# 题目

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。

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。

**示例 3：**

输入：head = [1], pos = -1

输出：no cycle

解释：链表中没有环。

**进阶：**

你是否可以不用额外空间解决此题？

# 分析

**方法一：哈希表**

**代码：**

class Solution {

public:

ListNode\* detectCycle(ListNode\* head) {

unordered\_map <ListNode\*, int> m;

while (head)

{

m[head] ++;

if (m[head] > 1) return head;

head = head->next;

}

return nullptr;

}

};

这种方法使用了额外的空间。

时间复杂度：O(n)。

空间复杂度：O(n)。

## 方法二：双指针/快慢指针

/\*\*

\* 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\* slow=head,\*fast=head;

while(fast && fast->next)

{

slow = slow->next;

fast = fast->next->next;

//此时判断出是环形链表，接下来寻找第一个入环的节点

if(fast == slow)

{

slow = head;

while(fast != slow)

{

fast = fast->next;

slow = slow->next;

}

return slow;

break;

}

}

return nullptr;

}

};

或：

/\*\*

 \* 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 \*slow = head;

        ListNode \*fast = head;

        bool isExistsLoop = false;

        while(slow && fast && fast->next)

        {

            slow = slow->next;

            fast = fast->next->next;

            if(slow == fast)

            {

                isExistsLoop = true;

                break;

            }

        }

        if(isExistsLoop)

        {

            slow = head;

            while(slow != fast)

            {

                slow = slow->next;

                fast = fast->next;

            }

            return slow;

        }

        return nullptr;

    }

};