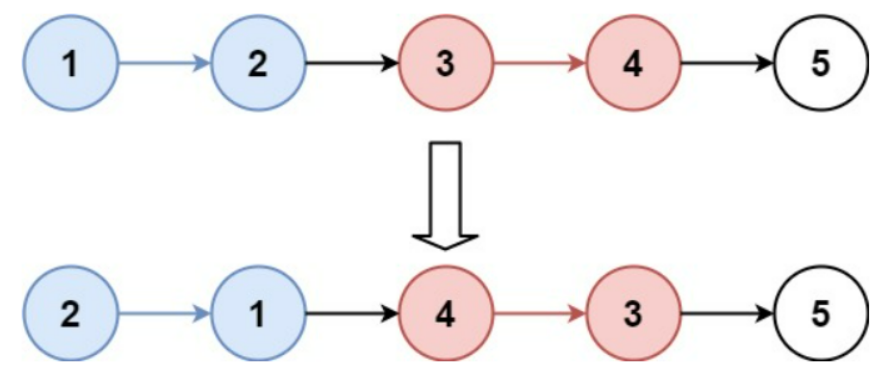
<https://leetcode.com/problems/reverse-nodes-in-k-group/>

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

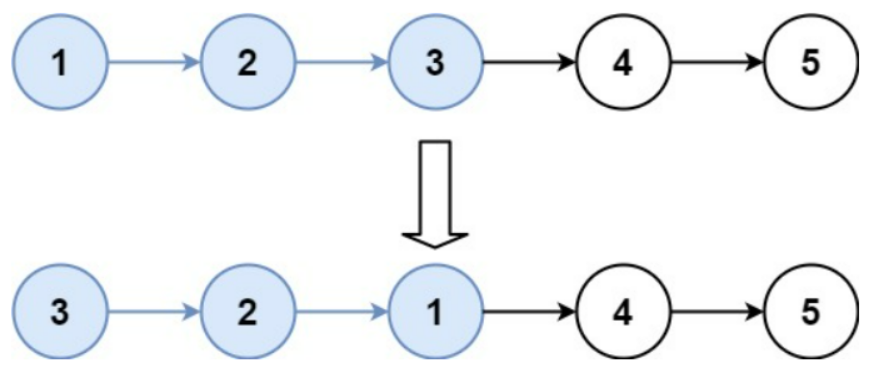
**Example 1:**



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]

**Constraints:**

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

**Follow-up:** Can you solve the problem in O(1) extra memory space?

**Attempt 1: 2023-02-12**

**Solution 1:  Iterative Solution (120 min, must remember the solution)**

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode reverseKGroup(ListNode head, int k) {

ListNode dummy = new ListNode();

dummy.next = head;

ListNode iter = dummy;

while(iter != null) {

ListNode node = iter;

// First check whether there are k nodes to reverse in current iteration

int i = 0;

while(i < k && node != null) {

node = node.next;

i++;

}

if(node == null) {

break;

}

// Now we know that we have k nodes, we will start from the first node

ListNode prev = null;

ListNode cur = iter.next;

ListNode next = null;

int j = 0;

while(j < k) {

next = cur.next;

cur.next = prev;

prev = cur;

cur = next;

j++;

}

ListNode tail = iter.next;

tail.next = cur;

iter.next = prev;

iter = tail;

}

return dummy.next;

}

}

**Refer to**

<https://leetcode.com/problems/reverse-nodes-in-k-group/solutions/183356/java-o-n-solution-with-super-detailed-explanation-illustration/>

This problem can be split into several steps:

Since we need to reverse the linked-list every k nodes, we need to check whether the number of list nodes are enough to reverse. Otherwise, there is no need to reverse.

If we need to reverse the k nodes, how to do that? Following is my idea:

If the structure of the linkedlist is like this:

1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7

Then there will always be a pointer, which points to the node **AHEAD** of the first node to reverse. The pointer will help to link the linkedlist after.

At first, we will add a dummy node in front of the linked list to act as the first pointer. After we add the pointer, the linked list will look like this:

0 (pointer) -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7

Suppose that there are enough nodes to be reversed, we just use the "reverse linked list" trick to reverse the k nodes. Please refer to "<https://leetcode.com/problems/reverse-linked-list/>" if you don't know how to reverse a linked list.

if k = 3, we can reverse 1 to 3 first using the following code:

ListNode prev = null, curr = pointer.next, next = null;

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

next = curr.next;

curr.next = prev;

prev = curr;

curr = next;

}

This is the illustration of the first 3 steps:

step1: 0 (pointer) -> 1 2 -> 3 -> 4 -> 5 -> 6 -> 7

step2: 0 (pointer) -> 1 <- 2 3 -> 4 -> 5 -> 6 -> 7

step3: 0 (pointer) -> 1 <- 2 <- 3 4 -> 5 -> 6 -> 7

This is an easy and general algorithm to reverse a linked list. However, if you are careful enough, you will find that after the for-loop, the link from 3 to 4 will be cut (as shown in step3).

Now we need to reconstruct the linked list and fix the issue. You will figure out that at step3, the 3 is the **prev** node, 4 is the **curr** node.

step3: 0 (pointer) -> 1 <- 2 <- 3 (prev) 4 (curr) -> 5 -> 6 -> 7

We can fix the sequence based on the following codes. The basic idea is to link the pointer to 3 and link 1 to 4:

ListNode tail = pointer.next;

tail.next = curr;

pointer.next = prev;

pointer = tail;

Then the result is:

after first line: 0 (pointer) -> 1 (tail) <- 2 <- 3 (prev) 4 (curr) -> 5 -> 6 -> 7

after second line: 0 (pointer) -> 1 (tail) <- 2 <- 3 (prev) 4 (curr) -> 5 -> 6 -> 7

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_↑

after third line:

|-------------------------------↓

0 (pointer) 1 (tail) <- 2 <- 3 (prev) 4 (curr) -> 5 -> 6 -> 7

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_↑

after forth line: 0 -> 3 -> 2 -> 1 (pointer) -> 4 -> 5 -> 6 -> 7

Now we get the new pointer, and we can repeat the process. Note that to retrieve the head, we need to record the first dummy node (0).

Here is the code:

public ListNode reverseKGroup(ListNode head, int k) {

ListNode dummy = new ListNode(0);

dummy.next = head;

ListNode pointer = dummy;

while (pointer != null) {

ListNode node = pointer;

// first check whether there are k nodes to reverse

for (int i = 0; i < k && node != null; i++) {

node = node.next;

}

if (node == null) break;

// now we know that we have k nodes, we will start from the first node

ListNode prev = null, curr = pointer.next, next = null;

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

next = curr.next;

curr.next = prev;

prev = curr;

curr = next;

}

ListNode tail = pointer.next;

tail.next = curr;

pointer.next = prev;

pointer = tail;

}

return dummy.next;

}

**Solution 2:  Recursive Solution (120 min, must remember the solution)**

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode reverseKGroup(ListNode head, int k) {

// Test weather we have more then k node left, if less then k node left we just return head

ListNode iter = head;

int count = 0;

while(count < k) {

if(iter == null) {

return head;

}

iter = iter.next;

count++;

}

// Reverse k node at current level

// 'prev' node point to the answer of sub-problem, represents the new head after reverse on current section

// e.g 1 -> 2 -> 3 -> 4 -> 5, k=2

// In 1st recursion, return 'prev'=5, in 2nd recursion, return 'prev'=4,

// in 3rd recursion, return 'prev'=2

ListNode prev = reverseKGroup(iter, k);

while(count > 0) {

ListNode next = head.next;

head.next = prev;

prev = head;

head = next;

count--;

}

return prev;

}

}

**Refer to**

<https://leetcode.com/problems/reverse-nodes-in-k-group/solutions/11423/short-but-recursive-java-code-with-comments/comments/12134>

This type of question can split into 2 steps to solve ( like reverse in pair , reverse in 3 node ... )

Here is my solution follow this 2 steps and with little more clear naming conventions.

public ListNode reverseKGroup(ListNode head, int k) {

//1. test weather we have more then k node left, if less then k node left we just return head

ListNode node = head;

int count = 0;

while (count < k) {

if(node == null)return head;

node = node.next;

count++;

}

// 2.reverse k node at current level

ListNode pre = reverseKGroup(node, k); //pre node point to the the answer of sub-problem

while (count > 0) {

ListNode next = head.next;

head.next = pre;

pre = head;

head = next;

count = count - 1;

}

return pre;

}