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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*链表T5：链表中是否含有环\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* 题目介绍：给定一个链表，判断其内部是否含有环。

\* Leetcode141LinkedListCycle\_Easy

## 思路分析

\* 思路分析：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。

## Java代码

public boolean hasLoop(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;

}

## 数学分析

# 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。

## Java代码

public int getLoopLength(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){**

**int count = 1;**

**while(fast.next != slow){**

**fast = fast.next;**

**count++;**

**}**

**return count;//不用return的时候，记得使用break跳出循环，否则无限循环**

**}**

}

return 0;

}

# 链表中环的入口节点

## 题目介绍

/\*\*\*\*\*\*\*\*链表T6：找出链表中环的入口节点\*\*\*\*\*\*/

\* 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。

## 思路分析

\* 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。

## Java代码

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;

}