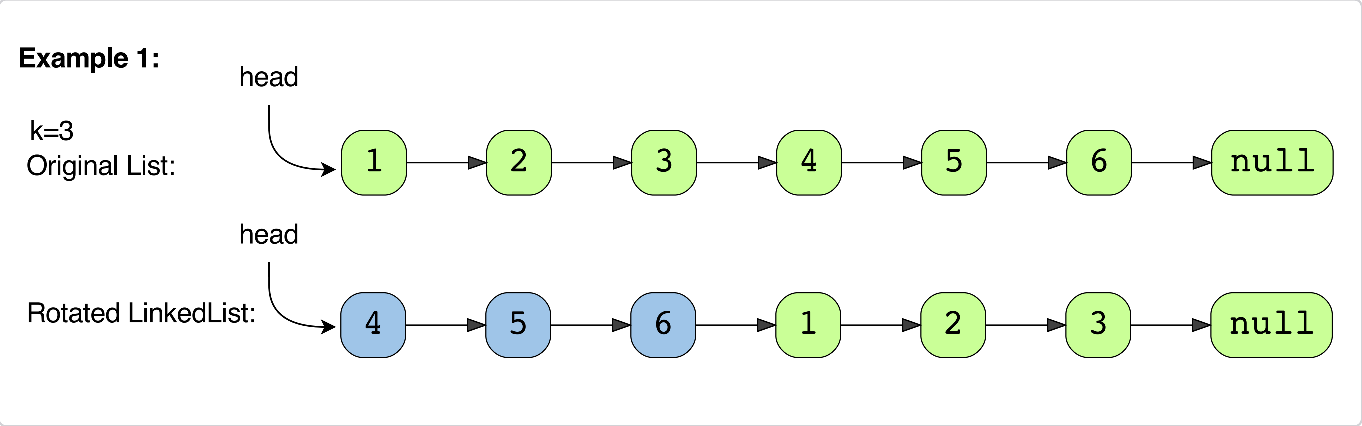
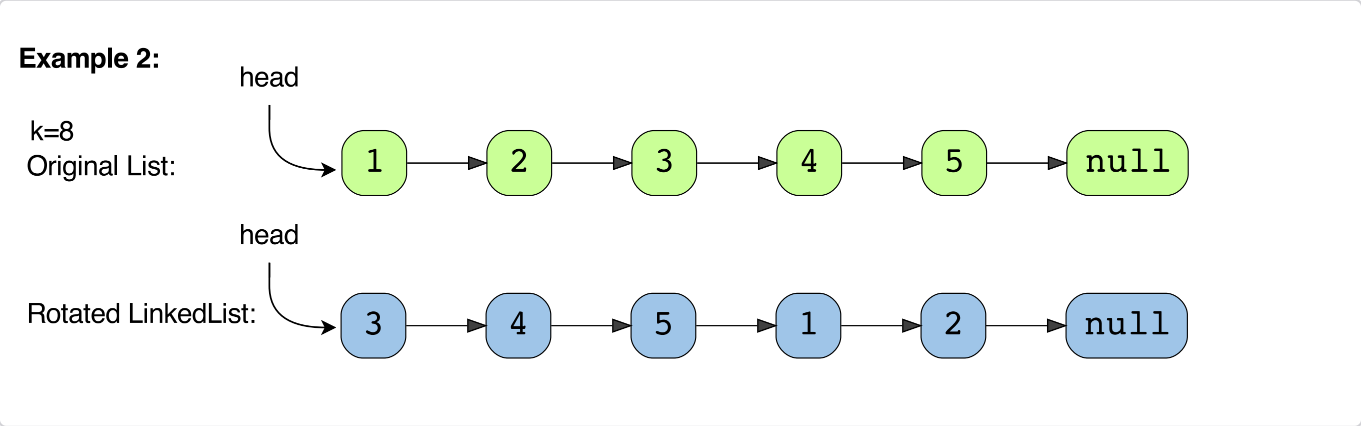
Solution Review: Problem Challenge 2

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

```
import java.util.*;

class ListNode {
  int value = 0;
  ListNode next;

  ListNode(int value) {
    this.value = value;
  }
}

class RotateList {

  public static ListNode rotate(ListNode head, int rotations) {
    if (head == null || head.next == null || rotations <= 0)
      return head;

    // find the length and the last node of the list
    ListNode lastNode = head;
    int listLength = 1;
    while (lastNode.next != null) {
      lastNode = lastNode.next;
      listLength++;
    }

    lastNode.next = head; // connect the last node with the head to make it a ci
    rotations %= listLength; // no need to do rotations more than the length of
    int skipLength = listLength - rotations;
    ListNode lastNodeOfRotatedList = head;
    for (int i = 0; i < skipLength - 1; i++)
      lastNodeOfRotatedList = lastNodeOfRotatedList.next;

    // 'lastNodeOfRotatedList.next' is pointing to the sub-list of 'k' ending no
    head = lastNodeOfRotatedList.next;
    lastNodeOfRotatedList.next = null;
    return head;
  }

  public static void main(String[] args) {
    ListNode head = new ListNode(1);
    head.next = new ListNode(2);
    head.next.next = new ListNode(3);
    head.next.next.next = new ListNode(4);
    head.next.next.next.next = new ListNode(5);
    head.next.next.next.next.next = new ListNode(6);

    ListNode result = RotateList.rotate(head, 3);
    System.out.print("Nodes of the reversed LinkedList are: ");
    while (result != null) {
      System.out.print(result.value + " ");
      result = result.next;
    }
  }
}
```

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

☑ Mark as Completed

Report an Issue