# p1.#206Problem: Reverse Linked List

## Description:

Reverse a singly linked list.

## Solution 1: store every data in the list into program

### Idea

要返着输出一个单链表。 单链表顾名思义从后一个node 没有办法找到前一个node 最先想到的办法就是遍历两遍此链表。 第一遍将链表中的数据读取出来并存入list，第二遍挨个赋值（从list 的尾向头赋值）。

* Step 1: process the head
* Step 2: walk through the given linked list and store into a local Arraylist (nodelist)
* Step 3: Reassign the given linked list using nodelist data from the end to the beginning

### Boundary process

Head:

if(head==null) return head;

tail:

无

## Solution 2: iterative solution

### Idea

因为单链表的特性，使得每一个节点难以找到前一个节点的位置，但是非常容易找到下一个节点的位置。所以反转单链表的操作是将当前节点中存储的next的位置，指向前一个节点。 一共使用三个指针： preNode 当前操作节点的前一个节点 currNode 当前要操作的节点 nextNode 当前操作节点的后一个节点

* Step 1: process the head
* Step 2: iterative loop
* Step 3: process the tail

### Boundary process

Head:

if(head == null || head.next == null) return head;

tail:

head.next = null; 输入的头节点成为了尾节点

## Solution 3: recursive solution

递归方法写起来确实代码量少。一共使用两个指针： A. 每次递归处理传入的一个节点 head 使 head.next.next = head 为了保证最后返回的链表结尾为null 每次都使 head.next = null B. 为了保证不丢弃新链表的头部（即旧链表的末尾节点） 每次返回值都为旧链表的末尾节点 。

* Step 1: process the head
* Step 2: call reverseList method (itself)
* Step 3: process pointer: set the input node's next node's pointer to the input node

### Boundary process

Head:

if(head == null || head.next == null) return head;

tail:

无

## Solutions compare:

## 单链表反转。有三种方法 ：

## 将链表的内容储存到本地。 该法效率最低 属于投机取巧 无技巧可言 浪费内存

## Iterative solution 反转链表指针的指向。 该法效率最高 正统优解

1. Recursive solution 递归实现 。该法思路值得学习 但是浪费内存 由于栈维护了每个函数调用的信息直到函数返回后才释放，这需要占用相当大的空间，尤其是程序中使用了许多递归调用的情况下。除此之外，因为有大量信息需要保存和恢复，因此生成和销毁栈也需要耗费一定的时间。所以此方法并不推荐。

## Related knowledge

### Recursive vs iterative

For recursive method, there is a significant cost in calling a function - rolling and unrolling stacks. Iterative solutions have better performance. But recursive solution helps ppl to think in a functional way. Many company interviews test this. Functional solution turns out to be much simpler and concise.