

Intersection of two Sorted Linked Lists

Given two lists sorted in increasing order, create and return a new list representing the intersection of the two lists. The new list should be made with its own memory — the original lists should not be changed.

For example, let the first linked list be 1->2->3->4->6 and second linked list be 2->4->6->8, then your function should create and return a third list as 2->4->6.

Method 1 (Using Dummy Node)

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.

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

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

void push(struct node** head_ref, int new_data);

/*This solution uses the temporary dummy to build up the result list */
struct node* sortedIntersect(struct node* a, struct node* b)
{

```

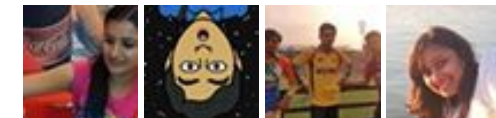
Google™ Custom Search



GeeksforGeeks



53,528 people like [GeeksforGeeks](#).



[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

```

struct node dummy;
struct node* tail = &dummy;
dummy.next = NULL;

/* Once one or the other list runs out -- we're done */
while (a != NULL && b != NULL)
{
    if (a->data == b->data)
    {
        push((&tail->next), a->data);
        tail = tail->next;
        a = a->next;
        b = b->next;
    }
    else if (a->data < b->data) /* advance the smaller list */
        a = a->next;
    else
        b = b->next;
}
return (dummy.next);
}

/* UTILITY FUNCTIONS */
/* Function to insert a node at the beginning 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;
    }
}

```

```

    }
}

/* Driver program to test above functions*/
int main()
{
    /* Start with the empty lists */
    struct node* a = NULL;
    struct node* b = NULL;
    struct node *intersect = NULL;

    /* Let us create the first sorted linked list to test the functions
       Created linked list will be 1->2->3->4->5->6 */
    push(&a, 6);
    push(&a, 5);
    push(&a, 4);
    push(&a, 3);
    push(&a, 2);
    push(&a, 1);

    /* Let us create the second sorted linked list
       Created linked list will be 2->4->6->8 */
    push(&b, 8);
    push(&b, 6);
    push(&b, 4);
    push(&b, 2);

    /* Find the intersection two linked lists */
    intersect = sortedIntersect(a, b);

    printf("\n Linked list containing common items of a & b \n ");
    printList(intersect);

    getchar();
}

```

Time Complexity: $O(m+n)$ where m and n are number of nodes in first and second linked lists respectively.

Method 2 (Using Local References)

This solution is structurally very similar to the above, but it avoids using a dummy node. Instead, it

Market research
that's fast and
accurate.

Get \$75 off

 Google consumer surveys

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

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

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

void push(struct node** head_ref, int new_data);

/* This solution uses the local reference */
struct node* sortedIntersect(struct node* a, struct node* b)
{
    struct node* result = NULL;
    struct node** lastPtrRef = &result;

    /* Advance comparing the first nodes in both lists.
       When one or the other list runs out, we're done. */
    while (a!=NULL && b!=NULL)
    {
        if (a->data == b->data)
        {
            /* found a node for the intersection */
            push(lastPtrRef, a->data);
            lastPtrRef = &((*lastPtrRef)->next);
            a = a->next;
            b = b->next;
        }
        else if (a->data < b->data)
            a=a->next;          /* advance the smaller list */
        else
            b=b->next;
    }
    return(result);
}

/* UTILITY FUNCTIONS */
/* Function to insert a node at the beginging of the linked list */
```

705



Subscribe

Recent Comments

Abhi You live US or India?

Google (Mountain View) interview · 49 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 · 1 hour ago

GOPI GOPINATH @admin Highlight this sentence "We can easily...

Count trailing zeroes in factorial of a number · 1 hour ago

newCoder3006 If the array contains negative numbers also. We...

Find subarray with given sum · 2 hours ago

AdChoices

► [Linked List](#)

► [Java Array](#)

► [Node](#)

AdChoices

► [Null Pointer](#)

```

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;
    }
}

/* Driver program to test above functions*/
int main()
{
    /* Start with the empty lists */
    struct node* a = NULL;
    struct node* b = NULL;
    struct node *intersect = NULL;


    /* Let us create the first sorted linked list to test the functions
    Created linked list will be 1->2->3->4->5->6 */
    push(&a, 6);
    push(&a, 5);
    push(&a, 4);
    push(&a, 3);
    push(&a, 2);
    push(&a, 1);

    /* Let us create the second sorted linked list
    Created linked list will be 2->4->6->8 */
    push(&b, 8);
    push(&b, 6);

```

► [Memory Array](#)

► [Core Temp](#)

AdChoices 

► [Java XML](#)

► [C++ Programming](#)

► [Core Java](#)

```

push(&b, 4);
push(&b, 2);

/* Find the intersection two linked lists */
intersect = sortedIntersect(a, b);

printf("\n Linked list containing common items of a & b \n ");
printList(intersect);

getchar();
}

```

Time Complexity: $O(m+n)$ where m and n are number of nodes in first and second linked lists respectively.

Method 3 (Recursive)

Below is the recursive implementation of sortedIntersect().

```

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

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

struct node *sortedIntersect(struct node *a, struct node *b)
{
    /* base case */
    if (a == NULL || b == NULL)
        return NULL;

    /* If both lists are non-empty */

    /* advance the smaller list and call recursively */
    if (a->data < b->data)
        return sortedIntersect(a->next, b);

    if (a->data > b->data)
        return sortedIntersect(a, b->next);
}

```

```

// Below lines are executed only when a->data == b->data
struct node *temp = (struct node *)malloc(sizeof(struct node));
temp->data = a->data;

/* advance both lists and call recursively */
temp->next = sortedIntersect(a->next, b->next);
return temp;
}

/* UTILITY FUNCTIONS */
/* 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 lists */
    struct node* a = NULL;
    struct node* b = NULL;
    struct node *intersect = NULL;

    /* Let us create the first sorted linked list to test the function.
    Created linked list will be 1->2->3->4->5->6 */

```

```

push(&a, 6);
push(&a, 5);
push(&a, 4);
push(&a, 3);
push(&a, 2);
push(&a, 1);

/* Let us create the second sorted linked list
   Created linked list will be 2->4->6->8 */
push(&b, 8);
push(&b, 6);
push(&b, 4);
push(&b, 2);

/* Find the intersection two linked lists */
intersect = sortedIntersect(a, b);

printf("\n Linked list containing common items of a & b \n ");
printList(intersect);

return 0;
}

```

Time Complexity: $O(m+n)$ where m and n are number of nodes in first and second linked lists respectively.

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

References:

cslibrary.stanford.edu/105/LinkedListProblems.pdf



Related Topics:

- 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



2



Tweet

0



0

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

33 Comments

GeeksforGeeks

Sort by Newest ▼





...on the intersection...



vignesh • 12 days ago

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

```
#include"lcreate.h"
```

```
void intersection(struct node *head1,struct node *head2,struct node **newlink
{
int first=1;
while(head1!=NULL && head2!=NULL)
{
if(head1->data==head2->data)
{
if(first)
{
push(newlink,head1->data);
first=0;
}
else
{
append(*newlink,head1->data);
```

[see more](#)

^ | v • Reply • Share ›



Deepthi Shree Bhat • 18 days ago

Shall we use Hashing?

Create a hash table for List1... compare each element of List2 with hash table

This will work with unsorted list also with O(n)

1 ^ | v • Reply • Share ›



Himanshu Dagar → Deepthi Shree Bhat • 17 days ago



Yeah exactly we can do like this way

^ | v • Reply • Share ›



Mohit • a month ago

In method 2

instead of

```
/* found a node for the intersection */
push(lastPtrRef, a->data);
lastPtrRef = &((*lastPtrRef)->next);
```

it should be

```
/* found a node for the intersection */
push(&lastPtrRef, a->data);
lastPtrRef = &((*lastPtrRef)->next);
```

What say ??

^ | v • Reply • Share ›



Himanshu Dagar • 3 months ago

another recursive way can be this also

<http://ideone.com/zce7ps>

^ | v • Reply • Share ›



Marsha Donna • 4 months ago

```
struct node* intersection_of_two_sorted_list(struct node *head1, struct node *I
{
    struct node *head3=NULL, *temp3=NULL;
    struct node *new_node=NULL;
    if(head1!=NULL&& head2!=NULL)
```

```

{ while(head1!=NULL&&head2!=NULL)
{
if(head1->data==head2->data)
{ new_node=(struct node*)malloc(sizeof(struct node));
new_node->data=head1->data;
new_node->link=NULL;
head1=head1->link;
head2=head2->link;
if(head3==NULL)
{head3=new_node;
temp3=new_node;
}
}
else

```

[see more](#)

^ | v • Reply • Share ›



nehamahajan • 8 months ago

I have another solution by using Maps. Its complexity is also $O(m+n)$

<http://mahajanneha.blogspot.co...>

^ | v • Reply • Share ›



Hanish Bansal • 11 months ago

In method 3 (recursive) , passing result as an argument is totally redundant.

Here is the simplified function :

```

struct node *sortedIntersect(struct node *a, struct node *b)
{
if(a == NULL || b == NULL)
return NULL;
if(a->data data)
return sortedIntersect(a->next, b);

```

```
return sortedIntersect(a, b->next);
else if(a->data == b->data)
{
struct node *temp = (struct node *)malloc(sizeof(struct node));
temp->data = a->data;
temp->next = sortedIntersect(a->next, b->next);
return temp;
}
}
```

^ | v • Reply • Share ›



Ronny → Hanish Bansal • 10 months ago

@Hanish Bansal

great observation.

@GeeksforGeeks I guess you should definitely look into this and add it
It's much easier and understandable.

^ | v • Reply • Share ›



GeeksforGeeks → Ronny • 10 months ago

@Hanish Bansal: Thanks for suggesting a simpler solution.

@Ronny: Thanks for bringing to this to notice. We have update

^ | v • Reply • Share ›



Ronny → GeeksforGeeks • 10 months ago

@GeeksforGeeks Another thing complexity of this algo
where n is the number of nodes in shorter list".

Since the algorithm is dependent on the values of the nodes
the link.

for example

LISTA = {1,2,3,4,5,6,7,8,9,10,11}and

LISTB [0, 10, 11, 12]

However LISTB is smaller in size but when above algorithm runs first.

So there is no co-relation between running time and size. It depends upon the values in the linked list.

Therefore any of the linked list can be exhausted first irrespective of its size.
Hence the complexity is $O(m+n)$.

^ | v · Reply · Share ›



GeeksforGeeks → Ronny · 10 months ago

@Ronny: Thanks for pointing this out. We have updated the code.

^ | v · Reply · Share ›



Prateek Sharma · a year ago

Recursive Solution with $O(n+m)$ time complexity.....

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
struct Node {
    int value;
    struct Node *next;
};
int insertAtEnd(struct Node *node,int val) {
    struct Node *temp;
    struct Node *ne;
    temp = node;
    while(temp->next!= NULL) {
        temp = temp->next;
    }
    ne = (struct Node *) malloc(sizeof(struct Node));
    ne->value = val;
```

[see more](#)

^ | v • Reply • Share ›



Hanish Bansal • a year ago

The time complexity should be $O(m+n)$ not $O(n)$ where n is length of smaller li
Consider the case :

List a : 1->2->3->4->5->6

List b : 11->12->13

Here, $m=6, n=3$

But List a will die out first. So the complexity does not depend on the smaller c

^ | v • Reply • Share ›



Devarshi → Hanish Bansal • a year ago

If the short list dies out then why would you continue?? just terminate tl

do correct me if i am wrong.

^ | v • Reply • Share ›



Hanish → Devarshi • a year ago

We will terminate after one of the lists die out.

By short here, I refer the length of the list not the values containe

So any of the lists, shorter or longer (in length) can die out first a

So, it can be $O(m)$ or $O(n)$ and

that is expressed as $O(m+n)$

^ | v • Reply • Share ›



varun jain → Hanish Bansal • a year ago

hanish bansal :;)...agreed!

^ | v • Reply • Share ›



lotus · a year ago

An edge case is missing in all the methods. IF the input is 2->2 for one list and >2. We can keep the last inserted number in a variable and add the next num to the last inserted number.

^ | v · Reply · Share ›



kartik → lotus · a year ago

If two linked list contain (2, 2) as common elements, then output should be with it.

^ | v · Reply · Share ›



lotus → kartik · a year ago

I got confused, thinking duplicates should be removed and result

^ | v · Reply · Share ›



Shan · a year ago

Can anyone explain how this algorithm is $O(N)$?. I figure in worst case it has to traverse the length of the smallest list and M is the length of the longest list. In worst case the smallest list is also the last element of the larger list. Am I missing something?

^ | v · Reply · Share ›



priso · a year ago

The recursive solution does not seem to work, as in the last else part you are returning the result. According to my understanding result should always point to the next node.

^ | v · Reply · Share ›



priso → priso · a year ago

The correct code would be :

```
struct node *sortedIntersect(struct node *a, struct node *b,
                             struct node *result)
```



```

{
    /* base case */
    if(a == NULL || b == NULL)
    {
        return NULL;
    }

    /* If both lists are non-empty */

    /* advance the smaller list and call recursively */
    if(a->data < b->data)
    {
        return sortedIntersect(a->next, b, result);
    }
}

```

[see more](#)

^ | v • Reply • Share ›



priso → priso • a year ago

@geeksforgeeks: I am very sorry for so many comments of mine. The trying to post. Can you please delete the repeated comments. Thanks

```

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

```

^ | v • Reply • Share ›



hmmm • 2 years ago

a simpler to understand recursive code.

```

struct node* sortedintersect(struct node *a, struct node *b)
{

```

```

if(a==NULL || b==NULL)
return NULL;
if(a->data==b->data)
{
    t=(struct node *)malloc(sizeof(struct node));
    t->data=a->data;
    t->next=sortedintersect(a->next,b->next);
    return t;
}
else if(a->data > b->data)
return sortedintersect(a,b->next);
else
return sortedintersect(a->next,b);
}

```

^ | v • Reply • Share ›



Avinash Srikantan • 2 years ago

- 1) The idea is to select a random list and insert its nodes in a hashmap.
- 2) Traverse the list and see if the node already exists in the hashmap.
- 3) If there is a match return the node.

Time complexity is $O(n) + O(m) = O(n+m)$

```

/* Paste your code here (You may delete these lines if not writing c)
struct node *FindIntersection(struct node *List1, struct node *List2)
{
    If (List1==NULL || List2==NULL) return NULL;
    struct node *current=List1;
    struct node *current2=List2;
    hash_map<struct node*,int> my_hasMap;
    hash_map<struct node*,int>::iterator it1;

```

```
pair<struct node*,int> myPair;  
while(current!=NULL)  
{  
    my_hashMap.insert(myPair<struct node*,int>(current,1)  
    current=current->next;  
}
```

[see more](#)

^ | v • Reply • Share ›



seeker7 • 3 years ago

For the recursive method ,there is a typo in code :
it should be:

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

^ | v • Reply • Share ›



foobar • 3 years ago

Another solution would be one pointer for each list and iterate them as if you are looking for an intersection, save it in the new list.

^ | v • Reply • Share ›



wittywoman • 3 years ago

what about unsorted linked list intersection? Does anyone have a O(N) solution?

^ | v • Reply • Share ›



Mike • 3 years ago

I am working on a code for linked lists and part of it is to have the intersection of two linked lists. My current code works fine, but I am trying to get the runtime down, since originally my intersection was quadratic time. I guess what my question is is how can I do it the above way if I return anything?

^ | v • Reply • Share ›



Sambasiva • 4 years ago

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

struct Node
{
    int data;
    struct Node *next;
    struct Node *arb;
};

typedef struct Node Node;
typedef Node* list;

Node *appendNode(list l, int elm);

list intersect(list l1, list l2)
{
```

[see more](#)

^ | v • Reply • Share ›



ajayreddy → Sambasiva • 3 years ago

```
list intersect(list l1, list l2)
{
    list l3 = NULL;
    list head = NULL;
    list tail = NULL;
    while(l1 && l2)
    {
        if(l1->data < l2->data)
```

```
        l1 = l1->next;
    }
    else if(l1->data > l2->data)
        l2 = l2->next;
    else
    {
        temp = (Node *)malloc(sizeof(Node));
        temp->data = l1->data;
        if(!head) {
            head = temp;
        }
    }
}
```

[see more](#)

^ | v • [Reply](#) • [Share](#) ›



veera reddy → Sambasiva • 3 years ago

This not good method , because for each insertion you have to travel to the new node . So it takes more time

^ | v • [Reply](#) • [Share](#) ›

[Subscribe](#)

[Add Disqus to your site](#)

@geeksforgeeks, **Some rights reserved**

Contact Us!

Powered by **WordPress** & **MooTools**, customized by geeksforgeeks team