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)
{
  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 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 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 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 */
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 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 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 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);

    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:

- 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.

Utkarsh Mishra • a month ago

```
node* intersection(node *head1,node *head2)
{

node *i=head1;
node *j=head2;
node *rs=(node*)malloc(sizeof(node));
rs->link=NULL;rs->data=0; //0 indicates start of list
node *k=rs;
while(i!=NULL && j!=NULL)
{
if(i->data==j->data)
{
```

```
node *temp=(node*)malloc(sizeof(node));
temp->link=NULL; //inserting in list
temp->data=i->data;
k->link=temp;
i=i->link; //i++
j=j->link; //j++
k=temp; //k++
}
else if(i->data<j->data)
i=i->link;
else
j=j->link;
}
return rs;
}
```


Umesh Lohani • a month ago

It can be done by using hashtable
Traverse a list and enter the addresses to the hashtable.
Now traverse second list and check containsValue in the hashtable. If it exists than that is the intersection node.

sid • 2 months ago

List* intersection(List list1, List list2)
//
List *list3 <- get new list
Node* nodePtr1 <- get start node of list1
Node* nodePtr2 <- get start node of list2

while nodePtr1 and nodePtr2 is not null
do
- if nodePtr1 ->data == nodePtr2 ->data
- then
- - list3.add(nodePtr1 ->data)
- - nodePtr1 <- nodePtr1->next
- - nodePtr2 <- nodePtr2->next
- else if nodePtr1->data < nodePtr2->data
- then
- - nodePtr1 <- nodePtr1->next
- else
- - nodePtr2 <- nodePtr2->next

return list3

• Reply • Share ›

surbhijain93 • 2 months ago

Please explain lastPtrRef = &((*lastPtrRef)->next); with the help of an example

• Reply • Share ›

Aditya Goel surbhijain93 • 15 days ago

see if this small code helps -

```
struct node* head = NULL;
        // Start out pointing to the head pointer
        struct node** lastPtrRef= &head;
        int i;
        for (i=1; i<6; i++) {
                // Add node at the last pointer in the list
                push(lastPtrRef, i);

                /* Advance to point to the new last pointer
                lastPtrRef always points to the last pointer
                in the list instead of to the last node */
                lastPtrRef= &((*lastPtrRef)->next);
        }
        // head == {1, 2, 3, 4, 5}; -- insertion in tail
```

• Reply • Share ›

surbhijain93 • 2 months ago

I have a doubt, result is defined inside the function sortedIntersect, so its scope remain inside that function...so, when we return result won't it's value be already turned garbage? i had read somewhere that if there is a function int sum{int c=2+4; return c} it will return garbage as c in scope of the function sum only

• Reply • Share ›

Aditya Goel surbhijain93 • 15 days ago

result is a pointer. It contains the address of a linked list node and function is just returning the address. That address will be valid outside sortedIntersect() function and will be assigned to intersect pointer variable in main().

• Reply • Share ›

ANKIT SINGH • 2 months ago

Please anybody explain the meaning of following statement briefly

lastPtrRef = &((*lastPtrRef)->next);

where lastPtrRef is a double pointer of the function sortedIntersect in the second method

where lastPtrRef is a double pointer of the function sortedIntersect in the second method.

1 • Reply • Share ›

surbhijain93 ANKIT SINGH • 2 months ago

I also didn't understand this...smeone pls explain..

• Reply • Share ›

Ankit Marothi • 2 months ago

This can be done through a stack as well as through a hash table.

• Reply • Share ›

ANKIT SINGH Ankit Marothi • 2 months ago

Can you be more precise and explain how you are going to do that?

• Reply • Share ›

Raviteja Somisetty • 3 months ago

Can't we use the merging logic from merge sort here? Except that we have to extract elements with condition as L[i] = R[j] where L and R are the given two linked lists?

• Reply • Share ›

ferrari21 • 3 months ago

see this code please....

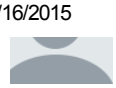
```
struct node *intersect(struct node *f1,struct node *f2)
{
struct node *ms1=f1;
struct node *ms2=f2;
struct node *sm=NULL,*a;
if((ms1==NULL)||(ms2==NULL))
{
printf("No intersection\n");
return;
}

while((ms1!=NULL)&&(ms2!=NULL))
{
if(ms1->data<ms2->data)
ms1=ms1->link;
else if(ms1->data>ms2->data)
ms2=ms2->link;
```

see more

• Reply • Share ›

Naval • 3 months ago

I make a code which would traverse a link list only once and I trace my code several time but



I make a code which would traverse a link list only once and I trace my code several time but could not find out error in my code can any one help me out to solve the problem

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

struct node
{
int data;
struct node* next;
};

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

{
struct node* new_node =
(struct node*) malloc(sizeof(struct node));
```

see more

• Reply • Share ›

ameya • 4 months ago

For finding the distinct of the intersection of the two list this can be used do reply if it has any error

It takes two list as an input and creates an new list.
It Traverses trough the list_1. within its while it compares data with list_2 if it finds a match then adds it to the new list and then compares the new list with list_1 to maintain a distinct list.

```
Node * intersection( Node *list_1, Node *list_2 )
{
Node * intersection_list = NULL;
Node **inter_ptr = &intersection_list;
Node *list_2_head = list_2;
Node *inter_head = NULL;

int exist = 0;

while( list_1 != NULL )
{
inter_head = intersection_list;
```

see more

• Reply • Share ›

zeus • 5 months ago

Implementation in Java:-

public void findIntersection(IntersectionOfTwoSortedList<integer> lL1,
IntersectionOfTwoSortedList<integer> lL2){

IntersectionOfTwoSortedList<integer> newLL = new IntersectionOfTwoSortedList<integer>();

Node temp1 = lL1.head;

Node temp2 = lL2.head;

while(temp1!=null && temp2!=null){

if((Integer) temp1.item == (Integer) temp2.item){

newLL.add((Integer) temp1.item);

temp1 = temp1.next;

temp2 = temp2.next;

---

see more

• Reply • Share ›

Neha Nigam • 6 months ago

i did using hashing ... is this one is efficient???

input -
no of elmnts in first -
no of elmnts in second -
first list
second list

input-
6
4
1 2 3 4 5 6
2 4 6 8

#include<iostream>

#include<stdlib.h>

#include<map>

---

see more

• Reply • Share ›

zeus Neha Nigam • 5 months ago

The idea is good but construction a hashtable will be a bit not so memory efficient.

• Reply • Share ›

Prateek Rathore Neha Nigam • 6 months ago

You wont be allowed to use map

• Reply • Share ›

Guest • 6 months ago

cant we use hashing here ??

• Reply • Share ›

sonu431 • 6 months ago

the second method fine working by using only struct node*result=NULL
thus adding a node at the begining(i.e. 6 4 2 would get printed .....)

```
while (a!=NULL && b!=NULL)
{
if (a->data == b->data)
{
/* found a node for the intersection */
push(&result, a->data);
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);
```

comment please is there any difference by coding in this way.....

• Reply • Share ›

guest sonu431 • 6 months ago

Question was to get result list as 2 4 6, and if you do this way, you need to reverse your list

• Reply • Share ›

sonu431 guest • 6 months ago

we also don't require to reverse the list. as we can insert the nodes at the end of list. i.e 2 4 6

• Reply • Share ›

guest ➚ sonu431 • 6 months ago

Yes I know, but what i said above is if you do in your way, you need to reverse it

• Reply • Share ›

guest ➚ guest • 6 months ago

Or write a function insert_tail and send result to it instead of push function, code of the function is as follows:

```
void insert_tail(struct node** head_ref, int new_data)
{
struct node* new_node = (struct node*) malloc(sizeof(struct node));
struct node *last = *head_ref;
new_node->data = new_data;
new_node->next = NULL;
if(*head_ref==NULL){
*head_ref=new_node;
return;
}
while (last->next != NULL)
last=last->next;
last->next=new_node;
}
```

• Reply • Share ›

ashu • 6 months ago

we can just traverse both lists and at each time checking smaller value node and incremnting it only until we get both list having equal nodes.

• Reply • Share ›

kajol • 7 months ago

if i will use hasp map and any element having count as 2 can b inserted in the new linked list. then what b will b the space complexity?

• Reply • Share ›

neil • 7 months ago

for the 1st method.

It takes O(m+n) time in the worst case, so what is the worst case?

The worst case occurs when the two lists are mutually exclusive and the greater data value alters in each iteration between the lists.

For example, we can talk about the even and odd lists of almost same length.

In this case, we need to traverse each of the lists alternately inside the while loop.

Parush • 7 months ago

Error ??
struct node

{

int data;

struct node *link;

};

void push(struct node** first,int item)

{

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

temp->data=item;

temp->link=(*first);

RK- An Unproven Theorem • 8 months ago

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

struct node* result;

while(a!=null && b!= null){

if(a->data < b->data){

a=a->next;

}
else if(a->data > b->data){

b=b->next;

}
else if(a->data==b->data){

```
push(result,a->data);

}

}

return result;

}
```

• Reply • Share ›

Gaurav Gupta • 8 months ago

the code doesn't work If the list is in non-decreasing order(which is still a sorted list).Example: 1->2->2->2->3->4->5 and 2->2->3->4->10->11 should return 2->2->3->4

• Reply • Share ›

Alok Patel • 8 months ago

How about this one ?

http://ideone.com/QN6cIb

• Reply • Share ›

Shadeslayer • 8 months ago

Why not hashing? We can have O(m+n) space complexity by hashing as well.

• Reply • Share ›

dhiru • 9 months ago

what is the time complexity?

• Reply • Share ›

Mittal • 9 months ago

Why do we want to create the list at its tail ?
Why can't the simple list creation work ? I mean the question doesn't ask to maintain the ascending order

• Reply • Share ›

Guest • 9 months ago

In Method 1, since we are stopping as soon as one node finishes, should not the time complexity be O(n) where n is the number of nodes in smaller list ?

• Reply • Share ›

Heracles • 9 months ago

Can we use this method :

Traverse the first list - add the elements to the hash table.

Traverse the second list - whenever we find that the elements of the second list are in the hash table - we create a node with that element and we create a list that way by connecting such nodes.

Will this work?

• Reply • Share ›

danny ➚ Heracles • 9 months ago

More efficient if List is Unsorted..

• Reply • Share ›

Mittal ➚ Heracles • 9 months ago

Yes, this should work.
But will have O(m+n) time complexity and O(n) auxillary space complexity too due to hash

• Reply • Share ›

anon • 10 months ago

here, with changing the original list

the intersected list is the list passed as 1st argument to function

```
struct A{

int a;

struct A *b;

}

void intersect(struct A **a, struct A **b)
{
struct A *c;

while (*a) {
while (*b && (*b)->a < (*a)->a)
b = &((*b)->b);
if(*b == 0)
```

see more

• Reply • Share ›

LePiaf • 10 months ago

Hi all,

you can also maintain circular linked list, and at the end make it regular linked list, returning first

node.

1 ^ | v • Reply • Share ›

vignesh • a year 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 • a year ago

Shall we use Hashing?
Create a hash table for List1... compare each element of List2 with hash table... If its present, add it to a new list...
This will work with unsorted list also with O(n)

4 ^ | v • Reply • Share ›

danny ➚ Deepthi Shree Bhat • 9 months ago

But it is more efficient if Lists are unsorted, in sorting we are adding an overhead of space complexity to the problem with this method..

1 ^ | v • Reply • Share ›

Himanshu Dagar ➚ Deepthi Shree Bhat • a year ago

Yeah exactly we can do like this way

^ | v • Reply • Share ›

Mohit • a year 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 ??

• Reply • Share ›

surbhijain93 ➦ Mohit • 2 months ago

No, because lastPtrRef is pointer to pointer .it contains address of result

• Reply • Share ›

Himanshu Dagar • a year ago

another recursive way can be this also

http://ideone.com/zce7ps

• Reply • Share ›

Shikha Gupta • a year ago

```
struct node* intersection_of_two_sorted_list(struct node *head1,struct node *head2)
{
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

• Reply • Share ›

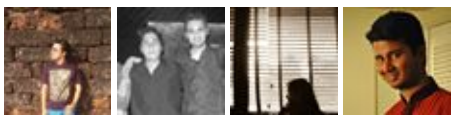
nehamahajan • 2 years ago

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

http://mahajanneha.blogspot.co...

• 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

- Bharath Kumar Reddy Janumpally

  findsubsets(int [] arr) { int numofsubsets =...

  Power Set · 1 minute ago

- Siya

  We can do it in O(nlogn) i guess by using...

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

- darkprotocol

  Thanks god u didnt selected for SE role. It...

  Amazon Interview | Set 124 (On-Campus) · 22 minutes ago

- Humble_Learner

  When the range represented by the node (...

  Segment Tree | Set 2 (Range Minimum Query) · 34 minutes ago

- Humble_Learner

  Learned Something New! :)

Segment Tree | Set 1 (Sum of given range) · 43 minutes ago

- tiger

  i think efficient solution for the problem in...

  Myntra Interview Experience | Set 4 (For Senior Software Engineer) · 44 minutes ago