GeeksforGeeks

A computer science portal for geeks

Practice IDE Q&A GeeksQuiz

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
   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 =
            (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);
}
```

Run on IDE

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);
}

/*  Helper function for kAltReverse().  It reverses k nodes of the list only if
    the third parameter b is passed as true, otherwise moves the pointer k
    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;
   }
}


/* 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);
}
```

Run on IDE

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.

62 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

(Login to Rate and Mark)

3.5 Average Difficulty : **3.5/5.0**
Based on **4** vote(s)

□ Add to TODO List

□ Mark as DONE

Like Share 4 people like this.

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

Contact Us! About Us! Advertise with us!