142. Linked List Cycle II

Tags	fun medium
Property	@September 13, 2022

Question

原文：

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

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to (**0-indexed**). It is `-1` if there is no cycle. **Note that** `pos` **is not passed as a parameter**.

Do not modify the linked list.

我的理解：

給定一個head（liked list），return 循環開始的節點 假如這個 linked list 沒有 loop 則 return NULL

假如是一個有 loop 的 linked list 會有一些 node 被重複到達（估計是被指標指到的意思），pos 表示 loop 發生的位置也就是 linked list 尾巴的陣列連接的node 會在(0~尾巴之間)，pos = -1表示無 loop ，pos並不是一個參數。

請不要更改 linked list 的值。

翻譯：

给出一个链表的 "头"，返回*循环开始的节点。如果没有循环，返回* `null`。

如果列表中存在一些节点，可以通过连续跟踪`下一个`指针来再次到达，那么链接列表中就存在一个循环。在内部，`pos`用来表示尾巴的`next`指针所连接的节点的索引（**0-索引**）。如果没有循环，它就是`-1'。**注意**`pos'**不作为参数传递。

不要修改链表。

自評翻譯正確性：90%

- Word Memory：
  - reached 達到

- denote 表示

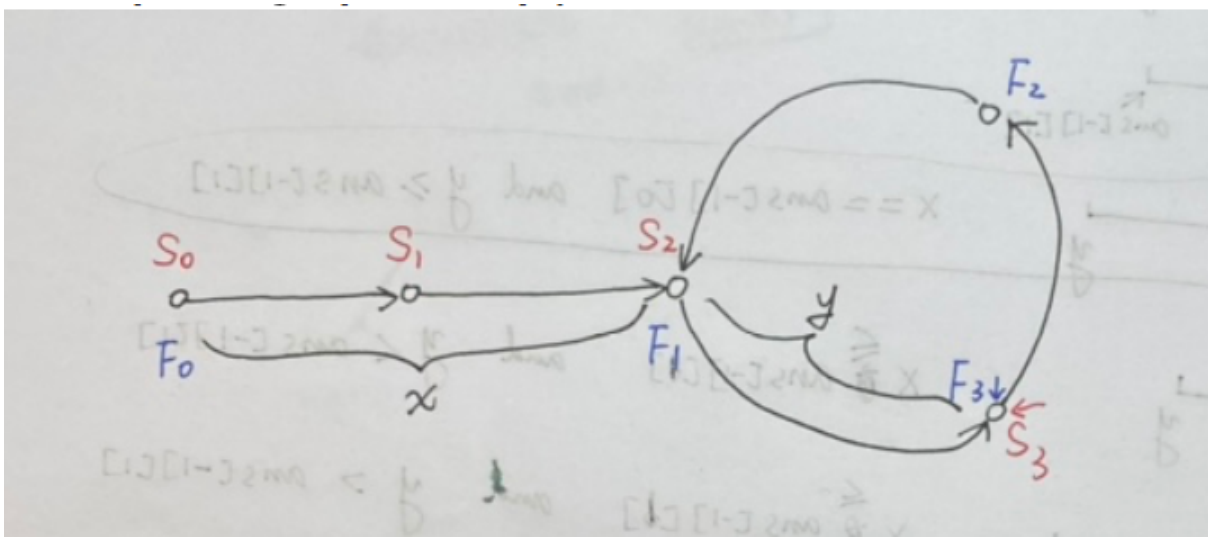
Code

```
/**
 * 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) {
        ListNode * slow;//慢指標
        ListNode * fast;//快指標

        slow=head;
        fast=head;

        while(fast&&fast->next!=NULL){//找出第一個快慢指標相遇的點，能找到表示有loop，找不到表示沒有
            slow=slow->next;
            fast=fast->next->next;

            if(slow==fast){
                break;
            }
        }
        if((fast==NULL)||(fast->next==NULL)){//while結束時 fast or fast->next 等於 NULL 表示無loop
            return NULL;
        }
        ListNode * P1;
        ListNode * P2;

        P1=head;
        P2=fast;//P2直接到相遇點

        while(P1!=P2){//此時 P1 P2同時向前，它們的相遇點，就是linked list 尾巴的next也就是loop的起點
            P1=P1->next;
            P2=P2->next;
        }
        return P1;
    }
};
```

> 思路：參考https://blog.csdn.net/a130737/article/details/44226413與下方優良code后理出；先建立快慢兩種指標，慢指標一次走一步，快指標一次走兩步，如果有迴圈的狀況發生，快指標終究會追上慢指標，等兩個指標重合，記錄下該位置，再開兩個座標P1P2（其實可沿用slow

> fast，但為了區分功能才新開）P1從起點開始，P2從相遇點開始，兩指標的相遇點就是loop發生的位置。

> 紅字原因解釋：
>
> - 考下圖 x 為起點至loop發生位置、y為loop發生位置至相遇點的距離、C是循環長度→loop發生位置走到底再走回來loop發生位置的步數
> - slow走的長度為x+y，而fast走的長度是2(x+y)
> - fast比slow多的路長，多x+y這麼長，且fast只走循環的段落，總長也就是C的倍數，假設走了N次，可得 x+y=N*C
> - 從定義可知，P1只要從起點走x步可以抵達loop發生位置；P2繼承fast的位置，也就是已經走了從loop到相遇點也就是y，再走x步就會再達到一次x+y也就是在跑完一次N*C，而因為從loop發生位置開始只要是走C步都會回到loop，所以有N個C步也是回到loop發生位置
> - 由第四點可知P1和P2的相遇點就是剛好是loop發生位置，並且在各自步伐相同走了x步之後



Success Details ›

Runtime: 8 ms, faster than 87.41% of C++ online submissions for Linked List Cycle II.

Memory Usage: 7.4 MB, less than 99.10% of C++ online submissions for Linked List Cycle II.

Next challenges:

Linked List Cycle | Find the Duplicate Number

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
09/12/2022 17:58	Accepted	8 ms	7.4 MB	cpp

優良code參考

```
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        ListNode *slow = head, *fast = head;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast) break;
        }
        if (!(fast && fast->next)) return NULL;
        while (head != slow) {
            head = head->next;
            slow = slow->next;
        }
        return head;
    }
};
```

> 思路：同上