Merge a linked list into another linked list at alternate positions

Given two linked lists, insert nodes of second list into first list at alternate positions of first list.
For example, if first list is 5->7->17->13->11 and second is 12->10->2->4->6, the first list should become 5->12->7->10->17->2->13->4->11->6 and second list should become empty. The nodes of second list should only be inserted when there are positions available. For example, if the first list is 1->2->3 and second list is 4->5->6->7->8, then first list should become 1->4->2->5->3->6 and second list to 7->8.

Use of extra space is not allowed (Not allowed to create additional nodes), i.e., insertion must be done in-place. Expected time complexity is O(n) where n is number of nodes in first list.

The idea is to run a loop while there are available positions in first loop and insert nodes of second list by changing pointers. Following are C and Java implementations of this approach.

C/C++

```
// C program to merge a linked list into another at
// alternate positions
#include <stdio.h>
#include <stdlib.h>

// A nexted list node
struct node
{
    int data;
    struct node *next;
};

/* Function to insert a node at the beginning */
void push(struct node ** head_ref, int new_data)
{
    struct node* new_node =
            (struct node*) malloc(sizeof(struct node));
    new_node->data  = new_data;
    new_node->next = (*head_ref);
    (*head_ref)  = new_node;
}

/* Utility function to print a singly linked list */
void printList(struct node *head)
{
    struct node *temp = head;
    while (temp != NULL)
```

```
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Main function that inserts nodes of linked list q into p at
// alternate positions. Since head of first list never changes
// and head of second list  may change, we need single pointer
// for first list and double pointer for second list.
void merge(struct node *p, struct node **q)
{
    struct node *p_curr = p, *q_curr = *q;
    struct node *p_next, *q_next;

    // While therre are avialable positions in p
    while (p_curr != NULL && q_curr != NULL)
    {
        // Save next pointers
        p_next = p_curr->next;
        q_next = q_curr->next;

        // Make q_curr as next of p_curr
        q_curr->next = p_next;  // Change next pointer of q_curr
        p_curr->next = q_curr;  // Change next pointer of p_curr

        // Update current pointers for next iteration
        p_curr = p_next;
        q_curr = q_next;
    }

    *q = q_curr; // Update head pointer of second list
}

// Driver program to test above functions
int main()
{
    struct node *p = NULL, *q = NULL;
    push(&p, 3);
    push(&p, 2);
    push(&p, 1);
    printf("First Linked List:\n");
    printList(p);

    push(&q, 8);
    push(&q, 7);
    push(&q, 6);
    push(&q, 5);
    push(&q, 4);
    printf("Second Linked List:\n");
    printList(q);

    merge(p, &q);

    printf("Modified First Linked List:\n");
    printList(p);

    printf("Modified Second Linked List:\n");
    printList(q);

    getchar();
    return 0;
}
```

Run on IDE

Java

```
// Java program to merge a linked list into another at
// alternate positions
class LinkedList
{
    Node head;  // head of list

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

    /* Inserts a new Node at 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;
    }

    // Main function that inserts nodes of linked list q into p at
    // alternate positions. Since head of first list never changes
    // and head of second list/ may change, we need single pointer
    // for first list and double pointer for second list.
    void merge(LinkedList q)
    {
        Node p_curr = head, q_curr = q.head;
        Node p_next, q_next;

        // While there are available positions in p;
        while (p_curr != null && q_curr != null) {

            // Save next pointers
            p_next = p_curr.next;
            q_next = q_curr.next;

            // make q_curr as next of p_curr
            q_curr.next = p_next; // change next pointer of q_curr
            p_curr.next = q_curr; // change next pointer of p_curr

            // update current pointers for next iteration
            p_curr = p_next;
            q_curr = q_next;
        }
        q.head = q_curr;
    }

    /* Function to print linked list */
    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 llist1 = new LinkedList();
        LinkedList llist2 = new LinkedList();
        llist1.push(3);
        llist1.push(2);
        llist1.push(1);

        System.out.println("First Linked List:");
        llist1.printList();

        llist2.push(8);
        llist2.push(7);
        llist2.push(6);
        llist2.push(5);
        llist2.push(4);

        System.out.println("Second Linked List:");

        llist1.merge(llist2);

        System.out.println("Modified first linked list:");
        llist1.printList();

        System.out.println("Modified second linked list:");
        llist2.printList();
    }
} /* This code is contributed by Rajat Mishra */
```

Run on IDE

Python

```
# Python program to merge a linked list into another at
# alternate positions
class LinkedList(object):
    def __init__(self):
    # head of list
        self.head = None

    # Linked list Node
    class Node(object):
        def __init__(self, d):
            self.data = d
            self.next = None

    # Inserts a new Node at front of the list.
    def push(self, new_data):

        # 1 & 2: Allocate the Node &
        # Put in the data
        new_node = self.Node(new_data)

        # 3. Make next of new Node as head
        new_node.next = self.head

        # 4. Move the head to point to new Node
        self.head = new_node

    # Main function that inserts nodes of linked list q into p at
    # alternate positions. Since head of first list never changes
    # and head of second list/ may change, we need single pointer
    # for first list and double pointer for second list.
    def merge(self, q):
```

```
        p_curr = self.head
        q_curr = q.head

        # While there are available positions in p;
        while p_curr != None and q_curr != None:

            # Save next pointers
            p_next = p_curr.next
            q_next = q_curr.next

            # make q_curr as next of p_curr
            q_curr.next = p_next # change next pointer of q_curr
            p_curr.next = q_curr # change next pointer of p_curr

            # update current pointers for next iteration
            p_curr = p_next
            q_curr = q_next
        q.head = q_curr

    # Function to print linked list
    def printList(self):
        temp = self.head
        while temp != None:
            print str(temp.data),
            temp = temp.next
        print ''

# Driver program to test above functions
llist1 = LinkedList()
llist2 = LinkedList()
llist1.push(3)
llist1.push(2)
llist1.push(1)

print "First Linked List:"
llist1.printList()

llist2.push(8)
llist2.push(7)
llist2.push(6)
llist2.push(5)
llist2.push(4)

print "Second Linked List:"

llist2.printList()
llist1.merge(llist2)

print "Modified first linked list:"
llist1.printList()

print "Modified second linked list:"
llist2.printList()

# This code is contributed by BHAVYA JAIN
```

Run on IDE

Output:

```
First Linked List:
1 2 3
Second Linked List:
4 5 6 7 8
```

```
Modified First Linked List:
1 4 2 5 3 6
Modified Second Linked List:
7 8
```

This article is contributed by **Chandra Prakash**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Company Wise Coding Practice Topic Wise Coding Practice

75 Comments Category: Linked Lists

Related Posts:

- Convert a Binary Tree to a Circular Doubly Link List
- Subtract Two Numbers represented as Linked Lists
- Rearrange a given list such that it consists of alternating minimum maximum elements
- Flatten a multi-level linked list | Set 2 (Depth wise)
- Decimal Equivalent of Binary Linked List
- Merge K sorted linked lists
- Check if a linked list is Circular Linked List
- Delete middle of linked list

(Login to Rate and Mark)

1.8 Average Difficulty : **1.8/5.0**
Based on **34** vote(s)

Add to TODO List

Mark as DONE

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

75 Comments **GeeksforGeeks** 1 **Login**

Recommend 5 **Share** **Sort by Newest**

Join the discussion…

Junaid Alam • 3 months ago

Simple C++ solution which works for all kind of input using only two extra pointers....
http://code.geeksforgeeks.org/...

• Reply • Share ›

Leo Fernandez • 3 months ago

c# implementation
http://ideone.com/vfFxyj

• Reply • Share ›

Venkata Pavan • 4 months ago

Please check the code here in c++ (Not the code written in C):
http://code.geeksforgeeks.org/...

Please let me know your comments.

This works for all combinations of the input lengths, except zero. Anyways, linked list of length zero can be considered non existent.

• Reply • Share ›

bhavik gujarati • 5 months ago

My solution:

```
void mergeAlternate(Node *head1, Node **head2) {
    Node *node1 = head1;
    Node *node2 = *head2;

    while(node1 && node2) {
        Node *temp = node1->next;
        node1->next = node2;
        *head2 = node2->next;
        node2->next = temp;
        node1 = node1->next->next;
        node2 = *head2;
    }
}
```

• Reply • Share ›

Nikhil Chaudhary • 5 months ago

```
//Check this
//Should work for unequal length also

void alternate_insert(struct node* h1, struct node* h2)
{
struct node* temp1=h1;
struct node* temp2=h2;
while(temp1||temp2)
{
temp2=h2;
h2=h2->next;
temp2->next=temp1->next;
temp1->next=temp2;
temp1=temp1->next->next;
}
}
```

▲ | ▼ • Reply • Share ›

Ajay Vijayasarathy • 6 months ago

http://code.geeksforgeeks.org/...

Works for Lists with un-equal lengths also

▲ | ▼ • Reply • Share ›

Tushar Saha • 8 months ago

```
void mergeList(struct node *head1,struct node *head2){

int a = 1;

struct node * curr=NULL;

struct node * curr1=head1;

while(head2 != NULL && head1 != NULL){

struct node * temp = (struct node*)malloc(sizeof(struct node));

temp->data = (head2)->data;

temp->next = NULL;

if(a==1){

curr = (head1)->next;

(head1)->next = temp;
```

see more

▲ | ▼ • Reply • Share ›

Lokesh • a year ago

http://code.geeksforgeeks.org/...

^ | v • Reply • Share ›

Hardik Sharma • a year ago

```
void Merge_Alternate(struct node*a, struct node**b)

{

struct node*temp;

while (a&&(*b))

{

temp = a->next;

a->next = *b;

*b = (*b)->next;

a->next->next = temp;

a = temp;

}

}
```

^ | v • Reply • Share ›

Klaus • a year ago

http://ideone.com/tVXwVd .This one works even when the linked lists are of unequal length.

^ | v • Reply • Share ›

Shivanshu Chauhan • a year ago

the code is not working

^ | v • Reply • Share ›

pragya saxena ➚ Shivanshu Chauhan • 5 months ago

yes different stacks should be maintained for both the list

^ | v • Reply • Share ›

Naveen Khatri • 2 years ago

```
//Nice and Easy
void merge(struct node *p, struct node **q)

{
```

```
struct node *curr1=p,*curr2=*q;

struct node *temp,*next;

while(curr1!=NULL&&curr2!=NULL)

{

next=curr1->next;

curr1->next=curr2;

curr2=curr2->next;

curr1->next->next=next;

curr1=curr1->next->next;

}

*q=curr2;

}
```


Naveen Khatri • 2 years ago

http://ideone.com/Q3H4vZ

Ashish Jaiswal • 2 years ago

```
#include<stdio.h>
#include<stdlib.h>
struct node;
void push(struct node**,int);
void merge(struct node*,struct node**);
void print(struct node*);
typedef struct node
{
int data;
struct node*next;
}Node;

int main()
{
Node*head1=NULL;
Node*head2=NULL;
push(&head1,43);
push(&head1,67);
```

see more

• Reply • Share ›

Narendra • 2 years ago

@SG Seth:

There are two problems

1) temp-temp->next observer minus here which leads to loop . it should be temp=temp->next
2) head2=ptr1; will not be reflected in main caller as it is not double pointer

• Reply • Share ›

SG Seth • 2 years ago

can someone help to correct this code:

```
void altmerge(node *head1, node *head2)

{node *ptr=head1, *ptr1= head2;

node *temp=ptr->next;

while( ptr1!=NULL)

{if(temp==NULL)

{ptr->next=ptr1;

ptr1=ptr1->next;

ptr->next->next=temp;

ptr=temp;

break;
```

see more

• Reply • Share ›

Chanakya Nani • 2 years ago

http://ideone.com/14eKU4

A recursive solution:

```
node* MergeAlt(node* a,node* b){
if(a==NULL )
return b;
if(b==NULL )
return a;
node* temp = a->next;
node* temp2 = b->next;
```

```
a->next = b;
b->next = MergeAlt(temp,temp2);
return a;
}
```

• Reply • Share ›

Laxmi Udaseen • 2 years ago

A better solution is:

```
node *mergeTwoLists(node **first, node **second) {

node *p = *first;
node *q = *second;
node *result = NULL;
*first = *second = NULL;
// if any one of the lists is empty
if(p == NULL || q == NULL) {
result = (p == NULL ? q : p);
}
else {
result = p;
while(p != NULL && q != NULL) {
node *tmp = p->next;
p->next = q;
p = q;
q = tmp;
}
}
*first = *second = NULL;
return result;
}
```

3 • Reply • Share ›

RK- An Unproven Theorem ➔ Laxmi Udaseen • 2 years ago

@Laxmi Udaseen: Thanks for your solution..

• Reply • Share ›

Vivek Gupta • 2 years ago

//Considering Following Lists: root1, root2

```
struct node *l1 = root1; // Another pointer so that root node is preserved
void merge()
{
while (true)
{
if (l1 == NULL || root2 == NULL)
break;
```

```
break;
Node *temp = root2;
root2 = root2 -> next;
temp->next = l1->next;
l1->next = temp;
l1 = temp->next;
}
}
```

• Reply • Share ›

codecrecker • 2 years ago

```
#include<stdio.h>

#include<stdlib.h>

typedef struct node n;

struct node{

int data;

n* next;

};

n *head,*cur,*head2;

void insert(int data,int j)

{

n *p;
```

see more

• Reply • Share ›

f2qbook • 2 years ago

```
#include<iostream>
/*inplace changed */
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

using namespace std;

struct node

{

int v;
```

```
node *next;

};
```

see more

• Reply • Share ›

munjal • 2 years ago

how about doing it recursively!!!

I will merge the list like first list is 1->2->3 and second list is 4->5->6->7->8, then FINAL list should become 1->4->2->5->3->6->7->8

ptr1 is pointer to first list(starting)
ptr2 is pointer to another list(starting)

calling to the function would be like merge(head1,head2).

```
---------------------------------------------
struct node *merge(struct node *ptr1,struct node *ptr2)
{
if(ptr1==NULL)
return ptr2;

else if(ptr2==NULL)

return ptr1;
else
```

see more

1 • Reply • Share ›

Vishal • 2 years ago

```
Node *MergeList(Node *head1, Node *head2)
{
Node *t = head1;
while(head2 && t)
{
Node *alt = t->next;
t->next = head2;
head2=head2->next;
if(alt) t->next->next = alt;
t = alt;
}
return head1;
}
```

• Reply • Share ›

kamran siddique • 2 years ago

http://ideone.com/vT8KiE

 • Reply • Share ›

Arnab • 2 years ago

```
void merge(struct node *p, struct node **q)

{

/*printf("%d %d",p->data,(*q)->data);*/

struct node *start1=p;

struct node *start2=(*q);

struct node *curr1=start1;

struct node *curr2=start2;

struct node *temp1=NULL;

struct node *temp2=NULL;

while(curr1!=NULL)

{
```

see more

 • Reply • Share ›

codecrecker • 2 years ago

```
void merge()
{
n *t1=head,*t2=head2;n *tmp,*tmp2;
while(1)
{

if(t1->next)
tmp=t1->next;
else
{
t1->next=t2;break;

}
if(t2->next)
tmp2=t2->next;
else
{
t1->next=t2;
```

see more

• Reply • Share ›

Mohaan Raja • 2 years ago

Can be done liket his too....

http://codepad.org/CGSt5fnA

• Reply • Share ›

Abhishek Kashyap • 2 years ago

another simple implementation..

```
mergelists(struct node *a,struct node **head)
{
struct node *t1,*t2;
t1=a;
while( (*head)!=NULL && t1!=NULL)
{

t2=*head;
*head=(*head)->next;
t2->next=t1->next;
t1->next=t2;
t1=t1->next->next;

}

}
```

• Reply • Share ›

Abhinav Bhardwaj • 2 years ago

Merging without extra space

```
public void MergeList(Node node1,Node node2)
{
Node temp=null;
Node current1=node1;
Node current2=node2;
Node temp2=null;

while(current1!=null && current2!=null)
{
temp=current1.next;
current1.next=current2;
temp2=current2.next;
current2.next=temp;
current2=temp2;
```

```
current1=current1.next.next;
}
}
```

• Reply • Share ›

sangee • 2 years ago

In the question it is said "Use of extra space is not allowed (Not allowed to create additional nodes),"..so why are all des nodes created?

• Reply • Share ›

Ashish Jaiswal ➚ sangee • 2 years ago

These are node variable...not extra space like stack ,array or others

• Reply • Share ›

Ajay Singh ➚ sangee • 2 years ago

see comment of OP CODER below

• Reply • Share ›

MK • 2 years ago

O(n) when both are same length 'n' in worst case. Am I correct?

• Reply • Share ›

Abhi shyam • 2 years ago

//please inform if any wrong to chabhishyam@gmail.com or please reply

```
#include<stdio.h>
#include<iostream>
#include<stdlib.h>
using namespace std;
struct ll{
int val;
struct ll *next ;
}*root1=NULL,*root2=NULL,*temp;

void mix(struct ll *head1,struct ll *head2)
{
struct ll *temp1,*temp2,*temp3,*temp4;
temp1=head1;
temp2=head2;
while(temp1!=NULL && temp2!=NULL)
{
temp3=temp1->next;
```

see more

• Reply • Share ›

ANA ➚ Abhi shyam • 2 years ago

its correct

• Reply • Share ›

SANTOSH KUMAR MISHRA • 2 years ago

```
#include<stdio.h>
#include<malloc.h>
struct node
{
int data;
struct node *next;
};
typedef struct node node;
node *InsertNode(node *head,int data);
void Display(node * head);
node *MergeList(node *head);
int main(void)
{
node *head = NULL;
int i,ch;
do
{
printf("\n\nPress < 01 > for Insert Node.");
```

see more

• Reply • Share ›

kaushik Lele • 2 years ago

Use of extra space is not allowed (Not allowed to create additional nodes). But you have created node references. I thought we need to complete task without creating additional variable.

• Reply • Share ›

OP Coder ➚ kaushik Lele • 2 years ago

You got it wrong. He only meant that we need to utilize the nodes which are already there in the two lists instead of creating anther node. ie creating new node using heap memory.

Creating node reference (only a pointer) is different from creating new node(node *n = new node()).
Hope this helps.

• Reply • Share ›

Ashish Jaiswal ➚ OP Coder • 2 years ago

Really helpful....thankss

• Reply • Share ›

Pranav • 2 years ago

```
node* mergeAlt(node* p,node* q)
{
if(p==NULL)
return q;
if(q==NULL)
return p;
//1-2-3-4
//6-7-8-9-10
node* temp = p;
node* p_prev = p;
while(p!=NULL && q!=NULL)
{
node* p_nxt = p->next;
node* q_nxt = q->next;

p->next = q;
q->next = p_nxt;
p_prev = p;
```

see more

• Reply • Share ›

Pranav Sawant ➚ Pranav • 2 years ago

Apologies for this,
please refer the below mentioned code

```
node* mergeAlt(node* p,node* q)
{
if(p==NULL)
return q;
if(q==NULL)
return p;
//1-2-3-4
//6-7-8-9-10
node* temp = p;

while(p!=NULL && q!=NULL)
{
node* p_nxt = p->next;
node* q_nxt = q->next;

p->next = q;
```

see more

1 • Reply • Share ›

Tarzan • 2 years ago

This solution does not consider the case where the first list is empty while the second list is not.

void merge(struct node **p, struct node **q){

if(*p==NULL){
*p = *q;
return;
}

• Reply • Share ›

Pranav ➚ Tarzan • 2 years ago

node* mergeAlt(node* p,node* q)

{

if(p==NULL)

return q;

if(q==NULL)

return p;

//1-2-3-4

//6-7-8-9-10

node* temp = p;

node* p_prev = p;

while(p!=NULL && q!=NULL)

see more

• Reply • Share ›

jim302 • 3 years ago

Above program fails when first list has less node than second list.
Example:
firstlist:1->2->NULL
secondlist: 11->12->13->14->NULL
Please check once.

• Reply • Share ›

danny • 3 years ago

Simple Implementation with O(1) space complexity.....
Please suggest me if you find some fault

void merge(struct node *head,struct node **root)

```
{
struct node *p=head,*q,*t=(*root);
while(p)
{
if(t!=NULL)
{
q=t;
t=t->next;
q->next=p->next;
p->next=q;
}
p=p->next->next;
}
*root=t;
}
```


Abhilash Kumar • 3 years ago

in the solution given above the header value of q is updated .
is it necessary to do this .

Anand • 3 years ago

```
/*Merge a linked list into another linked list at alternate positions*/
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>

struct l
{
int data;
struct l *link;
};

void add(struct l **,int );
void merge(struct l**,struct l **);
void display(struct l*);

int main()
{
struct l *r2,*r1;
r1 = NULL;
```


yogeshgfg • 3 years ago

struct node* mergeAlternative (struct node* start1, struct node* start2)

```
struct node* mergeAlternative ( struct node* start1, struct node* start2 )
{
if(( start1 == NULL)||(start2 == NULL) )
{
return start1;
}
else{
struct node* temp = start1->next;
start1->next = start2;
mergeAlternative(start2,temp);
return start1;
}
}
```

• Reply • Share ›

Load more comments