问题1、判断链表是否存在环。

设置两个指针fast和slow，初始值都指向头指针，slow每次前进一步，fast每次前进两步。如果存在环，则fast必先进入环，而slow后进入环，两个指针必定相遇（见下面的证明1）。（当然，fast先到达尾部为NULL，则为无环链表）。程序如下：

bool isExitsLoop(Node\* head) {

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

//fast && fast->next保证了fast可以接着向下移动

while(fast && fast->next) {

//slow 每次走一步

slow = slow->next;

//fast 每次走两步

fast = fast->next->next;

//若相遇，则证明有环

if(slow == fast)

break;

}

/\*

上述程序结束循环的条件是fast->next == NULL或者fast为NULL或者slow == fast，综上所述若fast为 NULL或者fast->next为NULL则没有环，返回false；否则有环，返回true

\*/

return !(fast == NULL || fast->next == NULL);

}

2、若链表存在环，找到环的入口点

如果fast和slow相遇，那么在相遇时，slow肯定没有遍历完链表，而fast已经在环内循环了n圈（n>=1）（见下面的证明2）。假设slow走了s步，则fast走了2s步（fast的步数还等于s加上在环上多转的n圈），设环长为r，则：

2s=s+nr

s=nr

设整个链表长L，入口环与相遇点距离为x，起点到环入口点的距离为a，则

a+x=s=nr

a+x=(n-1)r+r=(n-1)r+L-a

a=(n-1)r+(L-a-x)

(L-a-x)为相遇点到环入口点的距离。由此可知，从链表头到环入口点等于(n-1)循环内环+相遇点到环入口点。于是可以从链表头和相遇点分别设一个指针，每次各走一步，两个指针必定相遇，且相遇第一点为环入口点。程序描述如下

Node\* findLoopPort(Node \*head) {

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

//找到相遇点

while(fast && fast->next) {

//slow go one step

slow = slow->next;

//fast go 2 steps

fast = fast->next->next;

if(fast == slow) {

break;

}

}

//don't have the circle.没有环

if(fast == NULL || fast->next == NULL) {

return NULL;

}

//此时，fast和slow都指向相遇点

slow = head //slow指向头节点

while(slow != fast) {

//they both go one step

slow = slow->next;

fast = fast->next;

}

return slow;

}

扩展问题3、判断两个单链表是否相交？

法1、对链表1中的每个节点p1，判断链表2中是否有一个节点p2指向p1

loop:p1从head1到最后一个节点

loop:p2从head2到最后一个节点

if(p2是否指向p1)

相交

break

时间复杂度：O(list1.length \* list2.length)

空间复杂度：O(1)

法2、使用hash表

loop：p1从head1到最后一个节点

把p1放入hash表table中

loop:p2从head2到最后一个节点

if(p2在hash表中)

相交

时间复杂度:O(list1.length + list2.length)

空间复杂度：O(list1.length)

法3、将其中一个链表首尾相连，检测另一个链表是否存在环，如果存在，则两个链表相交，而检测出来的依赖环入口点即为相交的第一个点。程序描述如下：

找到list1的最后一个节点tail1

tail1->next=head1

判断head2是否存在环

tail1->next=NULL; //恢复tail1

法4、如果两个链表相交，**那么两个链表从相交点到链表结束都是相同的节点**。可以先分别遍历找出两个链表的尾节点，**如果两个个尾节点相同，则两个链表相交。**程序描述如下：

//找到list1的最后一个节点p1

p1=head1

while(p1->next不是NULL)

p1=p1->next

找出list2的最后一个节点p2

if(p1==p2)

相交

else

不相交

时间复杂度:O(list1.length + list2.length)

空间复杂度：O(1)

扩展问题4、如果两个链表相交，找出相交的第一个节点？

在判断相交的过程中要分别遍历两个链表，同时记下各自的长度。然后再遍历一次：长链表节点先从头节点出发前进(lengthMax-lenghMin)步，之后两个链表同时前进，每次一步，相遇的第一个节点即为两个链表相交的第一个节点。程序描述如下：

Node \*intersection(Node \*head1, Node \*head2) {

if(!head1 || !head2) {

return NULL;

}

int len1 = 1;

int len2 = 1;

bool result = false;

//判断两个链表是否相交，同时记下各个链表的长度

Node \*p = head1;

while(p->next) {

pLen++;

p = p->next

}

q = head2

while(q->next) {

len2++;

q = q->next

}

/\*

They are equals then result is true that means have the same nodes;

else is false that means don't have the same nodes.

\*/

result = (p == q);

if(result){

//To get the abs length distance

int steps = abs(len1 – len2);

//To get the max length's Node

Node \*head = len1 > len2 ? head1 : head2;

// the lengther Node go steps

while(steps) {

head = head->next

steps--

}

//go steps the same step

len1 > len2 ? p = head, q = head2 ? q = head, p = head1

while(p != q) {

p = p->next;

q = q->next;

}

return p;

}

return NULL;

}

补充5、判断两个带有环的链表是否相交？

情况一、两个链表在环上相交：

情况二、两个链表从非环处开始相交：

无论是哪种情况，对于环上的任意一个节点，都可以通过head1或者head2到达。