Reverse alternate K nodes in a Singly Linked List

Given a linked list, write a function to reverse every alternate k nodes (where k is an input to the function) in an efficient way. Give the complexity of your algorithm.

```
Example:
Inputs:   1->2->3->4->5->6->7->8->9->NULL and k = 3
Output:   3->2->1->4->5->6->9->8->7->NULL.
```

Method 1 (Process 2k nodes and recursively call for rest of the list)
This method is basically an extension of the method discussed in this post.

```
kAltReverse(struct node *head, int k)
  1)  Reverse first k nodes.
  2)  In the modified list head points to the kth node.  So change next
       of head to (k+1)th node
  3)  Move the current pointer to skip next k nodes.
  4)  Call the kAltReverse() recursively for rest of the n - 2k nodes.
  5)  Return new head of the list.
```

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

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

/* Reverses alternate k nodes and
```


Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion

```
   returns the pointer to the new head node */
struct node *kAltReverse(struct node *head, int k)
{
    struct node* current = head;
    struct node* next;
    struct node* prev = NULL;
    int count = 0;

    /*1) reverse first k nodes of the linked list */
    while (current != NULL && count < k)
    {
       next  = current->next;
       current->next = prev;
       prev = current;
       current = next;
       count++;
    }

    /* 2) Now head points to the kth node.  So change next
       of head to (k+1)th node*/
    if(head != NULL)
      head->next = current;

    /* 3) We do not want to reverse next k nodes. So move the current
        pointer to skip next k nodes */
    count = 0;
    while(count < k-1 && current != NULL )
    {
      current = current->next;
      count++;
    }

    /* 4) Recursively call for the list starting from current->next.
       And make rest of the list as next of first node */
    if(current !=  NULL)
       current->next = kAltReverse(current->next, k);

    /* 5) prev is new head of the input list */
    return prev;
}

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

Geometric Algorithms

Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

```
            (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)
{
    int count = 0;
    while(node != NULL)
    {
        printf("%d  ", node->data);
        node = node->next;
        count++;
    }
}

/* Drier program to test above function*/
int main(void)
{
    /* Start with the empty list */
    struct node* head = NULL;

    // create a list 1->2->3->4->5...... ->20
    for(int i = 20; i > 0; i--)
      push(&head, i);

     printf("\n Given linked list \n");
     printList(head);
     head = kAltReverse(head, 3);

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

     getchar();
     return(0);
}
```

Output:

Given linked list
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Modified Linked list
3 2 1 4 5 6 9 8 7 10 11 12 15 14 13 16 17 18 20 19

Time Complexity: O(n)

Method 2 (Process k nodes and recursively call for rest of the list)
The method 1 reverses the first k node and then moves the pointer to k nodes ahead. So method 1 uses two while loops and processes 2k nodes in one recursive call.
This method processes only k nodes in a recursive call. It uses a third bool parameter b which decides whether to reverse the k elements or simply move the pointer.

```
_kAltReverse(struct node *head, int k, bool b)
  1)  If b is true, then reverse first k nodes.
  2)  If b is false, then move the pointer k nodes ahead.
  3)  Call the kAltReverse() recursively for rest of the n - k nodes and link
       rest of the modified list with end of first k nodes.
  4)  Return new head of the list.
```

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

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

/* Helper function for kAltReverse() */
struct node * _kAltReverse(struct node *node, int k, bool b);
```

```
/* Alternatively reverses the given linked list in groups of
   given size k. */
struct node *kAltReverse(struct node *head, int k)
{
  return _kAltReverse(head, k, true);
}
```


Recent Comments

Abhi You live US or India?
Google (Mountain View) interview · 52 minutes ago

Aman Hi, Why arent we checking for conditions...
Write a C program to Delete a Tree. · 1 hour ago

kzs please provide solution for the problem...
Backtracking | Set 2 (Rat in a Maze) · 1 hour ago

Sanjay Agarwal bool tree::Root_to_leaf_path_given_sum(tree...
Root to leaf path sum equal to a given number · 2 hours ago

GOPI GOPINATH @admin Highlight this sentence "We can easily...
Count trailing zeroes in factorial of a number · 2 hours ago

newCoder3006 If the array contains negative numbers also. We...
Find subarray with given sum · 2 hours ago

```
/*  Helper function for kAltReverse().  It reverses k nodes of the lis
    the third parameter b is passed as true, otherwise moves the point
    nodes ahead and recursively calls iteself  */
struct node * _kAltReverse(struct node *node, int k, bool b)
{
   if(node == NULL)
       return NULL;

   int count = 1;
   struct node *prev = NULL;
   struct node  *current = node;
   struct node *next;

   /* The loop serves two purposes
      1) If b is true, then it reverses the k nodes
      2) If b is false, then it moves the current pointer */
   while(current != NULL && count <= k)
   {
       next = current->next;

       /* Reverse the nodes only if b is true*/
       if(b == true)
          current->next = prev;

       prev = current;
       current = next;
       count++;
   }

   /* 3) If b is true, then node is the kth node.
       So attach rest of the list after node.
     4) After attaching, return the new head */
   if(b == true)
   {
        node->next = _kAltReverse(current,k,!b);
        return prev;
   }

   /* If b is not true, then attach rest of the list after prev.
     So attach rest of the list after prev */
   else
   {
        prev->next = _kAltReverse(current, k, !b);
        return node;
   }
}
```

```c
/* 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)
{
    int count = 0;
    while(node != NULL)
    {
        printf("%d  ", node->data);
        node = node->next;
        count++;
    }
}

/* Drier program to test above function*/
int main(void)
{
    /* Start with the empty list */
    struct node* head = NULL;
    int i;

    // create a list 1->2->3->4->5...... ->20
    for(i = 20; i > 0; i--)
      push(&head, i);

    printf("\n Given linked list \n");
    printList(head);
    head = kAltReverse(head, 3);

    printf("\n Modified Linked list \n");
```

```
    printList(head);

    getchar();
    return(0);
}
```

Output:
Given linked list
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Modified Linked list
3 2 1 4 5 6 9 8 7 10 11 12 15 14 13 16 17 18 20 19

Time Complexity: O(n)

Source:
http://geeksforgeeks.org/forum/topic/amazon-interview-question-2

Please write comments if you find the above code/algorithm incorrect, or find other ways to solve the same problem.

Related Tpoics:

- Given a linked list, reverse alternate nodes and append at the end
- Pairwise swap elements of a given linked list by changing links
- Self Organizing List | Set 1 (Introduction)
- Merge a linked list into another linked list at alternate positions
- QuickSort on Singly Linked List
- Delete N nodes after M nodes of a linked list
- Design a stack with operations on middle element
- Swap Kth node from beginning with Kth node from end in a Linked List

Writing code in comment? Please use **ideone.com** and share the link here.

24 Comments GeeksforGeeks

Sort by Newest

Join the discussion…

Himanshu Dagar • 3 months ago

can refer to below code for above question

http://ideone.com/vEfXwP

• Reply • Share ›

vinod • 5 months ago

We can do it in O(n) also by just taking a stack of size K and storing values of k nodes in stack than once stack is full then replace the values of those K nodes in linked list by the values in stack by popping out one by one. After then repeat the same for the next K nodes until we hit NULL...!!

• Reply • Share ›

Trying • 10 months ago

Trying • 10 months ago

[sourcecode language="JAVA"]
public class LinkListRotateK<T> implements Iterable<T>
{
transient private Element<T> head;
transient private int size;

private void writeObject(ObjectOutputStream objectOutputStream) throws IOI
{
objectOutputStream.defaultWriteObject();
objectOutputStream.writeInt(size);
Element e = head;

while(e!=null)
{
objectOutputStream.writeObject(e.t);
e = e.next;
}
}

see more

• Reply • Share ›

vinod → Trying • 5 months ago

We can do it in O(n) also by just taking a stack of size K and storing va
stack is full then replace the values of those K nodes in linked list by th
one. After then repeat the same for the next K nodes until we hit NULL.

• Reply • Share ›

Coder • 11 months ago

Nice Recursive code easily understandable. Can we have an iterative solution
or O(n) space complexity..

```
/* Paste your code here (You may delete these lines if not writing co
```

• Reply • Share ›

abhishek08aug • a year ago

Intelligent :D

• Reply • Share ›

ChellaVignesh • a year ago

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

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

/* Function to reverse the linked list */
static void reverse(struct node** head_ref,int count)
{
    int ct=0;
    struct node* prev   = NULL;
    struct node* current = *head_ref;
    struct node* next;
    while (current != NULL && ct<count)
```

see more

• Reply • Share ›

Ankit Chaudhary • a year ago

iterative version:
1. reverse k nodes
2. make next of last node of previous sub-list pointing to
first node of reversed k nodes.(temp->next=prev (in code))
3. move ahead k nodes
4. save last node of this sub-list
goto step 1

```
 /* Paste your code here (You may delete these lines if not writing co
node *reverseKAlt(node *head,int k)
{
    if(!head)
        return head;
    int i;
    node *p,*temp,*cur,*next,*prev,*t;
    p=temp=cur=next=prev=t=NULL;
    p=head;
    while(p)
```

see more

Reply • Share ›

anurag • a year ago

I have one question. Cant we just find the required nodes and
reverse the content(current-->info) inside them?

```
 /* Paste your code here (You may delete these lines if not writing co
```

Reply • Share ›

Arindam Sanyal • a year ago

#include<stdio.h>

```
#include<conio.h>
#include<stdlib.h>

struct node{
int info;
struct node *link;.
};

struct node * addtoempty(struct node *, int);.
struct node * addtoend(struct node *, int);.
void display(struct node *);.
struct node *kreverse(struct node *, int, int);.

void main(){
clrscr();
struct node *start=NULL;.
int num, d,k;
printf("n enter the number of nodes...");.
```

see more

Reply • Share ›

Anuj • a year ago

I have more concise code: pasting below:

```
class LNode
{
public:
        LNode(int data) : data(data),next(NULL) {}

        int data;
        LNode* next;
```

```
};


// enter nodes till num node
LNode* CreateList(int num)
{
        LNode* firstpos = NULL;
```

see more

• Reply • Share ›

nahid • a year ago

why to skip k item after reversing 1 set

```
/* Paste your code here (You may delete these lines if not writing co
```

• Reply • Share ›

Animesh Pratap Singh • 2 years ago

```
#include "linked_list_library.c"

struct node* alt(struct node* head, int k)
{
    struct node *mover=head,*p=head, *current ;
    int i=0;
    if(head==NULL)
    {
        return head;
    }
    while(mover!=NULL&&i<k-1)
```

```
    {
        i++;
        if(mover->next==NULL)
        {
            return head;
        }
```

see more

Reply • Share ›

Animesh Pratap Singh • 2 years ago

```
#include "linked_list_library.c"

struct node* alt(struct node* head, int k)
{
    struct node *mover=head,*p=head, *current ;
    int i=0;
    if(head==NULL)
    {
        return head;
    }
    while(mover!=NULL&&i<k-1)
    {
        i++;
        if(mover->next==NULL)
        {
            return head;
        }
```

see more

Reply • Share ›

Suresh D. Kumar • 2 years ago

Reverse alternate K nodes in a Singly Linked List

ANS:
This is my code for K reverse the linked list.
Input:
Enter no. to be reversed
2
Enter list(0 at last)
1
2
3
4
5
6
7
8
9
10

see more

• Reply • Share ›

Rajdeep • 2 years ago

One Correction in method two, without which the program crashes;

```
/* 3) If b is true, then node is the kth node.
    So attach rest of the list after node.
  4) After attaching, return the new head */
if(b == true)
{
```

```
{
     if(current)
        node->next = _kAltReverse(current,k,!b);
     return prev;
}

/* If b is not true, then attach rest of the list after prev.
  So attach rest of the list after prev */
else
{
     if(current)
        prev->next = _kAltReverse(current, k, !b);
     return node;
}
```

• Reply • Share ›

Kartik ➦ Rajdeep • 2 years ago

I don't think that the program needs if(current) condition as the function first line. Please let me know if you think otherwise or please provide a crashes.

• Reply • Share ›

sasuke • 2 years ago

Iterative solution in C#

[sourcecode language="C#"]
public void ReverseKnodes(int k)
{
Node current = Head;
Node nextNode = null;
Node prev = null;
Node start = null;

```
Node start = null;
Node end = null;
int currentPosition = 0;
int count = 0;

if (k == 0)
{
Console.WriteLine("invalid K");
return;
}
if (k == 1)
```

see more

• Reply • Share ›

naddy • 2 years ago

```
#include<stdio.h>
#include<stdlib.h>
struct linklist
{
int info;
struct linklist *next;
};
typedef struct linklist node;
node *head=NULL,*tail,*temp,*headb=NULL;;

void add(int);
void print(node*);
node *rev(node*);

int main()
{
```

```
node *p,*q;
```

see more

• Reply • Share ›

naddy ➦ naddy • 2 years ago

here given list is 1->2->3->4->5->NULL and k=3
and output is 3->2->1->4->5->NULL

• Reply • Share ›

Venki • 3 years ago

Reversing from second sub-k lists. For example,

Given linked list
1 2 3 4 5 6 7 8 9 10
modified linked list
1 2 3 6 5 4 7 8 9 10

```
struct node *kAltReverse2(struct node *head, int k)
{
    struct node *pCrawl = head;
    struct node *pRHead;
    struct node *pNext;
    // Reference to last node of first sublist
    struct node *pLastInFirstList;
    // Reference to last node of reversed (second) sublist
    struct node *pLastInSecondList;
    int count;
```

see more

• Reply • Share ›

sukhmeet2390 • 3 years ago

what is the use of step 2 in Method1.. isn't it redundant..

• Reply • Share ›

WgpShashank ➦ sukhmeet2390 • 2 years ago

@sukhmeet. step is necessary becoz we wants to reverse alternative nodes , head will points to kth , 2kth, 3kth .....nkth , so to need to make nodes , we need to set head->next to (k+1)th nodes , so that we can ju simple & then repeat the same call for another n-2k nodes ?

please let me know if i missed anything ?

```
/* Paste your code here (You may delete these lines if not writ
```

• Reply • Share ›

Tianrific • 3 years ago

Can we just reverse the whole list, and divide size of the list by K, keep the qu remainder number of nodes from the reversed list, and connect every k chunk twice, complexity = O(n).

• Reply • Share ›

Subscribe Add Disqus to your site