# 题目

给定一个链表，判断链表中是否有环。

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。

**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。



**进阶：**

你能用 O(1)（即，常量）内存解决此问题吗？

# 分析

## 方法一：哈希表

**思路：**

以链表节点为Key值，出现次数为value，如果存在唤醒链表则values值必定=2。

**说明：**能够采用unordered\_map，也可以采用unordered\_set实现。

**代码：**

class Solution {

public:

bool hasCycle(ListNode \*head) {

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

while(head)

{

mp[head]++;

if(mp[head] > 1) return true;

head = head->next;

}

return false;

}

};

**复杂度分析：**

时间复杂度：由于需要把链表扫描一遍，因此其时间复杂度为O(n)。

空间复杂度：由于要创建哈希表，因此空间复杂度为O(n)。

## 方法二：双指针/快慢指针

**思路：**

通过使用具有不同速度的快、慢两个指针遍历链表，空间复杂度可以被降低至O(1)。慢指针每次移动一步，而快指针每次移动两步。

如果列表中不存在环，最终快指针将会最先到达尾部，此时我们可以返回 false。

**代码：**

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

bool hasCycle(ListNode \*head) {

if(nullptr == head || nullptr == head->next)

return false;

ListNode \*slow,\*fast;

slow = head;

fast = head->next;

while(slow != fast)

{

if(nullptr==fast || nullptr==fast->next)

return false;

slow = slow->next;

fast = fast->next->next;

}

return true;

}

};

**复杂度：**

**时间复杂度：**O(n)，让我们将n设为链表中结点的总数。为了分析时间复杂度，我们分别考虑下面两种情况。

**链表中不存在环：**

快指针将会首先到达尾部，其时间取决于列表的长度，也就是O(n)。

**链表中存在环：**

我们将慢指针的移动过程划分为两个阶段：非环部分与环形部分：

1. 慢指针在走完非环部分阶段后将进入环形部分：此时，快指针已经进入环中迭代次数=非环部分长度= N
2. 两个指针都在环形区域中：考虑两个在环形赛道上的运动员 - 快跑者每次移动两步而慢跑者每次只移动一步。其速度的差值为 1，因此需要经过 二者之间距离/速度差值次循环后，快跑者可以追上慢跑者。这个距离几乎就是"环形部分长度 K"且速度差值为1，我们得出这样的结论：迭代次数=近似于"环形部分长度 K".

因此，在最糟糕的情形下，时间复杂度为O(N+K)，也就是O(n)。

**空间复杂度：**O(1)，我们只使用了慢指针和快指针两个结点，所以空间复杂度为O(1)。

**错误写法：**

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     ListNode \*next;

 \*     ListNode(int x) : val(x), next(NULL) {}

 \* };

 \*/

class Solution {

public:

    bool hasCycle(ListNode \*head) {

        if(NULL==head || (NULL!=head && head==head->next))

            return true;

        ListNode \*slow = head;

        ListNode \*fast = head->next;

        while(head) //如果是循环链表则死循环，这里应该是快慢指针不同

        {

            if(slow == fast)

                return true;

            slow = slow->next;

            fast = (fast->next)->next;

            head = head->next;

        }

        return false;

    }

};

另一种写法：

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     ListNode \*next;

 \*     ListNode(int x) : val(x), next(NULL) {}

 \* };

 \*/

class Solution {

public:

    bool hasCycle(ListNode \*head) {

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

        while(fast && fast->next)

        {

            slow = slow->next;

            fast = fast->next->next;

            if(slow == fast)

                return true;

        }

        return false;

    }

};

或：

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     ListNode \*next;

 \*     ListNode(int x) : val(x), next(NULL) {}

 \* };

 \*/

class Solution {

public:

    bool hasCycle(ListNode \*head) {

        if(!head || !head->next)

        {

            return false;

        }

        ListNode \*slow = head;

        ListNode \*fast = head;

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

        {

            slow = slow->next;

            fast = fast->next->next;

            if(slow == fast)

            {

                return true;

            }

        }

        return false;

    }

};

# 拓展

双指针的应用：

1. 链表

环形链表

删除链表倒数第K个数字、删除链表重复项、移除元素

分割链表

旋转链表

合并两/K个有序序列

两两交换链表中的节点

1. 字符串

实现strstr

回文串