# 题目

给你一个链表，每 k 个节点一组进行翻转，请你返回翻转后的链表。

k 是一个正整数，它的值小于或等于链表的长度。

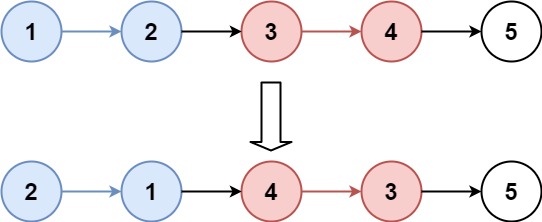
如果节点总数不是 k 的整数倍，那么请将最后剩余的节点保持原有顺序。

进阶：

你可以设计一个只使用常数额外空间的算法来解决此问题吗？

你不能只是单纯的改变节点内部的值，而是需要实际进行节点交换。

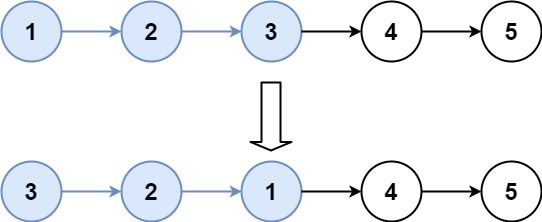
示例 1：



输入：head = [1,2,3,4,5], k = 2

输出：[2,1,4,3,5]

示例 2：



输入：head = [1,2,3,4,5], k = 3

输出：[3,2,1,4,5]

示例 3：

输入：head = [1,2,3,4,5], k = 1

输出：[1,2,3,4,5]

示例 4：

输入：head = [1], k = 1

输出：[1]

提示：

列表中节点的数量在范围 sz 内

1 <= sz <= 5000

0 <= Node.val <= 1000

1 <= k <= sz

# 分析

## 方法一：遍历

/ \*

\*prev

\*tail head

\*dummy 1 2 3 4 5

\*----------------------------

\*prev head tail

\*dummy 1 2 3 4 5

\* cur

\*----------------------------

\* 每次让prev.next的元素插入到当前tail之后,这样tail不断前移,被挪动的元素头插入tail之后的链表

\*prev tail head

\*dummy 2 3 1 4 5

\* cur

\*----------------------------

\*prev tail head

\*dummy 3 2 1 4 5

\* cur

\*----------------------------

\* prev

\* tail

\* head

\*dummy 3 2 1 4 5

\*----------------------------

\* prev head tail

\*dummy 3 2 1 4 5 null

\*----------------------------

\*/

class Solution {

public:

ListNode\* reverseKGroup(ListNode\* head, int k) {

ListNode\* dummy=new ListNode(0);

dummy->next=head;

ListNode\* pre=dummy;

ListNode\* cur=head;

ListNode\* tail=dummy;

while(true){

int count=0;

tail=pre;

while(tail!=NULL&&count<k){

tail=tail->next; //退出循环后tail指向待反转链表的末尾节点

count++;

}

if(tail==NULL)

break;

while(pre->next!=tail){ //pre->next==tail时退出循环

cur=pre->next;//(1)

pre->next=cur->next; //(1) 步骤(1):将cur从链表中切出来

cur->next=tail->next;//(2)

tail->next=cur; //(2) 步骤(2):将cur添加到tail后

}

pre=head;

tail=head;

head=pre->next;//head指向新的待翻转的链表头

}

return dummy->next;

}

};

## 方法二：递归

class Solution {

public:

ListNode\* reverseKGroup(ListNode\* head, int k) {

if (head == NULL) return NULL;

ListNode \*a = head;

ListNode \*b = head;

for (int i = 0; i < k; i++) {

if (b == NULL) return head;

b = b->next;

}

ListNode \*newNode = reverseOperator(a,b);

a->next = reverseKGroup(b,k);

return newNode;

}

ListNode\* reverseOperator(ListNode\* n,ListNode \*b) {

ListNode \*pre, \*cur, \*nxt;

pre = NULL; cur = n; nxt = n;

while (cur != b) {

nxt = cur->next;

cur->next = pre;

pre = cur;

cur = nxt;

}

return pre;

}

};

注释：

class Solution {

public:

ListNode\* reverseKGroup(ListNode\* head, int k) {

if(head==nullptr) return nullptr;//递归的结束条件

ListNode\*a=head;

ListNode\*b=head;

for(int i=0;i<k;i++){

//下面是递归时一条链接的情况

if(b==nullptr)return head;//说明不是k的整数倍，直接返回此时的head

b=b->next;//把b移动到k的前一个位置

}

//把a到b之间的节点反转，不算b

ListNode\*newNode = reverseOperator(a,b);

a->next= reverseKGroup(b,k);//递归调用本函数，这个太难想到了

return newNode;

}

ListNode\* reverseOperator(ListNode\* n, ListNode\* b ){

ListNode\* pre=nullptr;

ListNode\*cur=n;

ListNode\*nxt=n;

//循环条件就限定了只反转b之前的节点

while(cur!=b){

nxt=cur->next;

cur->next=pre;

pre=cur;

cur=nxt;

}

return pre;

}

};

另一种写法：

class Solution {

public:

ListNode\* reverseKGroup(ListNode\* head, int k) {

ListNode\* pre=head;

int count=0;

while(pre!=NULL&&count<k){

pre=pre->next;

count++;

} //退出循环后pre指向第k+1个节点

if(count==k) {

pre=reverseKGroup(pre,k);

while(count>0){

/\* head指向本次循环反转指向的节点

temp指向下次循环反转指向的节点

\*/

ListNode\* temp=head->next;

head->next=pre;

pre=head;

head=temp;

count--;

}

head=pre;

}

return head;

}

};