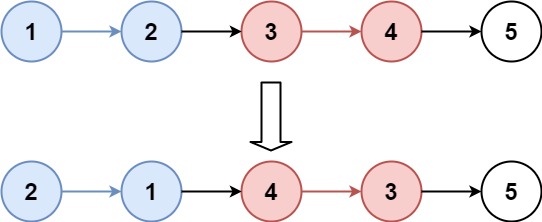
**Reverse Nodes in k-Group**

Given a linked list, reverse the nodes of a linked list *k* at a time and return its modified list.

*k* is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of *k* then left-out nodes, in the end, should remain as it is.

You may not alter the values in the list's nodes, only nodes themselves may be changed.

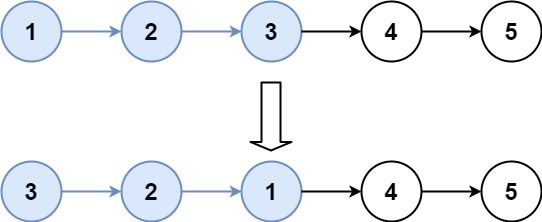
**Example 1:**



**Input:** head = [1,2,3,4,5], k = 2

**Output:** [2,1,4,3,5]

**Example 2:**



**Input:** head = [1,2,3,4,5], k = 3

**Output:** [3,2,1,4,5]

**Example 3:**

**Input:** head = [1,2,3,4,5], k = 1

**Output:** [1,2,3,4,5]

**Example 4:**

**Input:** head = [1], k = 1

**Output:** [1]

**Constraints:**

* The number of nodes in the list is in the range sz.
* 1 <= sz <= 5000
* 0 <= Node.val <= 1000
* 1 <= k <= sz

SOLUTION

class Solution {

public ListNode reverseKGroup(ListNode head, int k) {

if(head==null || k==1)

return head;

ListNode fake=new ListNode(0);

fake.next=head;

ListNode prev =fake;

int i=0;

ListNode p=head;

while(p!=null){

i++;

if(i%k==0){

prev = reverse(prev,p.next);

p=prev.next;

}else{

p=p.next;

}

}

return fake.next;

}

public ListNode reverse(ListNode prev , ListNode next){

ListNode last = prev.next;

ListNode curr = last.next;

while(curr!=next){

last.next = curr.next;

curr.next = prev.next;

prev.next = curr;

curr = last.next;

}

return last;

}

}