# Rotate a Linked List

Given a singly linked list, rotate the linked list counter-clockwise by k nodes. Where k is a given positive integer. For example, if the given linked list is 10->20->30->40->50->60 and k is 4, the list should be modified to 50->60->10->20->30->40. Assume that k is smaller than the count of nodes in linked list.

To rotate the linked list, we need to change next of kth node to NULL, next of last node to previous head node, and finally change head to (k+1)th node. So we need to get hold of three nodes: kth node, (k+1)th node and last node.
Traverse the list from beginning and stop at kth node. Store pointer to kth node. We can get (k+1)th node using kthNode->next. Keep traversing till end and store pointer to last node also. Finally, change pointers as stated above.

## C/C++

```
// C/C++ program to rotate a linked list counter clock wise

#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct node
{
    int data;
    struct node* next;
};

// This function rotates a linked list counter-clockwise and
// updates the head. The function assumes that k is smaller
// than size of linked list. It doesn't modify the list if
// k is greater than or equal to size
void rotate(struct node **head_ref, int k)
{
     if (k == 0)
       return;

    // Let us understand the below code for example k = 4 and
    // list = 10->20->30->40->50->60.
    struct node* current = *head_ref;

    // current will either point to kth or NULL after this loop.
    //  current will point to node 40 in the above example
    int count = 1;
```

```
    while (count < k && current != NULL)
    {
        current = current->next;
        count++;
    }

    // If current is NULL, k is greater than or equal to count
    // of nodes in linked list. Don't change the list in this case
    if (current == NULL)
        return;

    // current points to kth node. Store it in a variable.
    // kthNode points to node 40 in the above example
    struct node *kthNode = current;

    // current will point to last node after this loop
    // current will point to node 60 in the above example
    while (current->next != NULL)
        current = current->next;

    // Change next of last node to previous head
    // Next of 60 is now changed to node 10
    current->next = *head_ref;

    // Change head to (k+1)th node
    // head is now changed to node 50
    *head_ref = kthNode->next;

    // change next of kth node to NULL
    // next of 40 is now NULL
    kthNode->next = NULL;
}

/* UTILITY FUNCTIONS */
/* Function to push a node */
void push (struct node** head_ref, int new_data)
{
    /* allocate node */
    struct node* new_node =
        (struct node*) malloc(sizeof(struct node));

    /* put in the data  */
    new_node->data  = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref)    = new_node;
}

/* Function to print linked list */
void printList(struct node *node)
{
    while (node != NULL)
    {
        printf("%d ", node->data);
        node = node->next;
    }
}

/* Drier program to test above function*/
int main(void)
{
    /* Start with the empty list */
```

```
    struct node* head = NULL;

    // create a list 10->20->30->40->50->60
    for (int i = 60; i > 0; i -= 10)
        push(&head, i);

    printf("Given linked list \n");
    printList(head);
    rotate(&head, 4);

    printf("\nRotated Linked list \n");
    printList(head);

    return (0);
}
```

Run on IDE

## Java

```
// Java program to rotate a linked list

class LinkedList
{
    Node head;  // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d)
        {
            data = d;
            next = null;
        }
    }

    // This function rotates a linked list counter-clockwise
    // and updates the head. The function assumes that k is
    // smaller than size of linked list. It doesn't modify
    // the list if k is greater than or equal to size
    void rotate(int k)
    {
        if (k == 0) return;

        // Let us understand the below code for example k = 4
        // and list = 10->20->30->40->50->60.
        Node current  = head;

        // current will either point to kth or NULL after this
        // loop. current will point to node 40 in the above example
        int count = 1;
        while (count < k && current !=  null)
        {
            current = current.next;
            count++;
        }

        // If current is NULL, k is greater than or equal to count
        // of nodes in linked list. Don't change the list in this case
        if (current == null)
```

```
            return;

        // current points to kth node. Store it in a variable.
        // kthNode points to node 40 in the above example
        Node kthNode = current;

        // current will point to last node after this loop
        // current will point to node 60 in the above example
        while (current.next != null)
            current = current.next;

        // Change next of last node to previous head
        // Next of 60 is now changed to node 10

        current.next = head;

        // Change head to (k+1)th node
        // head is now changed to node 50
        head = kthNode.next;

        // change next of kth node to null
        kthNode.next = null;

    }

    /*  Given a reference (pointer to pointer) to the head
        of a list and an int, push a new node on the front
        of the list. */
    void push(int new_data)
    {
        /* 1 & 2: Allocate the Node &
                  Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
    }

    void printList()
    {
        Node temp = head;
        while(temp != null)
        {
            System.out.print(temp.data+" ");
            temp = temp.next;
        }
        System.out.println();
    }

    /* Drier program to test above functions */
    public static void main(String args[])
    {
        LinkedList llist = new LinkedList();

        // create a list 10->20->30->40->50->60
        for (int i = 60; i >= 10; i -= 10)
            llist.push(i);

        System.out.println("Given list");
        llist.printList();

        llist.rotate(4);
```

```
        System.out.println("Rotated Linked List");
        llist.printList();
    }
} /* This code is contributed by Rajat Mishra */
```

Run on IDE

## Python

```
# Python program to rotate a linked list

# Node class
class Node:

    # Constructor to initialize the node object
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:

    # Function to initialize head
    def __init__(self):
        self.head = None

    # Function to insert a new node at the beginning
    def push(self, new_data):
        # allocate node and put the data
        new_node = Node(new_data)

        # Make next of new node as head
        new_node.next = self.head

        # move the head to point to the new Node
        self.head = new_node

    # Utility function to prit the linked LinkedList
    def printList(self):
        temp = self.head
        while(temp):
            print temp.data,
            temp = temp.next

    # This function rotates a linked list counter-clockwise and
    # updates the head. The function assumes that k is smaller
    # than size of linked list. It doesn't modify the list if
    # k is greater than of equal to size
    def rotate(self, k):
        if k == 0:
            return

        # Let us understand the below code for example k = 4
        # and list = 10->20->30->40->50->60
        current = self.head

        # current will either point to kth or NULL after
        # this loop
        # current will point to node 40 in the above example
        count = 1
        while(count <k and current is not None):
            current = current.next
```

```
            count += 1

        # If current is None, k is greater than or equal
        # to count of nodes in linked list. Don't change
        # the list in this case
        if current is None:
            return

        # current points to kth node. Store it in a variable
        # kth node points to node 40 in the above example
        kthNode = current

        # current will point to lsat node after this loop
        # current will point to node 60 in above example
        while(current.next is not None):
            current = current.next

        # Change next of last node to previous head
        # Next of 60 is now changed to node 10
        current.next = self.head

        # Change head to (k+1)th node
        # head is not changed to node 50
        self.head = kthNode.next

        # change next of kth node to NULL
        # next of 40 is not NULL
        kthNode.next = None



# Driver program to test above function
llist = LinkedList()

# Create a list 10->20->30->40->50->60
for i in range(60, 0, -10):
    llist.push(i)

print "Given linked list"
llist.printList()
llist.rotate(4)

print "\nRotated Linked list"
llist.printList()

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Run on IDE

Output:

```
Given linked list
10  20  30  40  50  60
Rotated Linked list
50  60  10  20  30  40
```

Time Complexity: O(n) where n is the number of nodes in Linked List. The code traverses the linked list only once.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



83 Comments Category: Linked Lists

## Related Posts:

- Merge two sorted linked lists such that merged list is in reverse order
- Compare two strings represented as linked lists
- Rearrange a given linked list in-place.
- Sort a linked list that is sorted alternating ascending and descending orders?
- Select a Random Node from a Singly Linked List
- Merge Sort for Doubly Linked List
- Point to next higher value node in a linked list with an arbitrary pointer
- Swap nodes in a linked list without swapping data



Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.