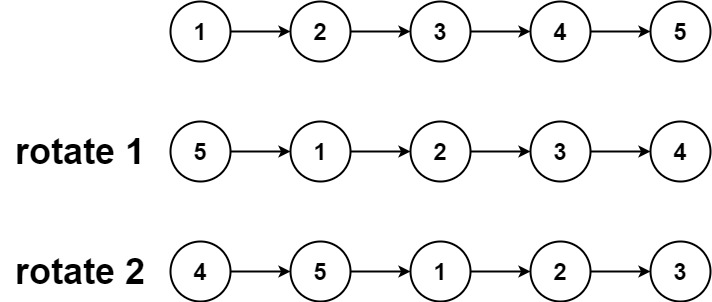
# Question

Given the head of a linked list, rotate the list to the right by k places.

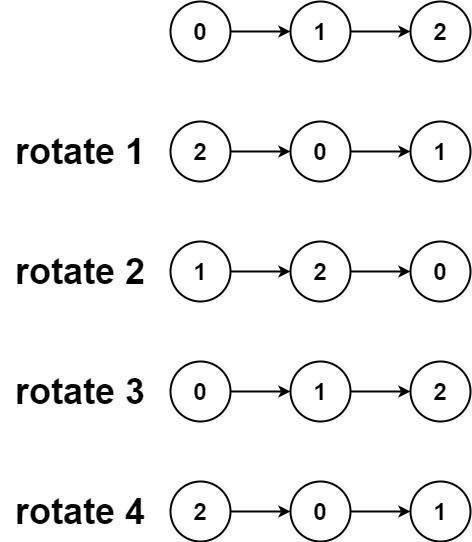
**Example 1:**



**Input:** head = [1,2,3,4,5], k = 2

**Output:** [4,5,1,2,3]

**Example 2:**



**Input:** head = [0,1,2], k = 4

**Output:** [2,0,1]

**Constraints:**

* The number of nodes in the list is in the range [0, 500].
* -100 <= Node.val <= 100
* 0 <= k <= 2 \* 109

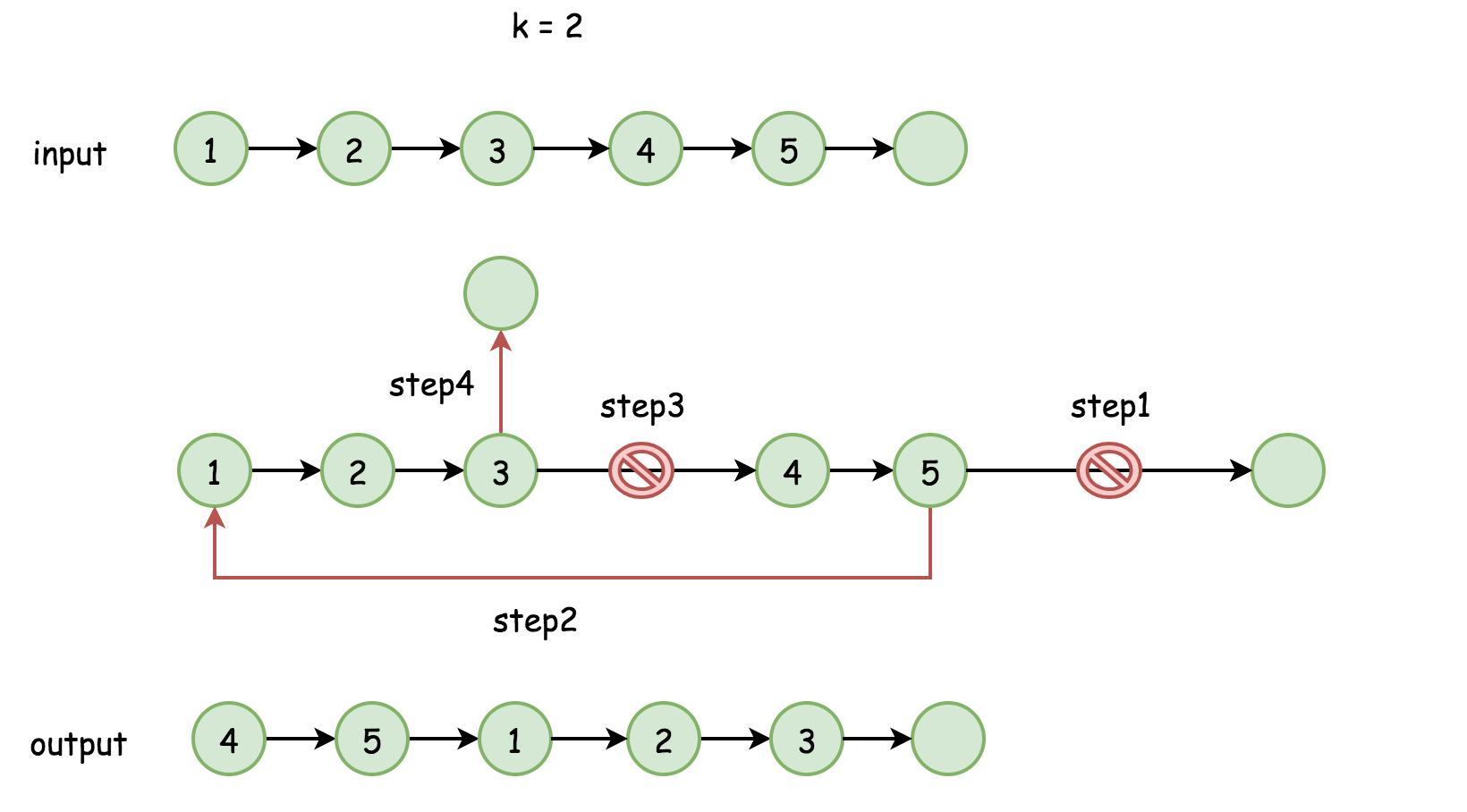
# Solution

#### **Approach 1:**

**Intuition**

The nodes in the list are already linked, and hence the rotation basically means

* To close the linked list into the ring.
* To break the ring after the new tail and just in front of the new head.



Where is the new head?

In the position n - k, where n is a number of nodes in the list. The new tail is just before, in the position n - k - 1.

We were assuming that k < n. What about the case of k >= n?

k could be rewritten as a sum k = (k // n) \* n + k % n, where the first term doesn't result in any rotation. Hence one could simply replace k by k % n to always have number of rotation places smaller than n.

**Algorithm**

The algorithm is quite straightforward :

* Find the old tail and connect it with the head old\_tail.next = head to close the ring. Compute the length of the list n at the same time.
* Find the new tail, which is (n - k % n - 1)th node from the head and the new head, which is (n - k % n)th node.
* Break the ring new\_tail.next = None and return new\_head.

**Implementation**

1 / 14

|  |
| --- |
| **class Solution {**  **public ListNode rotateRight(ListNode head, int k) {**  **// base cases**  **if (head == null) return null;**  **if (head.next == null) return head;**  **// close the linked list into the ring**  **ListNode old\_tail = head;**  **int n;**  **for(n = 1; old\_tail.next != null; n++)**  **old\_tail = old\_tail.next;**  **old\_tail.next = head;**  **// find new tail : (n - k % n - 1)th node**  **// and new head : (n - k % n)th node**  **ListNode new\_tail = head;**  **for (int i = 0; i < n - k % n - 1; i++)**  **new\_tail = new\_tail.next;**  **ListNode new\_head = new\_tail.next;**  **// break the ring**  **new\_tail.next = null;**  **return new\_head;**  **}**  **}** |

**Complexity Analysis**

* Time complexity : \mathcal{O}(N)O(*N*) where N*N* is a number of elements in the list.
* Space complexity : \mathcal{O}(1)O(1) since it's a constant space solution.