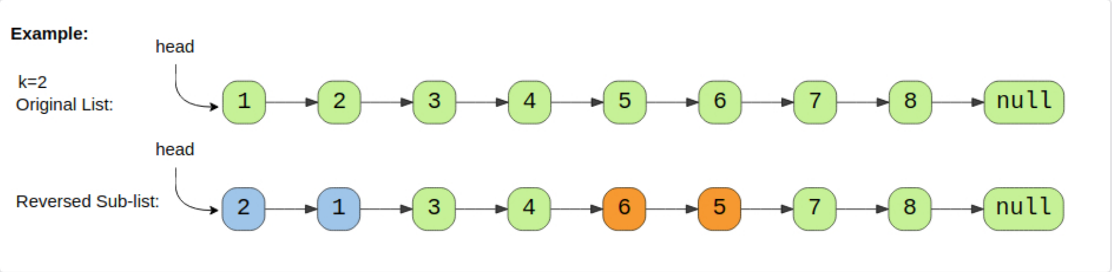
Solution Review: Problem Challenge 1

We'll cover the following

Reverse alternating K-element Sub-list (medium)

Given the head of a LinkedList and a number 'k', **reverse every alternating '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.



Solution

The problem follows the **In-place Reversal of a LinkedList** pattern and is quite similar to Reverse every K-element Sub-list. The only difference is that we have to skip 'k' alternating elements. We can follow a similar approach, and in each iteration after reversing 'k' elements, we will skip the next 'k' elements.

Code

Most of the code is the same as Reverse every K-element Sub-list; only the highlighted lines have a majority of the changes:

Java Python3 C++ JS

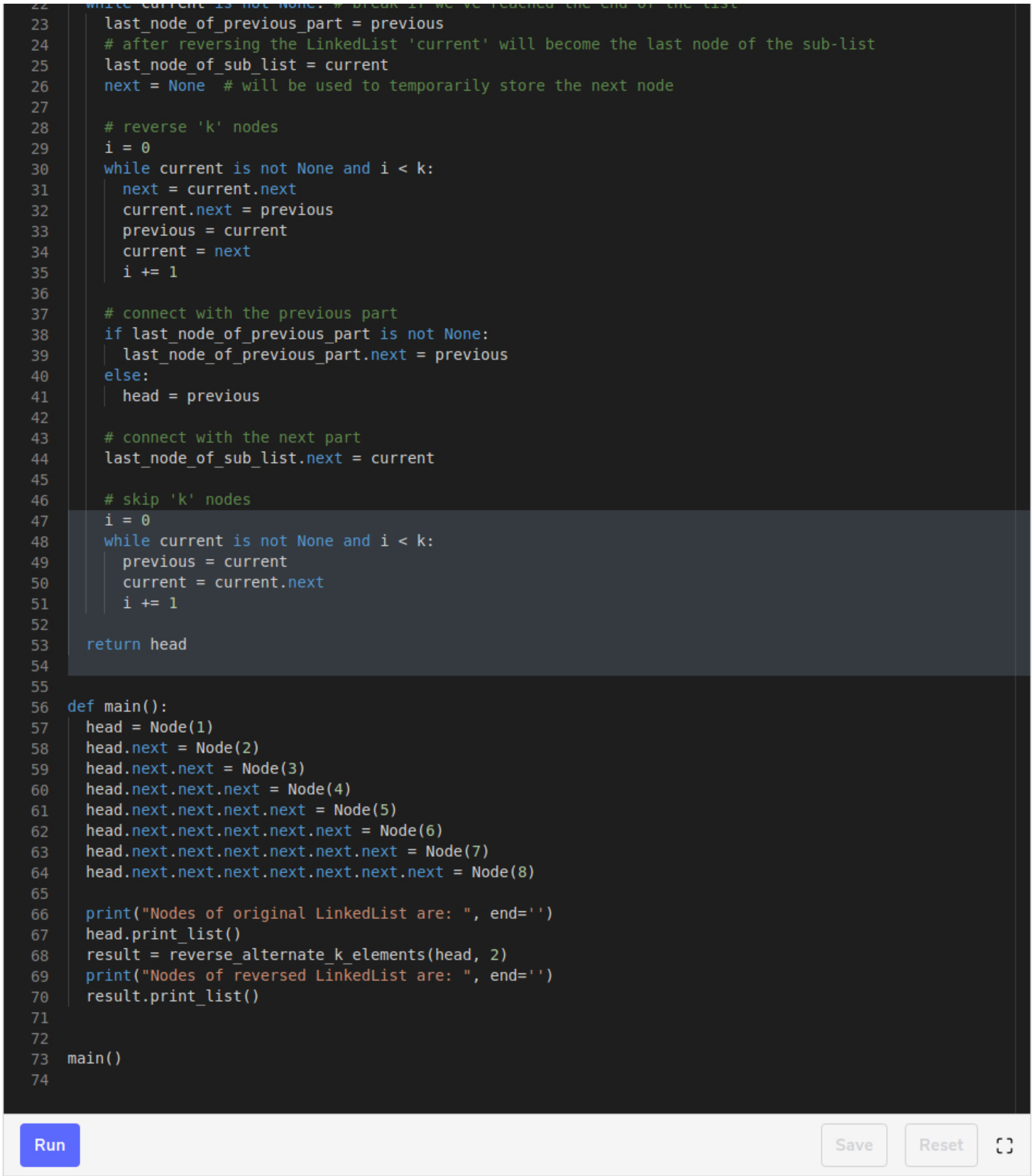
```python
from __future__ import print_function


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

  current, previous = head, None
  while current is not None: # break if we've reached the end of the list
```



```python
  while current is not None:  # break if we've reached the end of the list
    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

    # reverse 'k' nodes
    i = 0
    while current is not None and i < k:
      next = current.next
      current.next = previous
      previous = current
      current = next
      i += 1

    # connect with the previous part
    if last_node_of_previous_part is not None:
      last_node_of_previous_part.next = previous
    else:
      head = previous

    # connect with the next part
    last_node_of_sub_list.next = current

    # skip 'k' nodes
    i = 0
    while current is not None and i < k:
      previous = current
      current = current.next
      i += 1

  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)
  head.next.next.next.next.next.next = Node(7)
  head.next.next.next.next.next.next.next = Node(8)

  print("Nodes of original LinkedList are: ", end='')
  head.print_list()
  result = reverse_alternate_k_elements(head, 2)
  print("Nodes of reversed LinkedList are: ", end='')
  result.print_list()


main()
```

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 $O(1)$.

← Back
Problem Challenge 1

Next →
Problem Challenge 2

✓ Completed

Report an Issue   Ask a Question