GeeksforGeeks

A computer science portal for geeks

Merge two sorted linked lists

Write a SortedMerge() function that takes two lists, each of which is sorted in increasing order, and merges the two together into one list which is in increasing order. SortedMerge() should return the new list. The new list should be made by splicing
together the nodes of the first two lists.

For example if the first linked list a is 5->10->15 and the other linked list b is 2->3->20, then SortedMerge() should return a pointer to the head node of the merged list 2->3->5->10->15->20.

There are many cases to deal with: either ‘a’ or ‘b’ may be empty, during processing either ‘a’ or ‘b’ may run out first, and finally there’s the problem of starting the result list empty, and building it up while

going through ‘a’ and ‘b’.

Method 1 (Using Dummy Nodes)
The strategy here uses a temporary dummy node as the start of the result list. The pointer Tail always points to the last node in the result list, so appending new nodes is easy.
The dummy node gives tail something to point to initially when the result list is empty. This dummy node is efficient, since it is only temporary, and it is allocated in the stack. The loop proceeds, removing one node from either ‘a’ or ‘b’, and adding it to tail. When
we are done, the result is in dummy.next.

```
/*Program to alternatively split a linked list into two halves */
#include<stdio.h>
#include<stdlib.h>
#include<assert.h>

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

/* pull off the front node of the source and put it in dest */
void MoveNode(struct node** destRef, struct node** sourceRef);

/* Takes two lists sorted in increasing order, and splices their nodes togeth
struct node* SortedMerge(struct node* a, struct node* b)
{
   /* a dummy first node to hang the result on */
   struct node dummy;

   /* tail points to the last result node  */
   struct node* tail = &dummy;

   /* so tail->next is the place to add new nodes
     to the result. */
   dummy.next = NULL;
   while(1)
   {
      if(a == NULL)
      {
         /* if either list runs out, use the other list */
         tail->next = b;
         break;
      }
      else if (b == NULL)
      {
         tail->next = a;
         break;
      }
      if (a->data <= b->data)
      {
         MoveNode(&(tail->next), &a);
```

```
        }
        else
       {
          MoveNode(&(tail->next), &b);
       }
       tail = tail->next;
    }
    return(dummy.next);
}


/* UTILITY FUNCTIONS */
/*MoveNode() function takes the node from the front of the source, and move i
    It is an error to call this with the source list empty.

    Before calling MoveNode():
    source == {1, 2, 3}
    dest == {1, 2, 3}

    Affter calling MoveNode():
    source == {2, 3}
    dest == {1, 1, 2, 3}
*/
void MoveNode(struct node** destRef, struct node** sourceRef)
{
  /* the front source node  */
  struct node* newNode = *sourceRef;
  assert(newNode != NULL);

  /* Advance the source pointer */
  *sourceRef = newNode->next;

  /* Link the old dest off the new node */
  newNode->next = *destRef;

  /* Move dest to point to the new node */
  *destRef = newNode;
}


/* Function to insert a node at the beginging of the linked list */
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 nodes in a given linked list */
void printList(struct node *node)
{
  while(node!=NULL)
  {
   printf("%d ", node->data);
   node = node->next;
  }
}

/* Drier program to test above functions*/
int main()
{
  /* Start with the empty list */
  struct node* res = NULL;
  struct node* a = NULL;
  struct node* b = NULL;

  /* Let us create two sorted linked lists to test the functions
   Created lists shall be a: 5->10->15,  b: 2->3->20 */
  push(&a, 15);
  push(&a, 10);
  push(&a, 5);

  push(&b, 20);
  push(&b, 3);
  push(&b, 2);

  /* Remove duplicates from linked list */
  res = SortedMerge(a, b);

  printf("\n Merged Linked List is: \n");
  printList(res);

  getchar();
  return 0;
}
```

Method 2 (Using Local References)
This solution is structurally very similar to the above, but it avoids using a dummy node. Instead, it maintains a struct node** pointer, lastPtrRef, that always points to the last pointer of the result list. This solves the same case that the dummy node did — dealing with the result list when it is empty. If you are trying to build up a list at its tail, either the dummy node or the struct node** "reference" strategy can be used (see Section 1 for details).

```
struct node* SortedMerge(struct node* a, struct node* b)
```

```
{
  struct node* result = NULL;

  /* point to the last result pointer */
  struct node** lastPtrRef = &result;

  while(1)
  {
    if (a == NULL)
    {
      *lastPtrRef = b;
       break;
    }
    else if (b==NULL)
    {
       *lastPtrRef = a;
       break;
    }
    if(a->data <= b->data)
    {
      MoveNode(lastPtrRef, &a);
    }
    else
    {
      MoveNode(lastPtrRef, &b);
    }

    /* tricky: advance to point to the next ".next" field */
    lastPtrRef = &((*lastPtrRef)->next);
  }
  return(result);
}
```

Method 3 (Using Recursion)
Merge is one of those nice recursive problems where the recursive solution code is much cleaner than the iterative code. You probably wouldn't want to use the recursive version for production code however, because it will use stack space which is proportional to the length of the lists.

```
struct node* SortedMerge(struct node* a, struct node* b)
{
  struct node* result = NULL;

  /* Base cases */
  if (a == NULL)
     return(b);
  else if (b==NULL)
     return(a);

  /* Pick either a or b, and recur */
  if (a->data <= b->data)
  {
```

```
        result = a;
        result->next = SortedMerge(a->next, b);
    }
    else
    {
        result = b;
        result->next = SortedMerge(a, b->next);
    }
    return(result);
}
```

Source: http://cslibrary.stanford.edu/105/LinkedListProblems.pdf

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

Related Topics:

- Clone a linked list with next and random pointer | Set 2
- Given a linked list of line segments, remove middle points
- Construct a Maximum Sum Linked List out of two Sorted Linked Lists having some Common nodes
- 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

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

Aryan Parker • 21 days ago

Please check my easy solution for any bugs-
http://ideone.com/Mo5JFc

magicsign • a month ago

A clean solution with C# :

```
static LinkedList<int> merge(LinkedList<int> n1, LinkedList<int> n2) {
```

```
int[] res1 = n1.ToArray();

int[] res2 = n2.ToArray();

int[] res = new int[res1.Length+res2.Length];

Array.Copy(res1, 0, res, 0, res1.Length);

Array.Copy(res2,0,res,res1.Length,res2.Length);

Array.Sort(res);

LinkedList<int> merged = new LinkedList<int>(res);

return merged;

}
```

Author : Noe Dadon

• Reply • Share ›

Navin Purohit • 2 months ago

i do iterative method for this,but i stuck due to segmentation fault,can someone help me,why this happen?

i also not understand how to store the 3rd link list head,please someone help me out.....
my code here..

```
struct Node* MergeLists(Node *headA, Node* headB)

{

struct Node* a=(struct Node*)malloc(sizeof(struct Node));

struct Node *tempA=headA,*tempB=headB,*prv=NULL;

if(tempA==NULL&&tempB)

return tempB;

if(tempB==NULL&&tempA)
```

see more

• Reply • Share ›

Audi • 2 months ago

If you donot want to use recursive approach or Dummy nodes

You can try something like this
-- Reverse both the list
-- Now insert bigger element into the list
Basicaly by this smaller elements will be inserted in the beginning.

P.S - This is Whatsapp Coding Challenge Question

2 • Reply • Share ›

shivi • 3 months ago

what is the time complexity of the recursive solution ?

1 • Reply • Share ›

<HoldOnLife!#> • 3 months ago

In recursive approach , result = a; shouldn't be result->data=a->data and same for result=b ..??

• Reply • Share ›

<HoldOnLife!#> <HoldOnLife!#> • 3 months ago

Alright, well it will work too but for that we need to allocate heap section to result

• Reply • Share ›

tintin • 6 months ago

\\Java implementation\\

```
public static void merge(LinkedList finalList, Node head1, Node head2) {

Node firstCurrent = head1;

Node secondCurrent = head2;

while( firstCurrent != null && secondCurrent != null ) {

if ( firstCurrent.data < secondCurrent.data) {

finalList.insert(firstCurrent.data);

firstCurrent = firstCurrent.next;

} else if ( firstCurrent.data > secondCurrent.data ) {

finalList.insert(secondCurrent.data);

secondCurrent = secondCurrent.next;
```

see more

• Reply • Share ›

Ankit Srivastava • 6 months ago

```
public static Node mergeSortedListWithoutRecurrsion(Node a,Node b)
{
Node head=null, last=null, newNode=null;
int i=0;
while(a!=null && b!=null)
{
if(a.data <= b.data)
{
newNode = new Node(a.data,null);
if(i==0) head = newNode;
else
last.next = newNode;
a = a.next;
last = newNode;
i++;
}
else
{
```

see more

• Reply • Share ›

Neil • 7 months ago

IT IS JUST THE UNION OF TWO SORTED LISTS. HERE'S MY CODE

```
struct listNode* unionOfLists(struct listNode *firstList, struct listNode *secondList){

/* Step 1. Declare the dummy node and let the tail point to it. */

struct listNode dummy;

/* Note that the initialization of the dummy node is very important. Specifically, if you do not
make its next pointer NULL, it will create a big problem. */

dummy.data = 0;

dummy.next = NULL;

struct listNode *tail =&dummy;

/* Step 2. Iterate through the list till both becomes empty. */

while(firstList != NULL || secondList != NULL){
```

see more

• Reply • Share ›

shail • 8 months ago

Can someone explain me Method 2 (Using Local References)? I didn't understand when/how we store address of head of merged linked list in "result" valriable.

• Reply • Share ›

nobody • 9 months ago

Can sumone pls explain me the dummy node concept? I dont get t at all . thanx

• Reply • Share ›

guest ↗ nobody • 9 months ago

download this pdf http://cslibrary.stanford.edu/...
and read page 5-8

1 • Reply • Share ›

nobody ↗ guest • 9 months ago

thanku :))

• Reply • Share ›

typing.. • 10 months ago

niceeeee..... I like dummy node concept.... but it is not required if the question demands in place merging..

1 • Reply • Share ›

Gautam Goyal • 10 months ago

Another clean solution, choose on the basis of head nodes, the final head and then from second list, insert wherever required.

Time complexity: O(m+n)

Space Complexity:O(1)

```
typedef struct node* Link;

Link sortedMerge(Link* head1, Link* head2){

if(head1 == NULL)
return *head2;

if(head2 == NULL)
return *head1;

Link temp1,temp2,start,end,tempo,head;

if((*head1)->data > (*head2)->data){
```

see more

• Reply • Share ›

Arun • 10 months ago

The idea is using a dummy node is the crux. Its smart!

Rest of the code could be simple and straightforward as in http://ideone.com/IvVXxL.

```
Node * MergeLists( Node * headA, Node * headB ) {

Node dummy;
Node * tail = & dummy;
Node * pa = 0;
Node * pb = 0;

for( pa = headA, pb = headB; pa && pb; ) {

if( pa->data <= pb->data ) {
tail->next = pa;
tail = pa;
pa = pa->next;
}
else {
```

see more

• Reply • Share ›

thatsme ➚ Arun • 9 months ago

thanks for sharing...i liked your code :)

• Reply • Share ›

AMIT JAMBOTKAR • a year ago

JAVA IMPLEMENTATION OF METHOD 1 AND METHOD 3

```
* @author Virus

*/

public class LinkedList<e extends="" number=""> implements Cloneable {

Node<e> head = null;

Node<e> tail = null;

// Adding at the End

class Node<t extends="" number=""> {
```

T value;

Node<t> nextReference;

public Node(T value) {

see more

• Reply • Share ›

Himanshu Dagar • a year ago

For combined code we can refer to below Ideone link

http://ideone.com/wzxhi1

• Reply • Share ›

Guest • a year ago

```
typedef struct node
{
int data;
struct node* next;
} Node;

Node* mergeSortedLists(Node* l1, Node* l2) {
if (l1->data > l2->data) {
std::swap(l1, l2);
}

Node* c1 = l1;
Node* c2 = l2;
Node* p1 = NULL;
Node* p2 = NULL;
```

see more

1 • Reply • Share ›

Shikha Gupta • a year ago

how can i simplify the following code
http://ideone.com/2naUy4

• Reply • Share ›

Deepak • a year ago

Is it possible to solve the problem when the two lists are in reverse order??

• Reply • Share ›

Deepak ➚ Deepak • a year ago

I am talking about complexity O(n)

• Reply • Share ›

Lisper • 2 years ago

You can make the recursive version tail-recursive to make it as efficient as the iterative version. I will use DrRacket (a lisp dialect).

```
(define (merge-sorted lst1 lst2)
(merge lst1 lst2 empty))
(define (merge lst1 lst2 acc)
(cond [ (empty? lst1) (append acc lst2)]
[(empty? lst2) (append acc lst1)]
[else
(if (<= (first lst1) (first lst2))
(merge (rest lst1) lst2 (append acc (first lst1)))
(merge lst1 (rest lst2) (append acc (first lst2))))]))
```

• Reply • Share ›

Dharmendra Prasad • 2 years ago

Someone might also need the java version... here I have written the java code for the same. http://techieme.in/techieme/me...

1 • Reply • Share ›

Techie Me ➚ Dharmendra Prasad • 8 days ago

The link has actually changed. Here is the new link http://techieme.in/merging-two...

• Reply • Share ›

Vijay Apurva • 2 years ago

```
struct node * mergelist(struct node *q , struct node *w){.

struct node *result,*temp ;.

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

if(q==NULL){
return w;
}
if(w==NULL){
return q;
}

if(w->data < q->data).
```

```
{

temp->data=w->data;.

temp->next=NULL;.
result=temp;
```

see more

• Reply • Share ›

vijay apurva • 2 years ago

```
struct node * mergelist(struct node *q ,struct node *w){

struct node *result,*temp ;

temp= malloc(sizeof(struct node));

if(q==NULL){
return w;
}
if(w==NULL){
return q;
}

if(w->data data)
{
temp->data=w->data;
temp->next=NULL;
result=temp;
result->next = mergelist(q,w->next);
```

see more

• Reply • Share ›

pranjalgupta • 2 years ago

```
We can do inplace by the following function.
this also implies that merge sort can be done on linked lists inplace.
list* sortedmerge(list *head1, list *head2)
{
list *head,*tail;
if(head1->data>head2->data)
{
head=head2;
tail=head;
head2=head2->next;
}
```

```
else
{
head=head1;
tail=head;
head1=head1->next;
}
while(head1!=NULL && head2!=NULL)
```

see more

6 • Reply • Share ›

Naveen Khatri pranjalgupta • 2 months ago
nice algorithm
• Reply • Share ›

its_dark pranjalgupta • 5 months ago
Your code would fail if any of head1 or head2 is NULL
• Reply • Share ›

DeKryptonite pranjalgupta • 5 months ago
I think this line right at the top of your function:

if(head1->data > head2->data)

should be:

if(head1->data < head2->data)

This way, the greater value is placed at the head of the list.
• Reply • Share ›

Naveen Khatri DeKryptonite • 2 months ago
no , its right !!
• Reply • Share ›

Nitin Pallindrome pranjalgupta • a year ago
GOOD solution dude.....nice inplace algo ....
• Reply • Share ›

Arun Kumar Mohapatra • 2 years ago

```
Node** mergeTwoSortedList(Node *head1, Node* head2).
{
Node *tail = 0x0;.
Node *head = 0x0;.
if(head1 == 0x0 & head2 == 0x0).
```

return 0x0;
if(head1 == 0x0 & head2!= 0x0).
return &head2;
if(head1!= 0x0 & head2 == 0x0).
return &head1;

while(head1!= 0x0 && head2!= 0x0).
{
if(head1->item_ <= head2->item_).
{
if(head == 0x0).
{
head = new Node();.

see more

1 • Reply • Share ›

abhishek08aug • 2 years ago

Intelligent :D

• Reply • Share ›

Will • 2 years ago

Super stuck on a homework question.

```
 /* These are all nice and jazzy, but what if you need to write a recursive void function
that merges already sorted X-list & Y-list, into one sorted list (Z-list) without
creating new nodes? */
void SortedMergeRecur(Node*& headX, Node*& headY, Node*& headZ)
{
     //  How do I do this?
}
```

• Reply • Share ›

Will ➚ Will • 2 years ago

/*
Note:
Nevermind solved my own question.
Problem I was having: I was changing headZ then passing it, when I should have been just passing the link without changing original pointer. Hope this helps anybody out there that might be stuck in same boat.

Here is the working code...

```
*/

[code language="cpp"]
void SortedMergeRecur(Node*& headX, Node*& headY, Node*& headZ)
{
if (headX == 0 && headY == 0) return; // base case (both X&Y Empty)

if(headX != 0 && headY != 0) // if: X&Y Not Empty
{
if(headX->data <= headY->data) // do sorting
```

see more

• Reply • Share ›

Sanchita Ghai • 2 years ago

we can do it thiscway too-

```
void push(node **head, int d).
{
node *p=(node*)malloc(sizeof(node));
p->next=*head;
p->data=d;
*head=p;

}
node* merge(node *p, node *q).
{ node *head3=NULL;
while ((p) && (q)).
{
if(p->data > q->data).
{
push(&head3, q->data);
q=q->next;
}
```

see more

• Reply • Share ›

anonymous • 2 years ago

Here is a clean recursive solution using double pointers.
Please verify this.

```
void sorted_merge(struct node **head_ref,struct node *a,struct node* b)
{
if(a==NULL)
{
```

```
*head_ref=b;
return;
}
if(b==NULL)
{
*head_ref=a;
return;
}
if(a->data data)
{
*head_ref=a;
sorted_merge(&(*head_ref)->next,a->next,b);
}
else
{
*head_ref=b;
sorted_merge(&(*head_ref)->next,a,b->next);
}
}
```

• Reply • Share ›

Kabeer • 2 years ago

```
 node *merge(node *a,node *b)
{
node *head,*tail;
if(a==null && b==null)
return null;
else if(a==null)
return b
else if(b==null)
return a
else
{
if(a->data<=b->data)
{
head=a;
tail=a;
a=a->next;
}
else
```

see more

• Reply • Share ›

Anup Ravikumar • 2 years ago

Dude Rahul Use a typedef for ur struct.... I.e typedef struct NODE *ndptr ..... Now it will be even simpler to understand

 • Reply • Share ›

Rahul Gahlaut • 2 years ago

I can do it in easy way -

/* author : rahul G */.
/**************************
* merge two link list in sorted order.
with o(n) *********/.

/* function to insert node */.
void insert (NODE head , NODE n) ;.

/* merge two list */.
struct NODE * merge_list(NODE n1 , NODE n2).
{
struct NODE head = NULL ;.
if (n1 == NULL).
{.

head = n2 ;.

---

see more

 • Reply • Share ›

Sonam Priya • 2 years ago

Add a comment...

 • Reply • Share ›

aygul • 2 years ago

Iterative c# implementation. AdvanceSmallHead advances smaller head until it becomes grater than the grater head. then sets previous ones next to the greater one. In MergeTwoSortedList whichever the list contains the smaller first element, is returned.

Any Comments ?

```
 public static LNode MergeTwoSortedList(LNode head1, LNode head2)
{
    if (head1 == null) return head2;
    if (head2 == null) return head1;

    LNode retHead = head1.Data < head2.Data ? head1 : head2;
```

```
    while (head1 != null && head2 != null)
        if (head1.Data <= head2.Data)
            AdvanceSmallHead(ref head1, ref head2);
        else
            AdvanceSmallHead(ref head2, ref head1);
```

see more

• Reply • Share ›

deep • 2 years ago

i am not getting the use of dummy node in 1st method. sud not be it declare as struct node *dummy,

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

• Reply • Share ›

All • 3 years ago

Thanks

• Reply • Share ›

jayesh • 3 years ago

Here is the cleaner iterative solution using only 1 function

```
LNode* mergeList(LNode* a, LNode* b){
   printf("mergelist\n");
   LNode dummy;
   LNode* tail = &dummy;
   while(true){
       if(a == NULL){
           tail->next = b;
           break;
       }
       if(b == NULL){
           tail->next = a;
           break;
       }
       if(a->info <= b->info){
           tail->next = a;
           tail = tail->next;
```

see more

1 • Reply • Share ›

iami jayesh • 3 months ago

both if and else have "tail = tail->next;"
you can remove them and put it as the last line in the while loop

• Reply • Share ›

Load more comments

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

- Follow @GeeksforGeeks  Subscribe

Recent Comments

- Baba

  Traverse the tree in inorder fashion and insert...

  Populate Inorder Successor for all nodes · 54 minutes ago

- creeping_death

  Ruby solution, slightly different, easier to...

  Longest Even Length Substring such that Sum of First and Second Half is same · 1 hour ago

- darkprotocol

  Anyone round-1 4th question?

  Amazon Interview | Set 121 (On-Campus for SDE-1) · 1 hour ago

- S.Nkm

  Would a simple answer involving the Unix epoch...

  Find number of days between two given dates · 1 hour ago

- Shomina

  Found Real Job INterview Questions Here...

  Flipkart Interview Experience | Set 20 (For SDE-II) · 2 hours ago

- Shimona

  Found Real Job INterview Questions here...

  Myntra Interview Experience | Set 4 (For Senior Software Engineer) · 2 hours ago