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.

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 nodes, only nodes itself may be changed.

Only constant memory is allowed.

For example,
Given this linked list: `1->2->3->4->5`

For *k* = 2, you should return: `2->1->4->3->5`

For *k* = 3, you should return: `3->2->1->4->5`

Solution 1

Hi, guys! Despite the fact that the approach is recursive, the code is less than 20 lines. :)

```java
public ListNode reverseKGroup(ListNode head, int k) {
    ListNode curr = head;
    int count = 0;
    while (curr != null && count != k) { // find the k+1 node
        curr = curr.next;
        count++;
    }
    if (count == k) { // if k+1 node is found
        curr = reverseKGroup(curr, k); // reverse list with k+1 node as head
        // head - head-pointer to direct part,
        // curr - head-pointer to reversed part;
        while (count-- > 0) { // reverse current k-group:
            ListNode tmp = head.next; // tmp - next head in direct part
            head.next = curr; // preappending "direct" head to the reversed list
            curr = head; // move head of reversed part to a new node
            head = tmp; // move "direct" head to the next node in direct part
        }
        head = curr;
    }
    return head;
}
```

Hope it helps!

written by shpolsky original link here

Solution 2

```
public class Solution {
      public ListNode reverseKGroup(ListNode head, int k) {
          if (head==null||head.next==null||k<2) return head;

          ListNode dummy = new ListNode(0);
          dummy.next = head;

          ListNode tail = dummy, prev = dummy,temp;
          int count;
          while(true){
              count =k;
              while(count>0&&tail!=null){
                  count--;
                  tail=tail.next;
              }
              if (tail==null) break;//Has reached the end


              head=prev.next;//for next cycle
          // prev-->temp-->...--->....--->tail-->....
          // Delete @temp and insert to the next position of @tail
          // prev-->...-->...-->tail-->head-->...
          // Assign @temp to the next node of @prev
          // prev-->temp-->...-->tail-->...-->...
          // Keep doing until @tail is the next node of @prev
              while(prev.next!=tail){
                  temp=prev.next;//Assign
                  prev.next=temp.next;//Delete

                  temp.next=tail.next;
                  tail.next=temp;//Insert

              }

              tail=head;
              prev=head;

          }
          return dummy.next;

      }
  }
```

written by reeclapple original link here

Solution 3

```
-1 -> 1 -> 2 -> 3 -> 4 -> 5
 |    |    |    |
pre  cur  nex  tmp

-1 -> 2 -> 1 -> 3 -> 4 -> 5
 |         |    |    |
pre       cur  nex  tmp

-1 -> 3 -> 2 -> 1 -> 4 -> 5
 |              |    |    |
pre            cur  nex  tmp
```

Above is how it works inside one group iteration(for example, k=3)

```
class Solution {
public:
    ListNode *reverseKGroup(ListNode *head, int k) {
        if(head==NULL||k==1) return head;
        int num=0;
        ListNode *preheader = new ListNode(-1);
        preheader->next = head;
        ListNode *cur = preheader, *nex, *tmp, *pre = preheader;
        while(cur = cur->next)
            num++;
        while(num>=k) {
            cur = pre->next;
            nex = cur->next;
            for(int i=1;i<k;i++) {
                tmp= nex->next;
                nex->next = pre->next;
                pre->next = nex;
                cur->next = tmp;
                nex = tmp;
            }
            pre = cur;
            num-=k;
        }
        return preheader->next;
    }
};
```

Thanks to ciaoliang1992, the tmp pointer is no necessary, so the more concise solution is

```
class Solution {
public:
    ListNode *reverseKGroup(ListNode *head, int k) {
        if(head==NULL||k==1) return head;
        int num=0;
        ListNode *preheader = new ListNode(-1);
        preheader->next = head;
        ListNode *cur = preheader, *nex, *pre = preheader;
        while(cur = cur->next)
            num++;
        while(num>=k) {
            cur = pre->next;
            nex = cur->next;
            for(int i=1;i<k;++i) {
                cur->next=nex->next;
                nex->next=pre->next;
                pre->next=nex;
                nex=cur->next;
            }
            pre = cur;
            num-=k;
        }
        return preheader->next;
    }
};
```

written by luming.zhang.75 original link here