先说说我的理解：

设出发点到入环点距离为a，入环点到相遇点距离为b。设环的长度为c。

当slow走了a，则fast走了2\*a，fast已经入环且处于环中的第a%c位置(设入环点为0)。 Fast要追上slow，还差c-a%c的距离，显然两者每移动一次能追上1距离，因此还需要移动c-a%c次。可以看出，b=c-a%c， 即c-b = a%c。令a=k\*c+r, r<c， 则c-b=r，即a=k\*c+c-b。而 fast和slow相遇后，slow正好还需要走c-b步才能回到入环点。因此如果此时head开始一步一步走，slow也一步一步走，两者必将在入环点处相遇。(对于那个式子，还可以理解成，head与slow以相同速度，head走完a这段路程，slow正好走完k\*c+c-b这段路程，而此时两者正好都在入环点，相遇)

[Linked List Cycle II](http://blog.csdn.net/sysucph/article/details/15378043)

分类： [面试算法题目解答](http://blog.csdn.net/sysucph/article/category/1685629)2013-11-11 20:23 1040人阅读 [评论](http://blog.csdn.net/sysucph/article/details/15378043#comments)(6) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/sysucph/article/details/15378043#report)

Given a linked list, return the node where the cycle begins. If there is no cycle, return null

Follow up:

Can you solve it without using extra space?

给定一个链表的头指针，问你能不能只用常数的空间快速判断一个链表是不是有环，如果有环，返回环的起始位置。

eg：比如 A->B->C->D->E->C返回C的指针

/\*\*  
 \* 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) {  
        // IMPORTANT: Please reset any member data you declared, as  
        // the same Solution instance will be reused for each test case.         
    }  
};

对于判断链表是否有环，方法很简单，用两个指针，一开始都指向头结点，一个是快指针，一次走两步，一个是慢指针，一次只走一步，当两个指针重合时表示存在环了。

证明：假设链表有环，环的长度为N,慢指针在起始位置，快指针在位置k（位置从0开始计数），那么快指针只要比慢指针多走经过N-k步，就可以追上慢指针了。。。，因为每一次快指针都比慢指针多走一步，所以一定可以在有限的步数追上慢指针。

现在的问题是如何求出环的起始位置，我们先给出结论：当快指针和慢指针重合的时候，把一个指针重新指向头指针，两个指针现在速度一样，一次走一步，那么当两个指针值相同时，所在的指针就是我们要找的起始位置。

证明内容来自一个国外的网站：<http://umairsaeed.com/2011/06/23/finding-the-start-of-a-loop-in-a-circular-linked-list/>

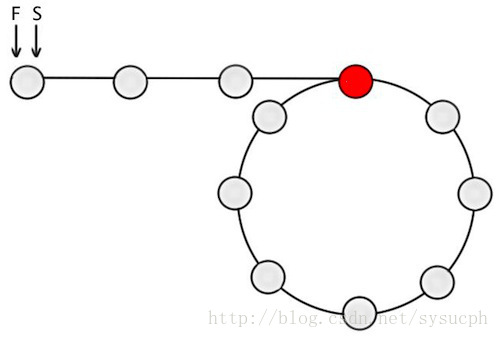
要证明上面这个结论，我们先证明一个结论，假设慢指针Slow停在环的起始位置时，快指针Fast停在第k个位置，那么两个指针相遇时会停在从起始位置倒数第k个位置，也就是n-k这个位置

从Slow停在环的起始位置，假设最终停在n-x相遇。

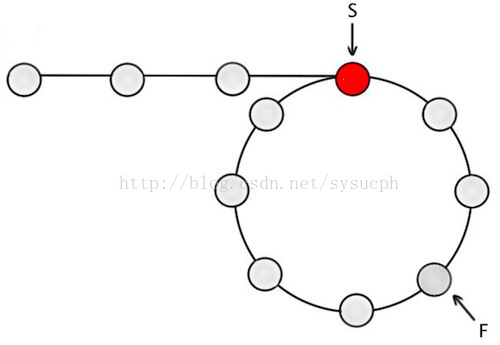
因为Fast只要再比Slow多走n-k步，就可以追上Slow，而每一次操作Fast都比Slow多走一步，因为只需要再走n-k步就可以追上Slow，此时Slow停在n-k这个位置。

这意味着，如果Slow从环的起始位置，Fast从环的第k个位置开始，最终两个指针会在n-k这个位置相遇（n为环的长度）

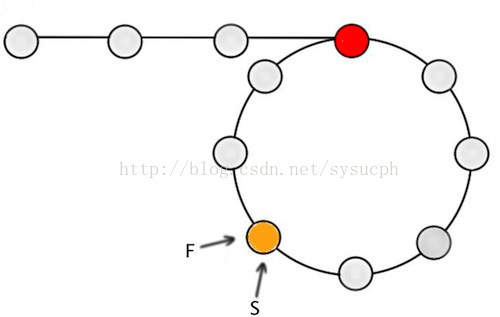
假设一开始Fast和Slow从开始位置开始遍历这个链表。



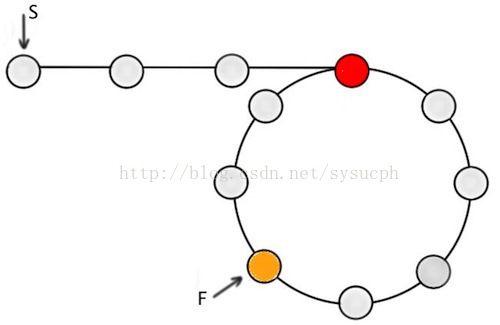
令m = 3，表示经过三步，Slow结点到达环的起始位置，此时Fast在环的第m个位置，因为Fast比Slow多走了m步



根据刚才的结论，当Slow停在起始位置，Fast停在m位置，两个链表最后会在n-m位置相遇



此时把Slow移到头结点位置，两个结点都是要经过m步，才刚刚好到达环的起始位置。



这里看起来好像m小于环的长度l，才成立。其实是一样的。

假设m = t\*l+k

这就是说，环最终停在n-k这个位置了。还需要k步就可以到达环的起始结点。而把Slow结点重新设置为头结点，则需要t\*l+k步才第一次到达环的起始结点，

但是注意了，多走了k\*l步，Fast结点还是会停在环的起始位置的。

得证！！！

/\*\*  
 \* 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) {  
        // IMPORTANT: Please reset any member data you declared, as  
        // the same Solution instance will be reused for each test case.  
        if(head==NULL)return NULL;  
        if(head->next==NULL)return NULL;  
        if(head->next->next==NULL)return NULL;  
        ListNode \*pre = head->next;  
        ListNode \*nxt = pre->next;  
        while(1){  
            if(pre==nxt)break;  
            if(pre->next==NULL)return NULL;  
            pre = pre->next;  
            if(nxt->next==NULL)return NULL;  
            nxt = nxt->next;  
            if(nxt->next==NULL)return NULL;  
            nxt = nxt->next;  
        }  
        pre = head;  
        while(pre!=nxt){  
            pre = pre->next;  
            nxt = nxt->next;  
        }  
        return pre;  
    }  
};

和室友大神lyc交流，大神的想了一会，秒了= =！

大神的解释是这样的：假设环的长度为l，从开始到两个指针第一次相遇总共走了m步，那么Fast指针走了2\*m步，Slow指针走了m步，Fast比Slow多走了m步，Fast比Slow多走了x圈，那么有x\*l==m，l是m的一个因子，再走m步，两个指针每次走一步，Fast移到开头位置，Slow走了m步停在第一次相遇位置，Fast因为也是每次走一步，所以也会停在相遇位置，而且可以看出来，一旦Fast进入环,Fast和Slow结点就保持相对静止了。。。碉堡了。。春哥