Reverse a linked list in groups of size k

Given a linked list, write a function to reverse every k nodes (where k is an input to the function).

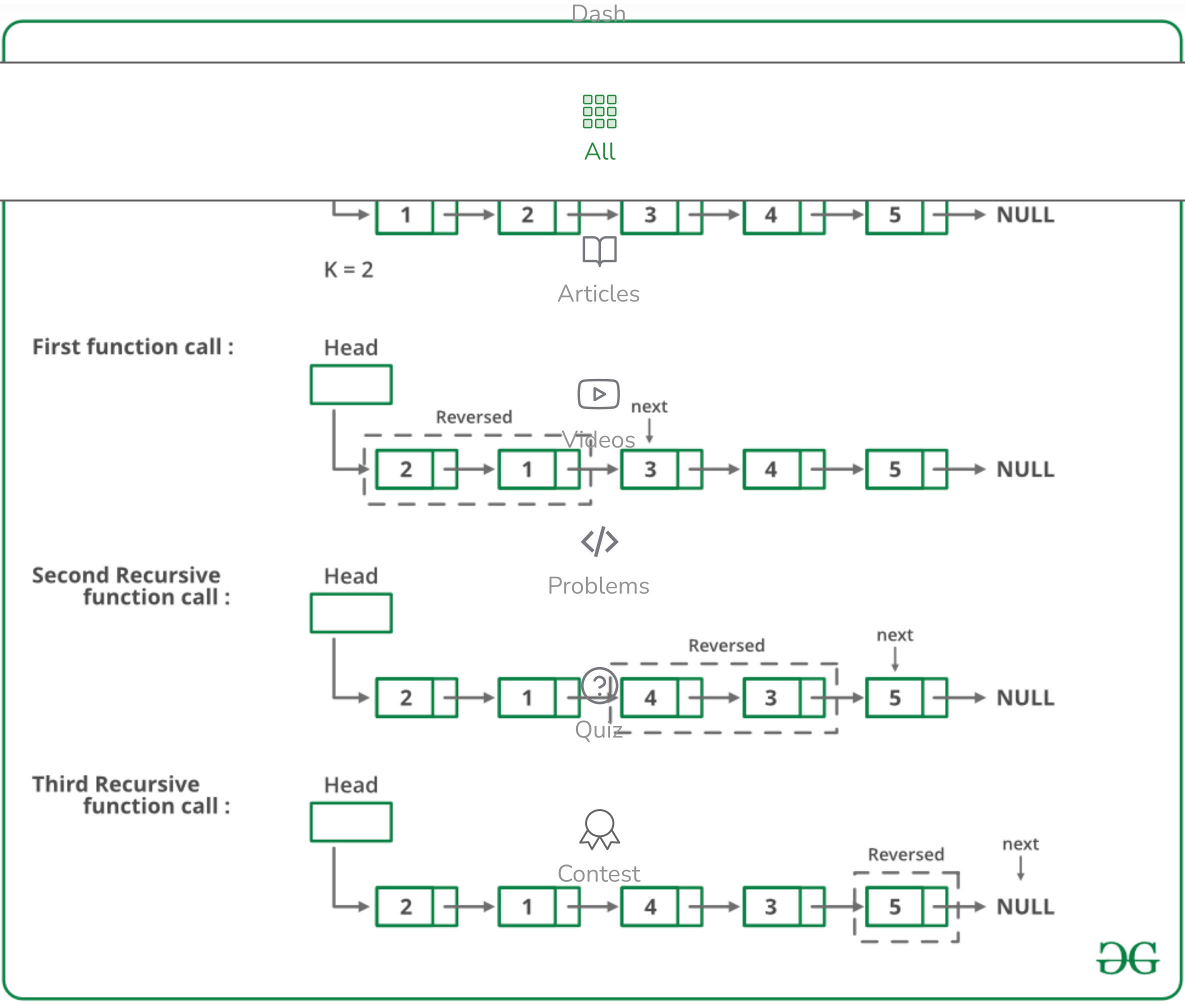
Example:

> ***Input***: *1->2->3->4->5->6->7->8->NULL, K = 3*
> ***Output***: *3->2->1->6->5->4->8->7->NULL*
> ***Input***: *1->2->3->4->5->6->7->8->NULL, K = 5*
> ***Output***: *5->4->3->2->1->8->7->6->NULL*

- Reverse the first sub-list of size k. While reversing keep track of the next node and previous node. Let the pointer to the next node be *next* and pointer to the previous node be *prev*. See this post for reversing a linked list.
- *head->next = reverse(next, k)* (Recursively call for rest of the list and link the two sub-lists)
- Return *prev* (*prev* becomes the new head of the list (see the diagrams of an iterative method of this post)

Below is image shows how the reverse function works:

Dash
All
Articles
Videos
Problems
Quiz
Contest



Below is the implementation of the above approach:

C++ Java

```
 // Java program to reverse a linked list in groups of
// given size
class LinkedList {
    Node head; // head of list

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

Menu

Track Progress
97 of 132 Complete. (74%)

Dash
All
Articles
Videos
Problems
Quiz
Contest
Menu

```
    {
        Node next = null;
        Node prev = null;

        int count = 0;

        /* Reverse first k nodes of linked list */
        while (count < k && current != null) {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
            count++;
        }

        /* next is now a pointer to (k+1)th node
        Recursively call for the list starting from
        current. And make rest of the list as next of
        first node */
        if (next != null)
            head.next = reverse(next, k);

        // prev is now head of input list
        return prev;
    }

    /* Utility functions */

    /* Inserts a new Node at front of the list. */
    public 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;
```

Track Progress
97 of **132** Complete. (74%)

Dash
All
Articles
Videos
Problems
Quiz
Contest
Menu

```
    }

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

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

        /* Constructed Linked List is 1->2->3->4->5->6->
        7->8->8->9->null */
        llist.push(9);
        llist.push(8);
        llist.push(7);
        llist.push(6);
        llist.push(5);
        llist.push(4);
        llist.push(3);
        llist.push(2);
        llist.push(1);

        System.out.println("Given Linked List");
        llist.printList();

        llist.head = llist.reverse(llist.head, 3);

        System.out.println("Reversed list");
        llist.printList();
    }
}
```

Track Progress

97 of **132** Complete. (74%)

Dash
All
Articles
Videos
Problems
Quiz
Contest

```
Given linked list
```

Complexity Analysis:

- **Time Complexity:** O(n).
  Traversal of list is done only once and it has 'n' elements.
- **Auxiliary Space:** O(n/k).
  For each Linked List of size n, n/k or (n/k)+1 calls will be made during the recursion.

We can solve this question in O(1) Space Complexity.

Approach – 2 Space Optimized – Iterative

The following steps are required for this Algorithm:

1. Create a dummy node and point it to the head of input i.e dummy->next = head.
2. Calculate the length of the linked list which takes O(N) time, where N is the length of the linked list.
3. Initialize three-pointers prev, curr, next to reverse k elements for every group.
4. Iterate over the linked lists till next!=NULL.
5. Points curr to the prev->next and next to the curr next.
6. Then, Using the inner for loop reverse the particular group using these four steps:

> - *curr->next = next->next*
> - *next->next = prev->next*
> - *prev->next = next*
> - *next = curr->next*

7. This for loop runs for k-1 times for all groups except the last remaining element, for the last remaining element it runs for the remaining length of the linked list – 1.

8. Decrement count after for loop by k count -= k, to determine the length of the remaining linked list.

Menu

9. Change prev position to curr, prev = curr.

Track Progress
97 of 132 Complete. (74%)

```
// Linked List Node
class Node {
    int data;
    Node next;
    Node(int a)
    {
        data = a;
        next = null;
    }
}

class GFG {
    // utility function to insert node in the list
    static Node push(Node head, int val)
    {
        Node newNode = new Node(val);
        if (head == null) {
            head = newNode;
            return head;
        }

        Node temp = head;
        while (temp.next != null)
            temp = temp.next;

        temp.next = newNode;
        return head;
    }

    // utility function to reverse k nodes in the list
    static Node reverse(Node head, int k)
    {
        // If head is NULL or K is 1 then return head
        if (head == null || head.next == null)
            return head;
        dummy.next = head;
        Node curr = dummy;
        Node next = dummy;

        // Calculating the length of linked list
        int count = 0;
        while (curr != null) {
            count++;
            curr = curr.next;
        }

        // Iterating till next is not NULL
        while (next != null) {
            curr = prev.next; // Curr position after every
                            // reverse group
            next = curr.next; // Next will always next to
                            // curr
            int toLoop
                = count > k
                    ? k
                    : count - 1; // toLoop will set to
                                // count - 1 in case of
                                // remaining element

            for (int i = 1; i < toLoop; i++) {
                // 4 steps as discussed above
                curr.next = next.next;
                next.next = prev.next;
                prev.next = next;
                next = curr.next;
            }
            prev = curr; // Setting prev to curr
            count -= k; // Update count
        }
        return dummy.next; // dummy -> next will be our new
                        // head for output linked
        // list
    }
```

Dash
All
Articles
Videos
Problems
Quiz
Contest
Menu

Track Progress

97 of **132** Complete. (74%)

```
    {
        }
        System.out.println(head.data);

    public static void main(String args[])
    {
        Node head = null;
        int k = 3;
        head = push(head, 1);
        head = push(head, 2);
        head = push(head, 3);
        head = push(head, 4);
        head = push(head, 5);
        head = push(head, 6);
        head = push(head, 7);
        head = push(head, 8);
        head = push(head, 9);

        System.out.println("Given Linked List");
        print(head);
        System.out.println("Reversed list");
        Node newHead = reverse(head, k);
        print(newHead);
    }
}
```

Output

```
Given linked list
1 2 3 4 5 6 7 8 9
Reversed Linked list
3 2 1 6 5 4 9 8 7
```

Complexity Analysis

Track Progress

97 of 132 Complete. (74%)

Space Complexity: O(1) : No extra space is used.

Dash

All

Articles

Videos

Problems

Quiz

Contest

Report An Issue

If you are facing any issue on this page. Please let us know.

Menu

Track Progress

97 of **132** Complete. (74%)