OneStepOffer 算法第四讲

Stack和Queue的巧用2

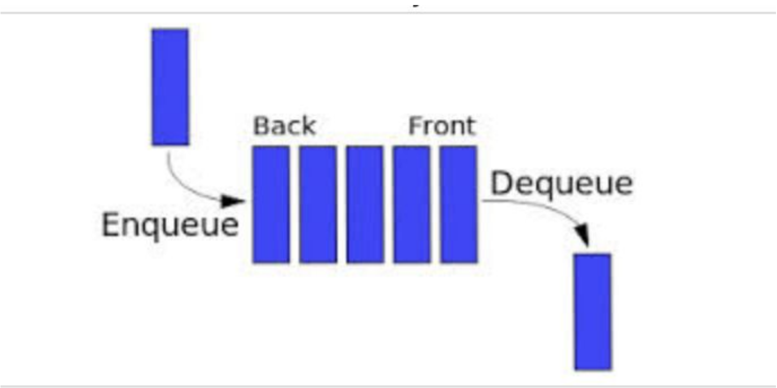
目录

1. Queue的说明，常见Queue -- 链表
2. Linkedlist的结构和底层实现
3. 例题
4. 一些练习题

Queue(队列)的定义以及实现

Queue -- First In Firs Out

常见Queue: 链表 LinkedList



LinkedList 链表的定义

链表是由节点和指针构成的数据结构，每个节点存有一个值，和一个指向下一个节点的指针，因此很多链表问题可以用**递归**来处理。

列表形式

```
class ListNode {
    int val;
    ListNode next;
    ListNode(int x) {
        this.val = x;
    }
}
```

Linkedlist 在java中基本的储存形式

linkedList在java中是以double linked list 双链表形式存储的

```
class Node<E> {
        E item;
        Node<E> next;
        Node<E> prev;
        Node(Node<E> prev, E element, Node<E> next) {
            this.item = element;
            this.next = next;
            this.prev = prev;
        }
}
```

linkedlist 中offer, remove 和peek的实现

```
public class LinkedList<E> {
    int size = 0;
    Node<E> first; // pointed to first node
    Node<E>last; // pointed to last node
    public E peek() {
        final Node<E> f = first;
        return (f == null) ? null : f.item;
    }
    public boolean offer (E e) {
        final Node<E> l = last;
        final Node<E> newNode = new Node(l, e, null);
        last = newNode;
        If (l == null) first = newNode;
        else l.next = newNode;
        return true;
     }
```

```
  public E poll() {
      final Node<E> f = first;
      return (f == null) ? null : unlinkFirst(f);
  }

  private E unlinkFirst(Node<E> f) {
     final E element = f.item;
     final Node<E> next = f.next;
     f.item = null;
     f.next = null;
     if (next == null) last = null;
     else next.prev = null;
     return element
  }
}
```

LinkedList 例题 - 翻转链表

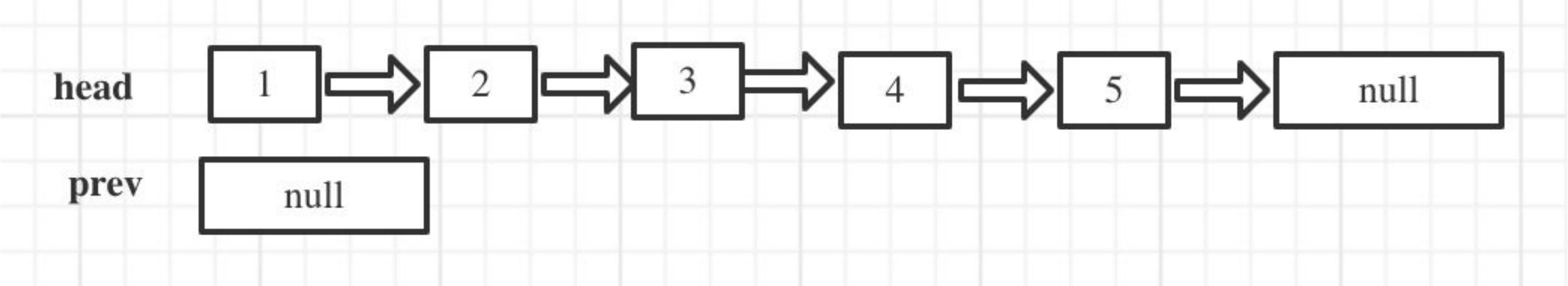
输入一个链表，输出该链表翻转后的结果。

Input: 1 → 2 → 3 → 4 → 5 → null

Output: 5 → 4 → 3 → 2 → 1 → null

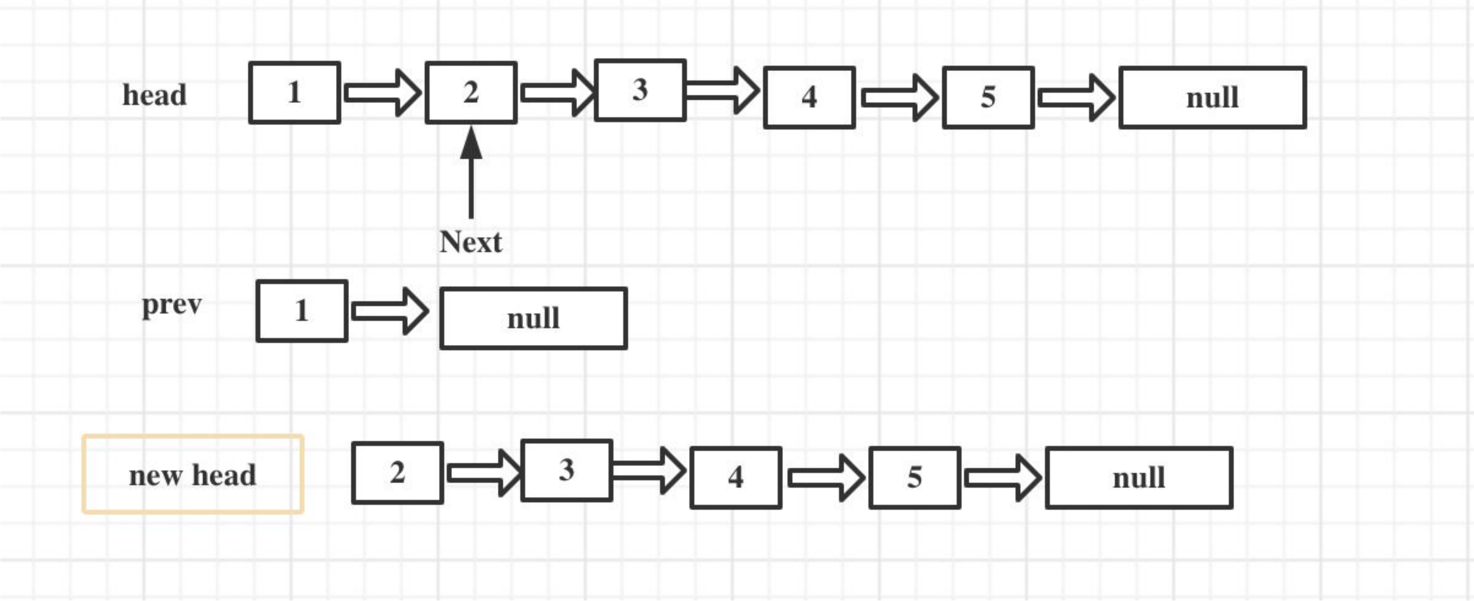
解题思路1: 递归

首先 创立一个新的链表 prev作为我们最后的结果



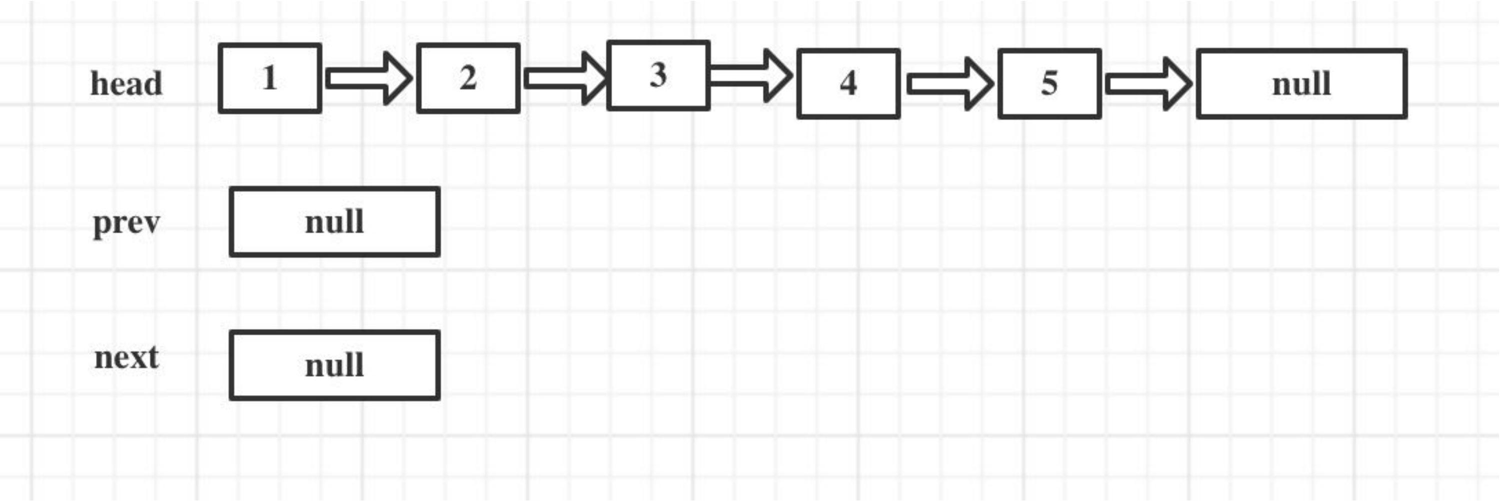
解题思路1 -- 递归

建立新指针（注意不是新链表）指向next, 作为新的head，并把旧head的next指向prev作为新prev



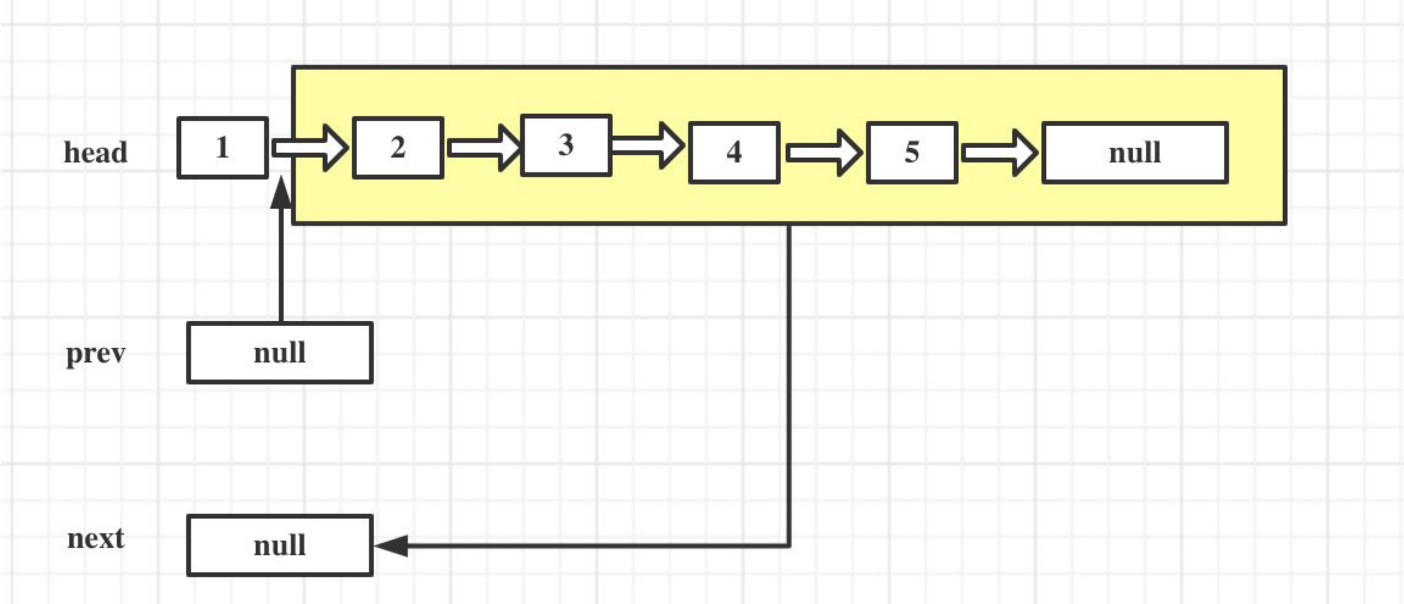
解题思路2: 非递归

非递归需要用到while loop, 建立两个新的链表next和prev



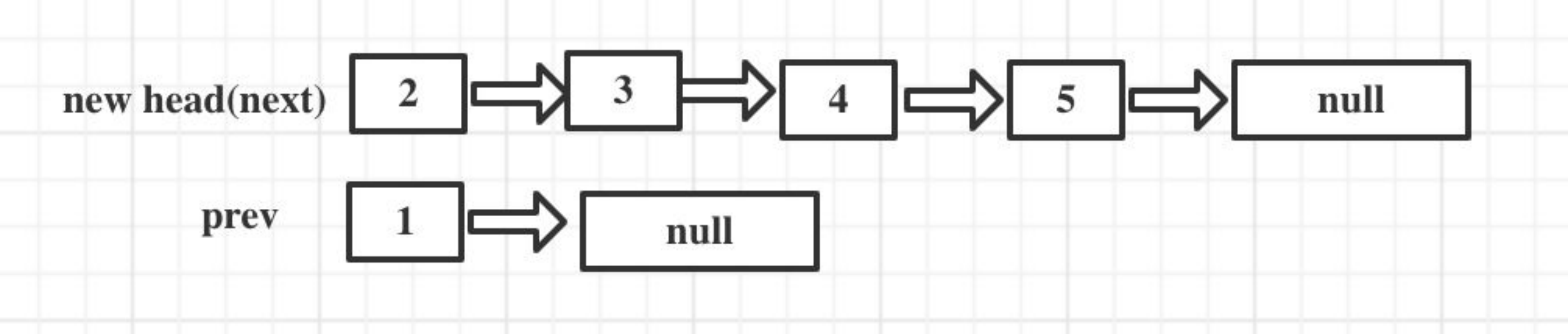
解题思路2: 非递归

把旧的head.next 转移到next里作为新header, prev 放新的head. 转移过程如下



解题思路2:非递归

经过此番操作后，新的header, prev, next 如下



递归和非递归 - Java

```
ListNode reverseList (ListNode head, ListNode prev) {

   if (head == null) return prev;

   ListNode next = head.next;

   head.next = prev;

   return reverseList(next, head);

}
```

```
ListNode reverseList (ListNode head) {
        ListNode prev = new ListNode();
        ListNode next = new ListNode();
       while (head != null) {
            next = head.next;
            head.next = prev;
            prev = head;
            head = next;
       }
      return prev;
}
```

递归和非递归 - c++

```cpp
ListNode* reverseList(ListNode* head, ListNode*prev=nullptr) {
    if (!head) {
        return prev;
    }
    ListNode* next = head->next;
    head->next = prev;
    return reverseList(next, head);
}
```

```cpp
ListNode* reverseList(ListNode* head) {
    ListNode *prev = nullptr, *next;
    while (head) {
        next = head->next;
        head->next = prev;
        prev = head;
        head = next;
    }
    return prev;
}
```

递归和非递归- python

```
class Solution:
# @param {ListNode} head
# @return {ListNode}
def reverseList(self, head):
    return self._reverse(head)

def _reverse(self, node, prev=None):
    if not node:
        return prev
    n = node.next
    node.next = prev

       return self._reverse(n, node)
```

```
class Solution:
# @param {ListNode} head
# @return {ListNode}
def reverseList(self, head):
    prev = None
    while head:
        curr = head
        head = head.next
        curr.next = prev
        prev = curr
     return prev
```

例题2: Swap Nodes in Pairs

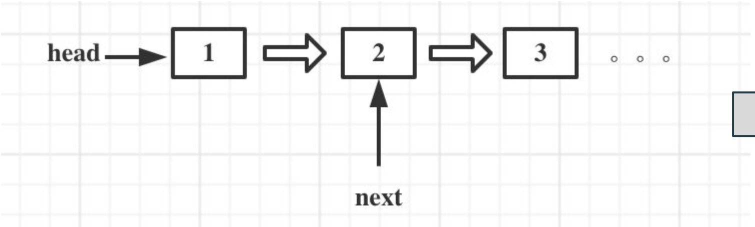
题目描述：给定一个矩阵，交换每个相邻的一对节点。

Input: 1->2->3->4

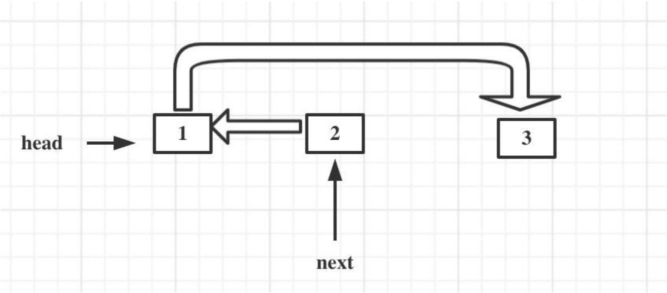
Output: 2->1->4->3

思路分析和问题拆解

1. 如何利用指针交换两个链表



起始链表



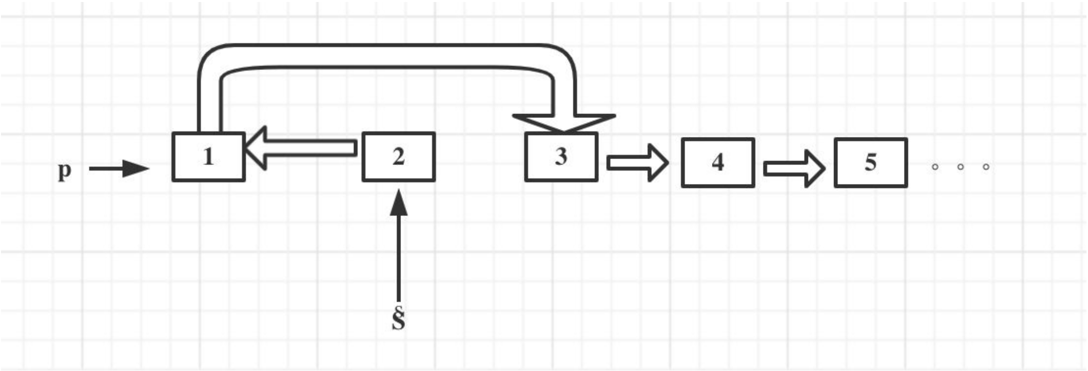
最终链表

利用指针交换链表 - pseudo code

```
Input: LinkedList head

LinkedList p = head; // First Index

LinkedList s = p.next; // Second Index

p.next = s.next; (head.next.next) // Set next of p to be next of s

head = s ; // let head = s

Return head;
```

2. 递归链表

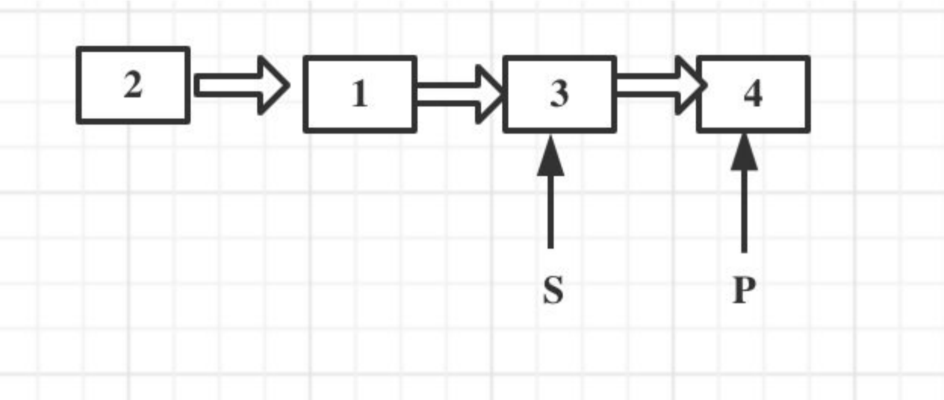
当我们确定p和s的指针后，如何继续递归下去？



第一次交换后的指针方向

递归链表

通过改变s和p的位置 检测是否存在递归的必要性，进而进行递归



递归链表 - Pseudo Code

```
While (p.next not null && p.next.next not null)
    s = p.next.next  // set s to the 4th place
    p.next.next  =  s.next // set p to the 3rd place
    s.next = p.next
     p.next = s;
     p=s.next;
```

Swap s and p

Java 答案

```
ListNode swapPairs(ListNode head) {
    ListNode p = head;
    ListNode s = p.next;
    if (p != null && p.next != null) {
      p.next = s.next;
      s.next = p;
      head = s;
      while (p.next != null && p.next.next != null) {
        s = p.next.next;
        p.next.next = s.next;
        s.next = p.next;
        p.next = s;
        p=s.next;
      }
    }
    return head;
  }
```

C ++ 答案

```
ListNode* swapPairs(ListNode* head) {
        ListNode *p = head, *s;
        if (p && p->next) {
                s = p->next;
                p->next = s->next;
                s->next = p;
                head = s;
                while (p->next && p->next->next) {
                        s = p->next->next;
                        p->next->next = s->next;
                        s->next = p->next;
                        p->next = s;
                        p = s->next;
                }
        }
        return head;
}
```

Python 答案

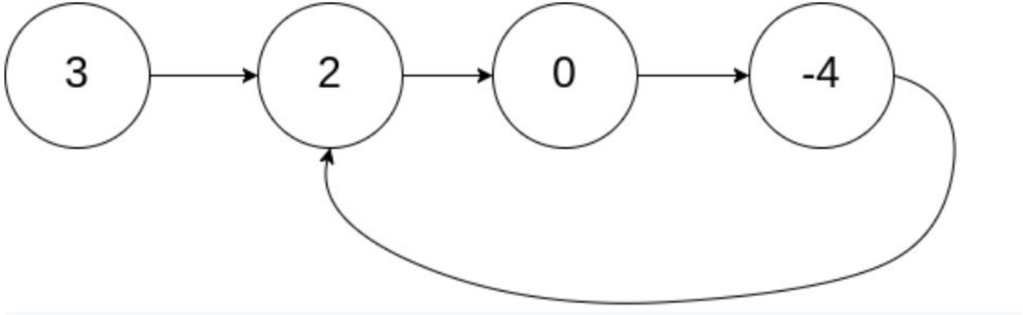
```python
class Solution(object):
    def swapPairs(self, head):
        if not head or not head.next: return head
        dummy = ListNode(0)
        dummy.next = head
        cur = dummy

        while cur.next and cur.next.next:
            first = cur.next
            sec = cur.next.next
            cur.next = sec
            first.next = sec.next
            sec.next = first
            cur = cur.next.next
          return dummy.next
```

例题3 LinkedList Cycle

题目：给定一个链表，返回循环开始的节点。如果没有循环，则返回null。如果链表中有某个节点，则可以通过连续跟随下一个指针来再次访问该节点，这将导致一个循环。在内部，pos用于表示尾部的下一个指针所连接的节点的索引。os不作为参数传递,不能修改链接列表。

Input: head = [3,2,0,-4], pos = 1

Output: tail connects to node index 1



分解题目

首先思考，如何判断linkedlist是否存在cycle -- 快慢指针法

快指针一次走两个node, 慢指针一次走一个node 直到相遇

第一步 快慢指针 pseudo code

```
input: LinkedList head;
output: boolean

if head is null or head.next is null return false;
LinkedList fast = head;
LinkedList slow = head;
while (fast.next not null && fast.next.next != null) {
        slow = slow.next();
        fast = fast.next.next();
        if (fast == slow) return true;
}
return false;
```

第二步，找到出现循环的第一个node

找到是否为循环链表后，找到第一个出现循环的node

令快/慢指针指向head, 向后移动指针，直到找到相同node

找到第一个循环node

前提 : fast = slow = 链表中某一个node

```
fast = head ;
while (fast != slow) {
      fast = fast.next
      slow = slow.next
 }
return fast
```

C++ 代码

```
ListNode *detectCycle(ListNode *head) {
    if (head == NULL || head->next == NULL) return NULL;

    ListNode* firstp = head;
    ListNode* secondp = head;
    bool isCycle = false;

    while(firstp != NULL && secondp != NULL) {
        firstp = firstp->next;
        if (secondp->next == NULL) return NULL;
        secondp = secondp->next->next;
        if (firstp == secondp) { isCycle = true; break; }
    }

    if(!isCycle) return NULL;
    firstp = head;
    while( firstp != secondp) {
        firstp = firstp->next;
        secondp = secondp->next;
    }

    return firstp;
  }
```

Java 代码

```java
public ListNode detectCycle(ListNode head) {
    ListNode slow = head;
    ListNode fast = head;
    while (fast!=null && fast.next!=null){
         fast = fast.next.next;
         slow = slow.next;

          if (fast == slow){
               ListNode slow2 = head;
               while (slow2 != slow){
                     slow = slow.next;
                     slow2 = slow2.next;
               }
               return slow;
           }
      }
       return null;
}
```

Python 代码

```python
def detectCycle(self, head):
    slow = fast = head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow == fast:
            break
    else:
        return None
    while head != slow:
        slow = slow.next
        head = head.next
    return head
```

练习题

1. Merge 2 sorted Lists -- (Easy): 给定两个增序的链表，试将其合并成一个增序的链表。
2. Palindrome Linked List -- (Easy): 以 O(1) 的空间复杂度，判断链表是否回文。
3. Sort Linked List -- (Medium): 利用快慢指针找到链表中点后，可以对链表进行归并排序。
4. Odd Even LinkedList -- (Medium): 给定一个单链表，将所有奇数偶数节点分别组合在一起。
5. Max Points on a Line -- (Hard): 给定2D平面上的n个点，求出同一直线上的最大点数。