141. 环形链表 I

地址：环形链表 I

题目：

- English：

141. Linked List Cycle

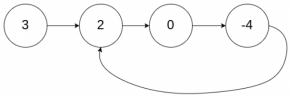
Easy 4927 638 Add to List Share

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

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**.

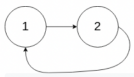
Return `true` *if there is a cycle in the linked list*. Otherwise, return `false`.

Example 1:




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

Example 2:




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

Example 3:


```
Input: head = [1], pos = -1
Output: false
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：

141. 环形链表

难度 简单 1118

给定一个链表，判断链表中是否有环。

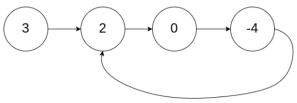
如果链表中有某个节点，可以通过连续跟踪 `next` 指针再次到达，则链表中存在环。为了表示给定链表中的环，我们使用整数 `pos` 来表示链表尾连接到链表中的位置（索引从 0 开始）。 如果 `pos` 是 `-1`，则在该链表中没有环。**注意：`pos` 不作为参数进行传递**，仅仅是为了标识链表的实际情况。

如果链表中存在环，则返回 `true` 。否则，返回 `false` 。

进阶：

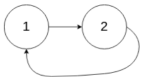
你能用 *O(1)*（即，常量）内存解决此问题吗？

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。
```

提示：

- 链表中节点的数目范围是 $[0, 10^4]$
- $-10^5 <= Node.val <= 10^5$
- `pos` 为 `-1` 或者链表中的一个 **有效索引**。

思路1：双指针-快慢指针

分析

⭐ 双指针，一快一慢；

⭐ 起始：快慢指针均从列表头开始；

⭐ 有环：

- 如果快指针与慢指针相遇，即 slow == fast, 表示有 环，返回 true;

- 无环：
  - 如果快指针指向空节点 null， 即 fast == nul or fast.next == nulll, 即，列表被遍历完全，没有环,返回false;

- 指针移动：
  - 慢指针移动一个节点：slow = slow.next;
  - 快指针移动两个节点：fast = fast.next.next;
    - 快指针，移动两个节点，以确保快指针可以追上慢指针，且不会超过链表；
    - 快指针，移动一个节点，快指针追不上慢指针，造成超时；
    - 快指针，移动多个节点，快指针会因移动过快，再没环的情况下，超过链表，报错。

代码：

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


public class Solution {


  public boolean hasCycle(ListNode head) {

    ListNode slow = head;          // slow pointer
    ListNode fast = head;     // fast pointer


    while (true) {

      // no cycle
      if (fast == null || fast.next == null) return false;

      slow = slow.next;       // move the slow pointer to next node
      fast = fast.next.next;  // move the fast pointer to next next node

      // when slow pointer meet fast pointer find the cycle,
      // break loop and return true
      if (slow == fast) return true;
    }
  }
}
```

```python
# Python
# Time : 2021 - 07 - 10


class Solution:
    def hasCycle(self, head: ListNode) -> bool:
```

```
        slow = head
        fast = head.next


        while true:
            if fast == None or fast.next == None:
                return False

            slow = slow.next
            fast = fast.next.next

            if slow == fast:
               return True
```

复杂度分析：

- 时间复杂度：O(n)
- 空间复杂度：O(1)