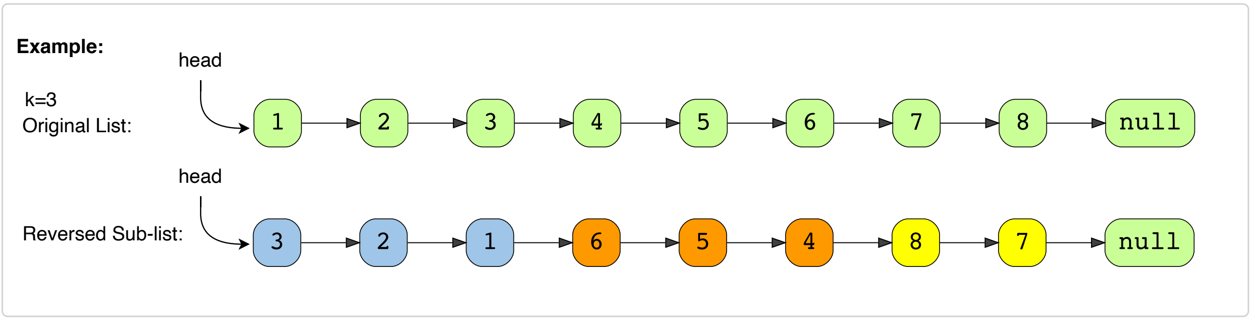
Reverse every K-element Sub-list (medium)

We'll cover the following

Problem Statement

Given the head of a LinkedList and a number 'k', **reverse every 'k' sized sub-list** starting from the head.

If, in the end, you are left with a sub-list with less than 'k' elements, reverse it too.



Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
from __future__ import print_function


class Node:
  def __init__(self, value, next=None):
    self.value = value
    self.next = next

  def print_list(self):
    temp = self
    while temp is not None:
      print(temp.value, end=" ")
      temp = temp.next
    print()


def reverse_every_k_elements(head, k):
```

educative

```
def reverse_every_k_elements(head, k):
  # TODO: Write your code here
  return head


def main():
  head = Node(1)
  head.next = Node(2)
  head.next.next = Node(3)
  head.next.next.next = Node(4)
  head.next.next.next.next = Node(5)
  head.next.next.next.next.next = Node(6)
```

Solution

The problem follows the **In-place Reversal of a LinkedList** pattern and is quite similar to Reverse a Sub-list (https://www.educative.io/collection/page/5668639101419520/5671464854355968/5714632037629952/). The only difference is that we have to reverse all the sub-lists. We can use the same approach, starting with the first sub-list (i.e. `p=1, q=k`) and keep reversing all the sublists of size 'k'.

Code

Most of the code is the same as Reverse a Sub-list (https://www.educative.io/collection/page/5668639101419520/5671464854355968/5714632037629952/); only the highlighted lines have a majority of the changes:

Java | Python3 | C++ | JS

```python
from __future__ import print_function


class Node:
  def __init__(self, value, next=None):
    self.value = value
    self.next = next

  def print_list(self):
    temp = self
    while temp is not None:
      print(temp.value, end=" ")
      temp = temp.next
    print()


def reverse_every_k_elements(head, k):
  if k <= 1 or head is None:
    return head

  current, previous = head, None
  while True:
    last_node_of_previous_part = previous
    # after reversing the LinkedList 'current' will become the last node of the sub-list
    last_node_of_sub_list = current
    next = None  # will be used to temporarily store the next node
    i = 0
    while current is not None and i < k:  # reverse 'k' nodes
```

Time complexity

The time complexity of our algorithm will be $O(N)$ where 'N' is the total number of nodes in the LinkedList.

Space complexity

We only used constant space, therefore, the space complexity of our algorithm is $O(1)$.