作为参数传递到函数中。**

说明： 不允许修改给定的链表。

进阶：

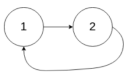
- 你是否可以使用 `O(1)` 空间解决此题？

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。
```

提示：

- 链表中节点的数目范围在范围 `[0, 10^4]` 内
- `-10^5 <= Node.val <= 10^5`
- `pos` 的值为 `-1` 或者链表中的一个有效索引

通过次数 252,285 提交次数 459,695

思路1：快慢指针

- ⭐ 确定是否存在环（参照 141. 环形列表 I）：
  - ⭐ 无环：返回 null;
  - ⭐ 有环：快指针与慢指针相遇，结束查找环的循环。
- ⭐ 查找环的入口：
  - ⭐ 将 快指针 移动到 列表头；
  - ⭐ 然后开始迭代移动快慢指针，每次均只移动一个节点；
  - ⭐ 当快慢指针再次相遇时，就是环的入口。

代码：

```java
// Java
// Time : 2021 - 07 -12


public ListNode detectCycle(ListNode head) {

      ListNode slow = head;
      ListNode fast = head;



      // find the cycle whether exist :
      // 1. no : return null
      // 2. exist : find the meet position


      while (true) {
        if (fast == null || fast.next == null) return null;
        slow = slow.next;
        fast = fast.next.next;
        if (fast == slow) break;
      }


      // find the cycle entrance :
      // 1. move fast pointer to head position
      // 2. move slow & fast pointer one node every loop until slow meets fast
      //    pointer (reason for 2 : when slow pointer meets fast pointer,
      //    they just arrive at the entrance. )


      fast = head;


      while (slow != fast) {
        slow = slow.next;
        fast = fast.next;
      }

```

```
40
41          // return result node
42          return fast;
43      }
```

复杂度分析：

- 时间复杂度：O(N) - 并列两套循环，所以为 O(2N)，省略常数为 O(N)
- 空间复杂度：O(1) - 没有额外申请内存空间