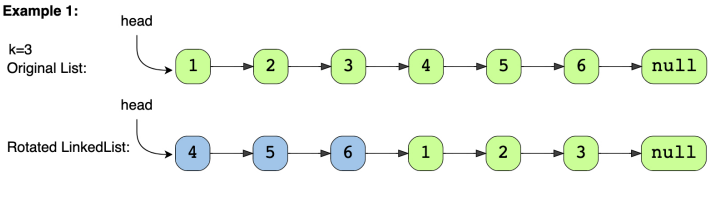
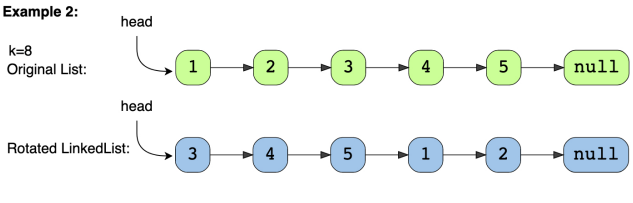
Solution Review: Problem Challenge 2

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

```
class Node {
  constructor(value, next = null) {
    this.value = value;
    this.next = next;
  }

  print_list() {
    let temp = this;
    while (temp !== null) {
      process.stdout.write(`${temp.value} `);
      temp = temp.next;
    }
    console.log();
  }
}

function rotate(head, rotations) {
  if (head === null || head.next === null || rotations <= 0) {
    return head;
  }

  // find the length and the last node of the list
  let last_node = head;
  let list_length = 1;
  while (last_node.next !== null) {
    last_node = last_node.next;
    list_length += 1;
  }
```

RUN | SAVE | RESET

Close

Output 4.846s

```
Nodes of original LinkedList are: 1 2 3 4 5 6
Nodes of reversed LinkedList are: 4 5 6 1 2 3
```

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 $O(1)$.