链表中的环判断、环长度、入口节点

# 链表中是否存在环

## 题目介绍

单链表有环的定义：**链表的尾节点指向了链接中间的某个节点**。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*1.链表中是否含有环\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* LeetCode\_141\_LinkedListCycle\_Easy\_JianzhiOffer\_23\_DoesLinkedListHasLoop

\* https://leetcode.com/problems/linked-list-cycle/description/

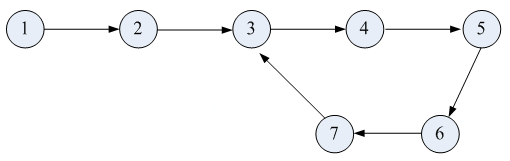
\* 题目介绍：给定一个链表，判断其内部是否含有环。

\* Given a linked list, determine if it has a cycle in it.

\* Follow up:

\* Can you solve it without using extra space?

\*  **Leetcode141\_LinkedListCycle\_Easy**



## 思路分析

\* 思路分析：

\* 方法1. 经典算法：快慢指针。利用两个指针，一个fast，一个slow，fast一次走两步，

\* slow 一次走一步，如果存在环，则fast与slow会指向同一个节点；

\* 若fast遇到null，则表述不存在。

\* 空间复杂度为O(1).

\* <前提：只要存在环，快指针和慢指针必定相遇。>

\* 链表中是否有环的判断

\* 可以设置两个指针(fast,slow)，初始值均指向头，slow每次向前一步，fast每次向前两步；

\* 如果链表中有环，则fast先进入环中，而slow后进入环中，两个指针在环中必定相遇;

\* 如果fast遍历到尾部为NULL，则无环。

\* 方法2. 利用Set集合实现：

\* 和寻找两个链表的第一个公共节点类似，这里更简单些。

\* 在向set集合中add元素之前，首先判断set中是否含有该节点，

\* 若有该节点，即存在环，且该节点就是环的入口节点；

\* 若没有，则直接add进去。

## Java代码

### **方法1：经典算法：快慢双指针。**

public boolean hasCycle(ListNode head) {

if (head == null) return false;//可删去，后面会对head判断

ListNode fast = head, slow = head;

while (fast != null && fast.next != null) {//只要fast或fast.next为null，则说明不存在环

fast = fast.next.next;

slow = slow.next;

if (fast == slow) return true;

}

return false;

}

### 方法2：利用Set集合解决

public boolean hasCycleBySet(ListNode head) {

if (head == null) return false;

Set<ListNode> set = new HashSet<ListNode>();

while(head != null){

if(set.contains(head)){

return true;

}else{

set.add(head);

}

head = head.next;

}

return false;

}

# 链表中环的长度

## 题目介绍

/\*\*\*\*链表T5：求链表中环的长度\*\*\*\*\*\*/

\* 题目介绍：链表T5：求链表中环的长度

\* 若链表中存在环，则返回环的长度；否则返回0。

## 思路分析

\* 思路分析：判断链表中是否存在环的基础上，改进算法。

\* 判断链表中是否含有环，方法是fast和slow一定在环中同时指定一个节点，

\* 那么找到该节点之后，通过循环走一圈计数即可。

\* 判断有没有环都是通过找到了环中的一个节点，因此，根据该节点转一圈并计数即可。

\* 可以根据快慢指针，也可以根据Set集合。

## Java代码

### 方法1：

/\*\*

\* 方法1：

\*/

public int getCycleLength(ListNode head) {

if (head == null) return 0;//后面包括

ListNode fast = head, slow = head;

while (fast != null && fast.next != null) {//只要fast或fast.next为null，则说明不存在环

fast = fast.next.next;

slow = slow.next;

if (fast == slow) break;

}

if(fast == null) return 0;//不存在环

int count = 1;

while (fast.next != slow) {

count++;

fast = fast.next;

}

return count;

}

### 方法2：利用Set集合求环长度。

public int **getCycleLengthBySet**(ListNode head) {

if (head == null) return 0;

Set<ListNode> set = new HashSet<ListNode>();

while(head != null){

if(set.contains(head)){

break;

}else{

set.add(head);

}

head = head.next;

}

if(head == null) return 0;

int len = 1;

ListNode temp = head;

while(temp.next != head){

len++;

temp = temp.next;

}

return len;

}

# 链表中环的入口节点

## 题目介绍

\* LeetCode\_142\_LinkedListCycleII

\* https://leetcode.com/problems/linked-list-cycle-ii/description/

\* 难度：Medium

\* 题目介绍：

\* 给一个链表，若其中包含环，请找出该链表的环的入口结点，否则，输出null。

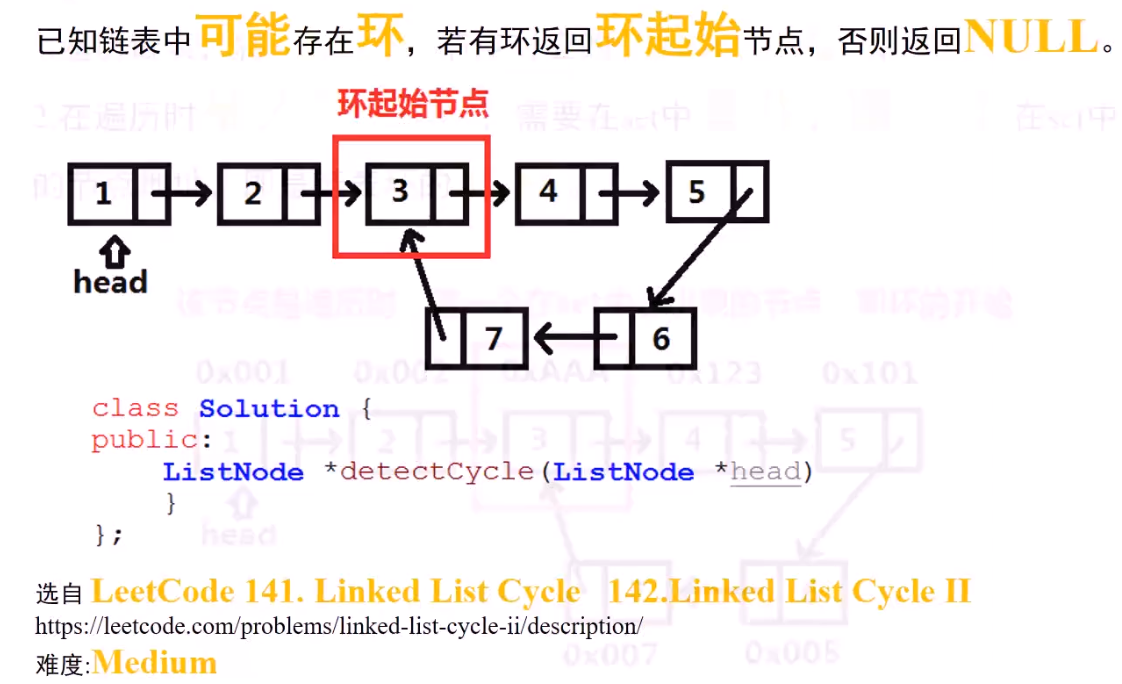
\*Given a linked list, return the node where the cycle begins.

\* If there is no cycle, return null.

\* Note: Do not modify the linked list.

\* Follow up:

\* Can you solve it without using extra space?



## 思路分析

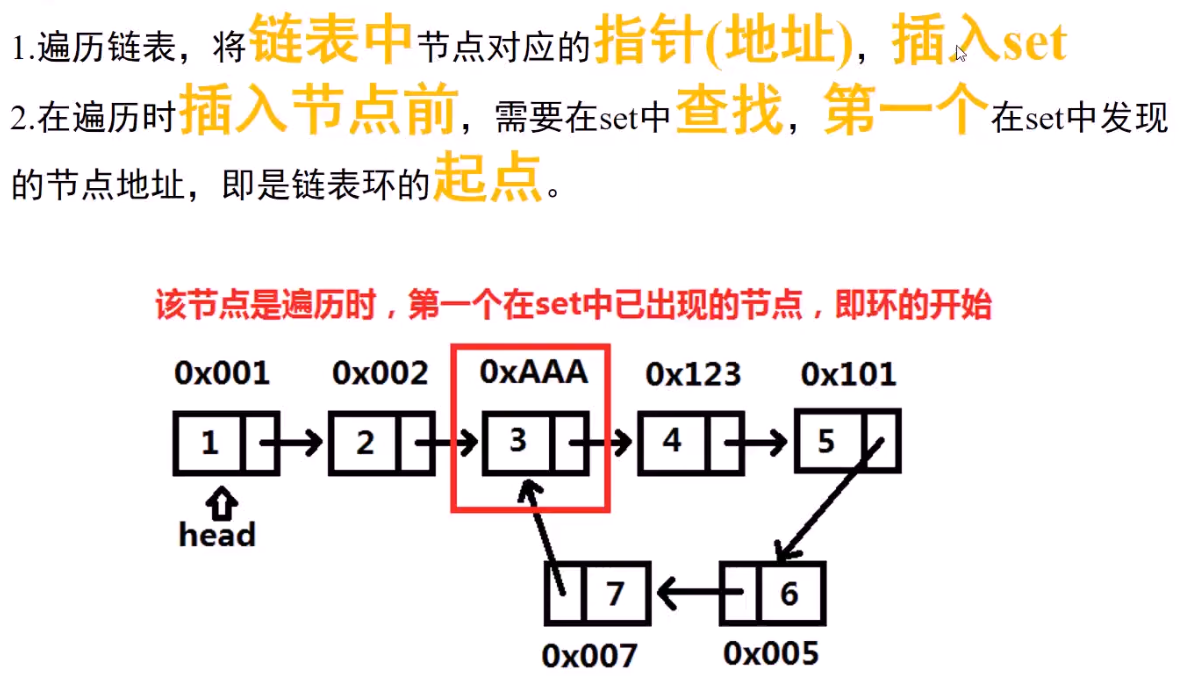
### 方法1： 利用Set集合实现。

\* 和寻找两个链表的第一个公共节点类似，这里更简单些。

\* 在向set集合中add元素之前，首先判断set中是否含有该节点，

\* 若有该节点，即存在环，且该节点就是环的入口节点；

\* 若没有，则直接add进去。



### 方法2：根据环长度。

\* 思路分析：首先判断链表中是否存在环；不存在，返回null；

\* 然后，获取环的长度n，若长度为0，表示环不存在；

\* 最后，利用两个引用ahead和behind，先让ahead走n步，

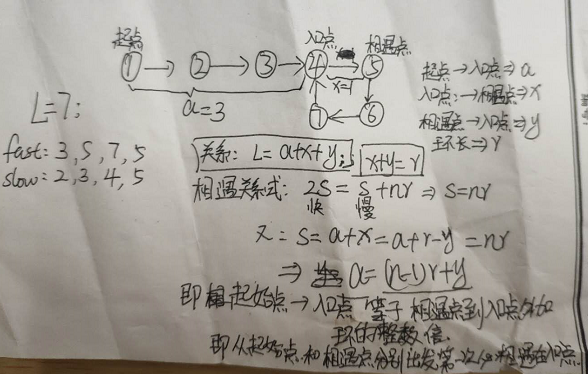
\* 再让它们同时前进，步伐都是，它们相遇的节点就是环的入口节点。

### 方法3：**利用数学推导**

最终结论：**环入口点与起始点的距离等于相遇点至环入口节点的距离加上环的整数倍**。

对应算法：**在判断出相遇点后，设置两个指针，一个指向相遇点，一个指向环起始点，两指针必定在环的入口节点第一次相遇**。

数学分析：

：

## Java代码

### 方法1：利用Set集合解决。

public ListNode EntryNodeOfCycleBySet(ListNode head) {

if (head == null) return null;

Set<ListNode> set = new HashSet<ListNode>();

while(head != null){

if(set.contains(head)){

return head;

}else{

set.add(head);

}

head = head.next;

}

return null;

}



### 方法2：根据环长度。

public ListNode **EntryNodeOfLoop**(ListNode head){

if (head == null) return null;

ListNode fast = head,slow = head;

int loopLength = 0;

while(fast != null && fast.next != null){

fast = fast.next.next;

slow = slow.next;

if(fast == slow){//判断是否相遇，一定在环中某节点相遇

loopLength++;

while(fast.next != slow){

fast = fast.next;

loopLength++;

}

**break;//容易忘记break，导致无限循环。**

**// 单独求环长度时是return结束，索引在这里一定要记得break。**

}

}

if(loopLength <= 0) return null;//不存在环

ListNode ahead = head,behind = head;

while(loopLength-- > 0){//ahead先走loopLength步

ahead = ahead.next;

}

//相遇点即环的入口点

**while(ahead != behind){**

**ahead = ahead.next;**

**behind = behind.next;**

**}**

return ahead;

}

### 方法3：利用数学公式。

public ListNode **EntryNodeOfLoop2**(ListNode head){

if (head == null) return null;

ListNode fast = head,slow = head;

int loopLength = 0;

while(fast != null && fast.next != null){

fast = fast.next.next;

slow = slow.next;

if(fast == slow){//判断是否相遇，一定在环中某节点相遇

**ListNode start = head;**

**while(start != fast){//起点与相遇点必定在入口点第一次相遇**

**start = start.next;**

**fast = fast.next;**

**}**

**return fast;//一定要写在里面，否则用break跳出循环**

}

}

return null;

}

