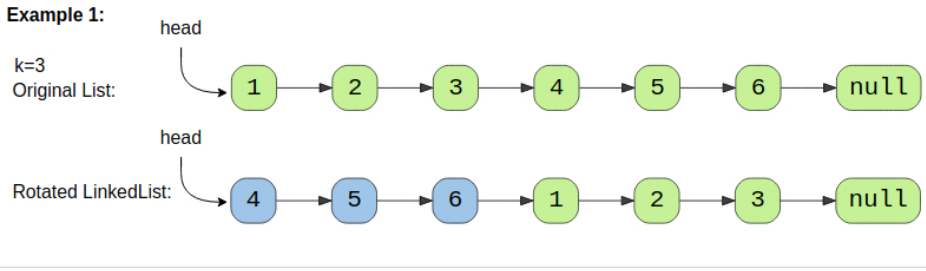
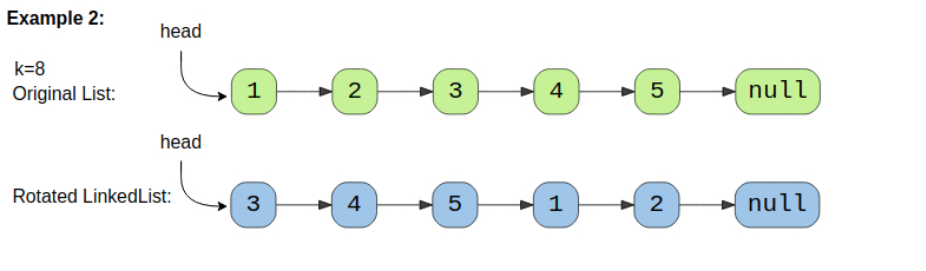
Solution Review: Problem Challenge 2

We'll cover the following

Rotate a LinkedList (medium)

Given the head of a Singly LinkedList and a number 'k', rotate the LinkedList to the right by 'k' nodes.





Solution

Another way of defining the rotation is to take the sub-list of 'k' ending nodes of the LinkedList and connect them to the beginning. Other than that we have to do three more things:

1. Connect the last node of the LinkedList to the head, because the list will have a different tail after the rotation.
2. The new head of the LinkedList will be the node at the beginning of the sublist.
3. The node right before the start of sub-list will be the new tail of the rotated LinkedList.

Code

Here is what our algorithm will look like:

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 rotate(head, rotations):
  if head is None or head.next is None or rotations <= 0:
    return head

  # find the length and the last node of the list
  last_node = head
  list_length = 1
  while last_node.next is not None:
    last_node = last_node.next
    list_length += 1

  last_node.next = head  # connect the last node with the head to make it a circular list
  rotations %= list_length  # no need to do rotations more than the length of the list
  skip_length = list_length - rotations
  last_node_of_rotated_list = head
  for i in range(skip_length - 1):
    last_node_of_rotated_list = last_node_of_rotated_list.next

  # 'last_node_of_rotated_list.next' is pointing to the sub-list of 'k' ending nodes
  head = last_node_of_rotated_list.next
  last_node_of_rotated_list.next = None
  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)

  print("Nodes of original LinkedList are: ", end='')
  head.print_list()
  result = rotate(head, 3)
  print("Nodes of rotated LinkedList are: ", end='')
  result.print_list()


main()

```

Run Save Reset

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 2

Next →

Introduction

Completed

Report an Issue Ask a Question