Rotate a Linked List

In this article, we will solve the problem: “Rotate a Linked List”

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

Examples:

Example 1:

Input:

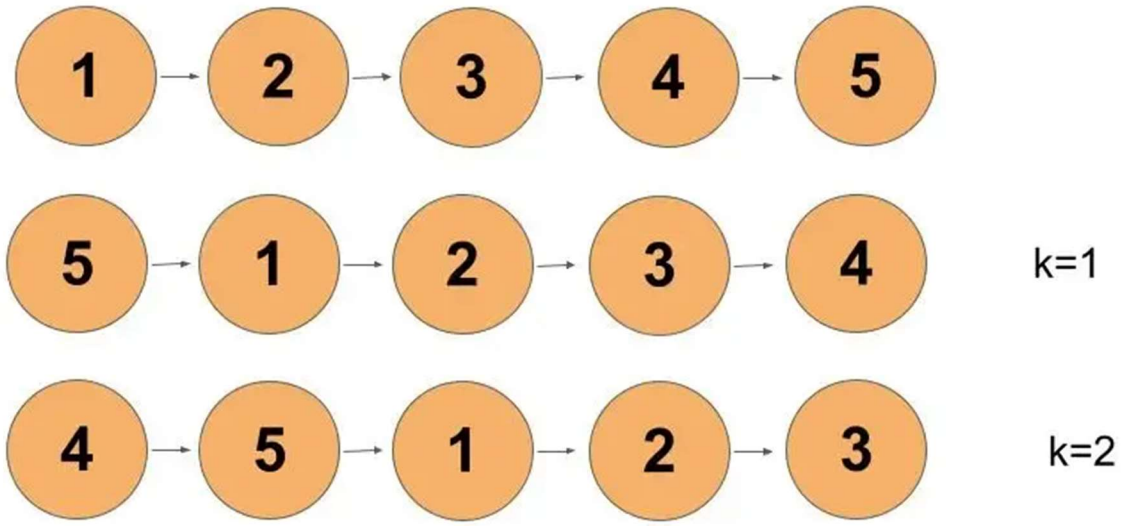
```
head = [1,2,3,4,5]

k = 2
```

Output:

```
head = [4,5,1,2,3]
```

Explanation:

```
We have to rotate the list to the right twice.
```



Example 2:

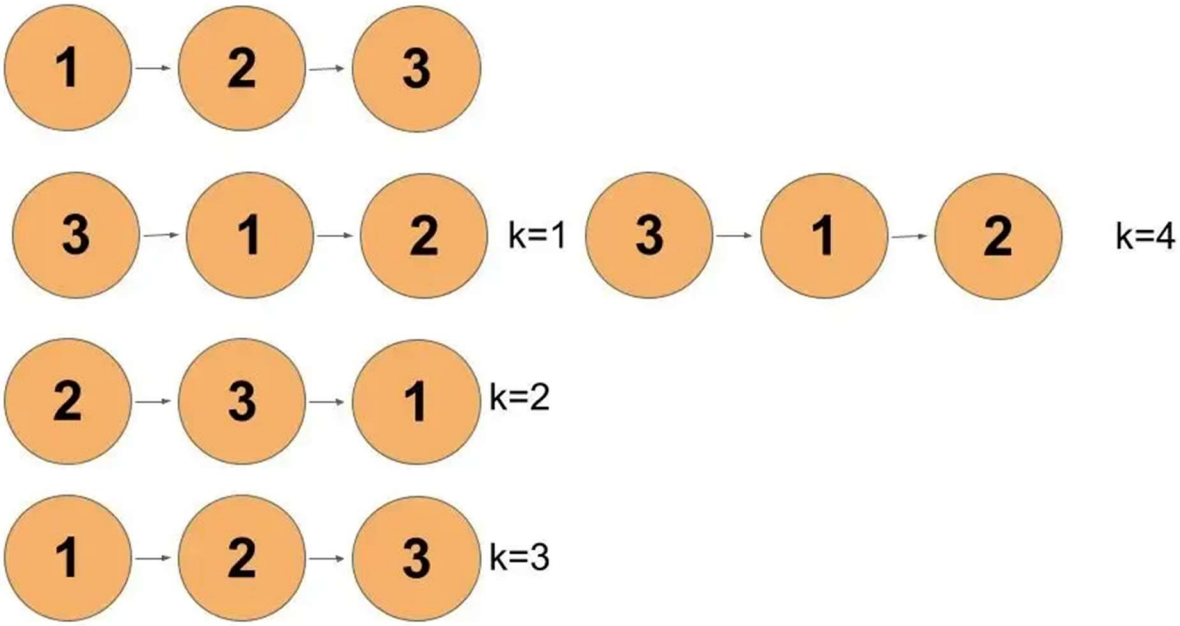
Input:

```
head = [1,2,3]
k = 4
```

Output:

```
head = [3,1,2]
```

Explanation:



Solution: Brute Force

Approach:

We have to move the last element to first for each k.

For each k, find the last element from the list. Move it to the first.

Code:

- C++ Code
- Java Code
- Python Code

```cpp
#include<bits/stdc++.h>
using namespace std;

class node {
    public:
        int num;
        node* next;
        node(int a) {
```

```
            num = a;
            next = NULL;
        }
};
//utility function to insert node at the end of the list
void insertNode(node* &head,int val) {
    node* newNode = new node(val);
    if(head == NULL) {
        head = newNode;
        return;
    }
    node* temp = head;
    while(temp->next != NULL) temp = temp->next;

    temp->next = newNode;
    return;
}
//utility function to rotate list by k times
node* rotateRight(node* head,int k) {
    if(head == NULL||head->next == NULL) return head;
    for(int i=0;i<k;i++) {
        node* temp = head;
        while(temp->next->next != NULL) temp = temp->next;
        node* end = temp->next;
        temp->next = NULL;
        end->next = head;
        head = end;
    }
    return head;
}
//utility function to print list
void printList(node* head) {
    while(head->next != NULL) {
        cout<<head->num<<"->";
        head = head->next;
    }
    cout<<head->num<<endl;
    return;
}

int main() {
    node* head = NULL;
    //inserting Node
    insertNode(head,1);
    insertNode(head,2);
    insertNode(head,3);
    insertNode(head,4);
```

```
    insertNode(head,5);

    cout<<"Original list: ";
    printList(head);

    int k = 2;
    node* newHead = rotateRight(head,k);//calling function for rotating right
of
    the nodes by k times

    cout<<"After "<<k<<" iterations: ";
    printList(newHead);//list after rotating nodes
    return 0;
}
```

Output:

Original list: 1->2->3->4->5
After 2 iterations: 4->5->1->2->3

Time Complexity: O(Number of nodes present in the list*k)

Reason: For k times, we are iterating through the entire list to get the last element and move it to first.

Space Complexity: O(1)

Reason: No extra data structures is used for computations

Solution: Optimal Solution

Approach:

Let's take an example.

head = [1,2,3,4,5] k = 2000000000

If we see a brute force approach, it will take O(5*2000000000) which is not a good time complexity when we can optimize it.

We can see that for every k which is multiple of the length of the list, we get back the original list. Try to operate brute force on any linked list for k as a multiple of the length of the list.

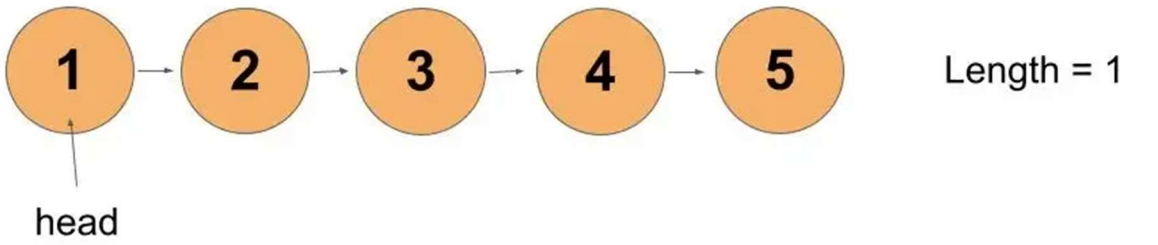
This gives us a hint that for k greater than the length of the list, we have to rotate the list for k%length of the list. This reduces our time complexity.

Steps to the algorithm:-

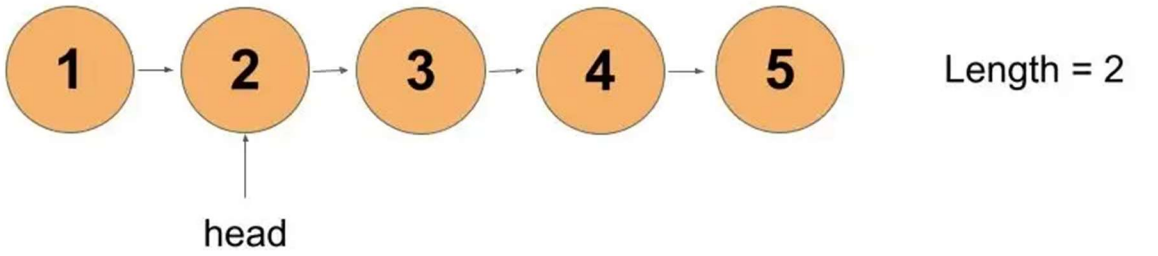
- Calculate the length of the list.
- Connect the last node to the first node, converting it to a **circular linked list**.
- Iterate to cut the link of the last node and start a node of k%length of the list rotated list.

Dry Run:

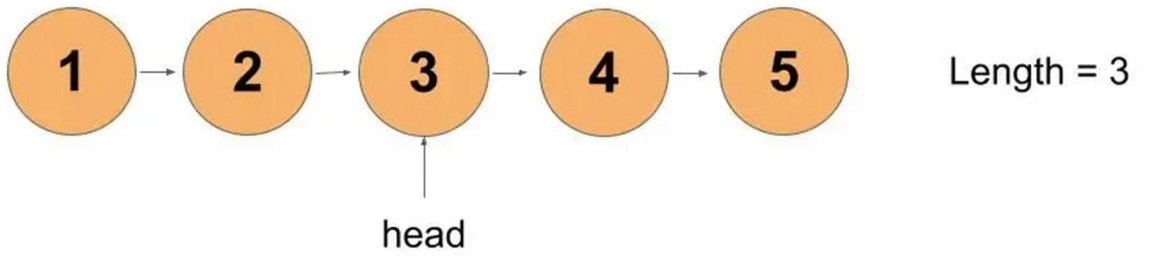
Let's calculate the length of the list by iterating on it until it reaches null and increasing the count. Once the length is calculated we will connect the last node to the first node.



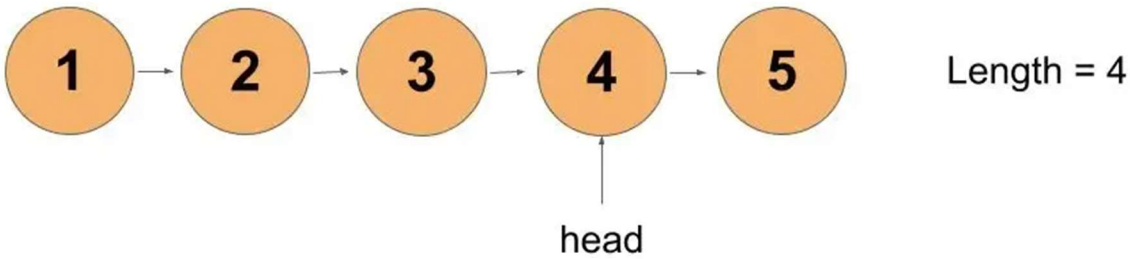
Length = 1



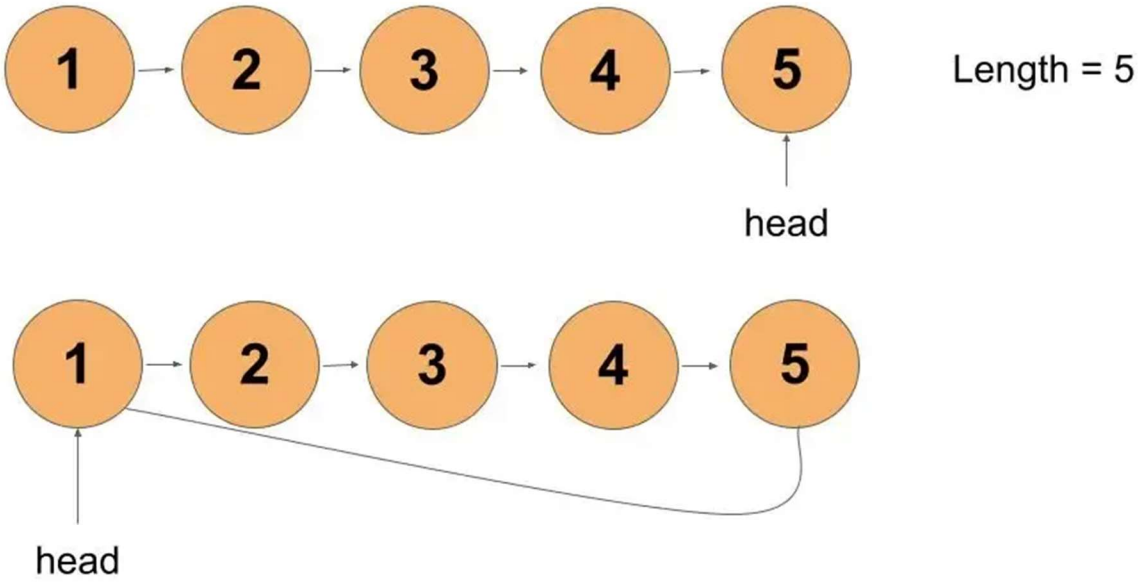
Length = 2



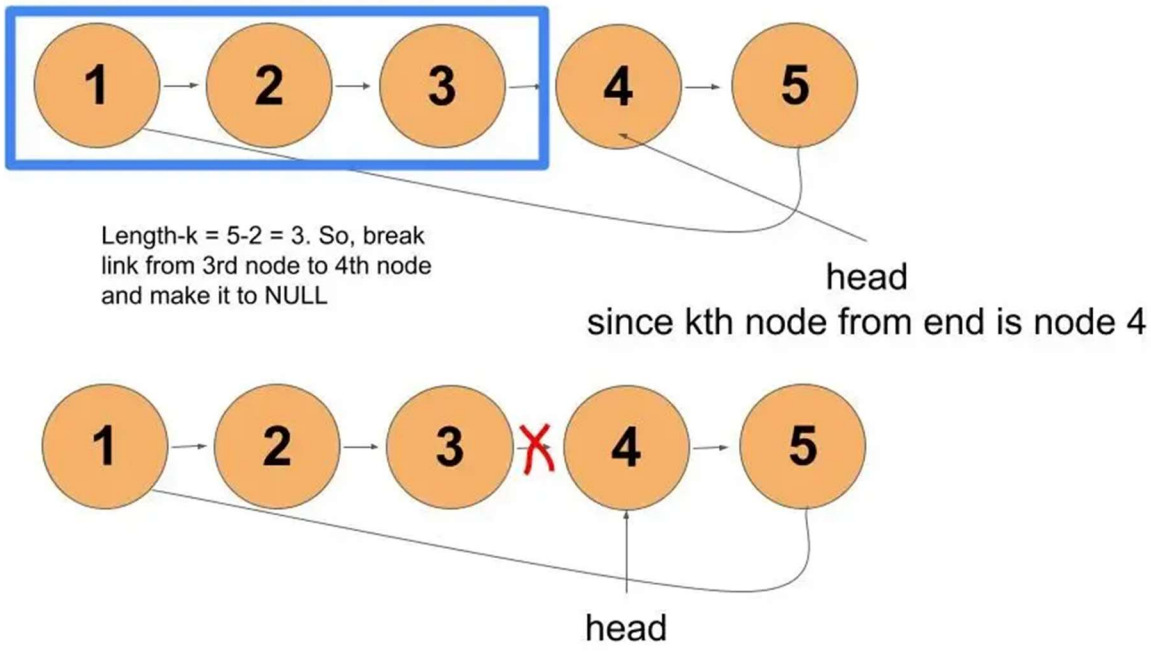
Length = 3



Length = 4



Now, the length of the list is 5 and k is 2. k is less than the length of the given list. So, we will have the head of the rotating list at the kth element from the end remove the link from the length-k node from its next node and make it NULL.



Thus, we received our desired output.

Code:

- C++ Code
- Java Code
- Python Code

```cpp
#include<bits/stdc++.h>
using namespace std;


class node {
    public:
        int num;
        node* next;
        node(int a) {
            num = a;
            next = NULL;
        }
};
//utility function to insert node at the end of the list
void insertNode(node* &head,int val) {
    node* newNode = new node(val);
    if(head == NULL) {
        head = newNode;
        return;
    }
    node* temp = head;
    while(temp->next != NULL) temp = temp->next;

    temp->next = newNode;
    return;
}
//utility function to rotate list by k times
node* rotateRight(node* head,int k) {
    if(head == NULL||head->next == NULL||k == 0) return head;
    //calculating length
    node* temp = head;
    int length = 1;
    while(temp->next != NULL) {
        ++length;
        temp = temp->next;
    }
    //link last node to first node
    temp->next = head;
```

```
    k = k%length; //when k is more than length of list
    int end = length-k; //to get end of the list
    while(end--) temp = temp->next;
    //breaking last node link and pointing to NULL
    head = temp->next;
    temp->next = NULL;

    return head;
}

//utility function to print list
void printList(node* head) {
    while(head->next != NULL) {
        cout<<head->num<<"->";
        head = head->next;
    }
    cout<<head->num<<endl;
    return;
}

int main() {
    node* head = NULL;
    //inserting Node
    insertNode(head,1);
    insertNode(head,2);
    insertNode(head,3);
    insertNode(head,4);
    insertNode(head,5);

    cout<<"Original list: ";
    printList(head);

    int k = 2;
    node* newHead = rotateRight(head,k);//calling function for rotating right
of the nodes by k times

    cout<<"After "<<k<<" iterations: ";
    printList(newHead);//list after rotating nodes
    return 0;
}
```

Output:

Original list: 1->2->3->4->5
After 2 iterations: 4->5->1->2->3

Time Complexity: O(length of list) + O(length of list – (length of list%k))

Reason: O(length of the list) for calculating the length of the list. O(length of the list – (length of list%k)) for breaking link.

Space Complexity: O(1)

Reason: No extra data structure is used for computation.