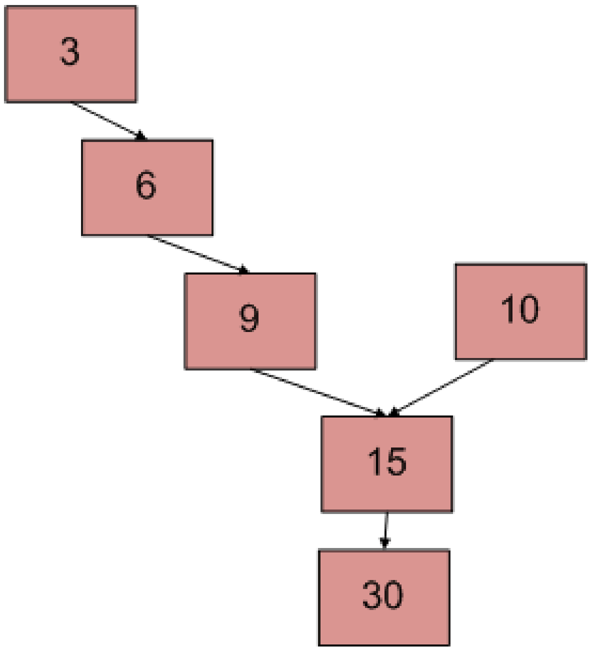
Write a function to get the intersection point of two Linked Lists.

There are two singly linked lists in a system. By some programming error the end node of one of the linked list got linked into the second list, forming a inverted Y shaped list. Write a program to get the point where two linked list merge.



Above diagram shows an example with two linked list having 15 as intersection point.

Method 1(Simply use two loops)
Use 2 nested for loops. Outer loop will be for each node of the 1st list and inner loop will be for 2nd list. In the inner loop, check if any of nodes of 2nd list is same as the current node of first linked list. Time complexity of this method will be O(mn) where m and n are the number of nodes in two lists.

Method 2 (Mark Visited Nodes)
This solution requires modifications to basic linked list data structure. Have a visited flag with each node. Traverse the first linked list and keep marking visited nodes. Now traverse second linked list, If you see a visited node again then there is an intersection point, return the intersecting node. This solution works in

O(m+n) but requires additional information with each node. A variation of this solution that doesn't require modification to basic data structure can be implemented using hash. Traverse the first linked list and store the addresses of visited nodes in a hash. Now traverse the second linked list and if you see an address that already exists in hash then return the intersecting node.

Method 3(Using difference of node counts)

1) Get count of the nodes in first list, let count be c1.

2) Get count of the nodes in second list, let count be c2.

3) Get the difference of counts d = abs(c1 – c2)

4) Now traverse the bigger list from the first node till d nodes so that from here onwards both the lists have equal no of nodes.

5) Then we can traverse both the lists in parallel till we come across a common node. (Note that getting a common node is done by comparing the address of the nodes)

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

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

/* Function to get the counts of node in a linked list */
int getCount(struct node* head);

/* function to get the intersection point of two linked
   lists head1 and head2 where head1 has d more nodes than
   head2 */
int _getIntesectionNode(int d, struct node* head1, struct node* head2);

/* function to get the intersection point of two linked
   lists head1 and head2 */
int getIntesectionNode(struct node* head1, struct node* head2)
{
  int c1 = getCount(head1);
  int c2 = getCount(head2);
  int d;

  if(c1 > c2)
  {
    d = c1 - c2;
    return _getIntesectionNode(d, head1, head2);
  }
  else
  {
    d = c2 - c1;
    return _getIntesectionNode(d, head2, head1);
  }
}

/* function to get the intersection point of two linked
   lists head1 and head2 where head1 has d more nodes than
   head2 */
int _getIntesectionNode(int d, struct node* head1, struct node* head2)
{
  int i;
  struct node* current1 = head1;
  struct node* current2 = head2;

  for(i = 0; i < d; i++)
  {
    if(current1 == NULL)
    {  return -1; }
    current1 = current1->next;
  }

  while(current1 !=  NULL && current2 != NULL)
  {
    if(current1 == current2)
      return current1->data;
    current1= current1->next;
    current2= current2->next;
  }

  return -1;
}

/* Takes head pointer of the linked list and
   returns the count of nodes in the list */
int getCount(struct node* head)
{
  struct node* current = head;
  int count = 0;

  while (current != NULL)
  {
    count++;
    current = current->next;
  }

  return count;
}

/* IGNORE THE BELOW LINES OF CODE. THESE LINES
   ARE JUST TO QUICKLY TEST THE ABOVE FUNCTION */
int main()
{
  /*
    Create two linked lists

    1st 3->6->9->15->30
    2nd 10->15->30

    15 is the intersection point
  */

  struct node* newNode;
  struct node* head1 =
            (struct node*) malloc(sizeof(struct node));
  head1->data  = 10;

  struct node* head2 =
            (struct node*) malloc(sizeof(struct node));
  head2->data  = 3;

  newNode = (struct node*) malloc (sizeof(struct node));
  newNode->data = 6;
  head2->next = newNode;

  newNode = (struct node*) malloc (sizeof(struct node));
  newNode->data = 9;
  head2->next->next = newNode;
  newNode = (struct node*) malloc (sizeof(struct node));
  newNode->data = 15;
  head1->next = newNode;
  head2->next->next->next  = newNode;

  newNode = (struct node*) malloc (sizeof(struct node));
  newNode->data = 30;
  head1->next->next= newNode;

  head1->next->next->next = NULL;

  printf("\n The node of intersection is %d \n",
          getIntesectionNode(head1, head2));

  getchar();
}
```

Run on IDE

Time Complexity: O(m+n)

Auxiliary Space: O(1)

Method 4(Make circle in first list)

Thanks to Saravanan Man for providing below solution.

1. Traverse the first linked list(count the elements) and make a circular linked list. (Remember last node so that we can break the circle later on).
2. Now view the problem as find the loop in the second linked list. So the problem is solved.
3. Since we already know the length of the loop(size of first linked list) we can traverse those many number of nodes in second list, and then start another pointer from the beginning of second list. we have to traverse until they are equal, and that is the required intersection point.
4. remove the circle from the linked list.

Time Complexity: O(m+n)

Auxiliary Space: O(1)

Method 5 (Reverse the first list and make equations)

Thanks to Saravanan Mani for providing this method.

```
1) Let X be the length of the first linked list until intersection point.
   Let Y be the length of the second linked list until the intersection point.
   Let Z be the length of the linked list from intersection point to End of
   the linked list including the intersection node.
   We Have
           X + Z = C1;
           Y + Z = C2;
2) Reverse first linked list.
3) Traverse Second linked list. Let C3 be the length of second list - 1.
     Now we have
        X + Y = C3
     We have 3 linear equations. By solving them, we get
```

```
        X = (C1 + C3 – C2)/2;
        Y = (C2 + C3 – C1)/2;
        Z = (C1 + C2 – C3)/2;
       WE GOT THE INTERSECTION POINT.
 4)  Reverse first linked list.
```

Advantage: No Comparison of pointers.
Disadvantage : Modifying linked list(Reversing list).

Time complexity: O(m+n)
Auxiliary Space: O(1)

Method 6 (Traverse both lists and compare addresses of last nodes) This method is only to detect if there is an intersection point or not. (Thanks to NeoTheSaviour for suggesting this)

```
1) Traverse the list 1, store the last node address
2) Traverse the list 2, store the last node address.
3) If nodes stored in 1 and 2 are same then they are intersecting.
```

Time complexity of this method is O(m+n) and used Auxiliary space is O(1)

Method 7 (Use Hashing)
Basically we need to find common node of two linked lists. So we hash all nodes of first list and then check second list.
1) Create an empty hash table such that node address is used as key and a binary value present/absent is used as value.
2) Traverse the first linked list and insert all nodes' addresses in hash table.
3) Traverse the second list. For every node check if it is present in hash table. If we find a node in hash table, return the node.

Please write comments if you find any bug in the above algorithm or a better way to solve the same problem.

251 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)

2.2 Average Difficulty : **2.2/5.0**
Based on **12** vote(s)

☐ Add to TODO List
☐ Mark as DONE

Like Share 36 people like this.

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

251 Comments GeeksforGeeks 1 Login

Recommend 7 Share Sort by Newest

Join the discussion…

Kalyani • 3 days ago

We can first check if intersection exists by comparing tails. Now we can traverse the first list and point each node to NULL. Then we traverse the second list till the node which points to NULL. That node is intersecting one.

• Reply • Share ›

Lokesh • 17 days ago

Method 3 doesn't work if both the lists are of equal length ?
How to accommodate this scenario ?

• Reply • Share ›

Pranav Kumar Jha → Lokesh • 15 days ago

It does work even in case of lists of equal length.

• Reply • Share ›

Atul Soni • a month ago

Best Solution would be
1) traverse the 1st linked list till end and join the last node to first node. ie make it circular.
2) Now traverse the 2nd linked list and match each node with the first node of first linked list, if it matches then both linked list are intersecting. If both linked list doesnt intersect then, 2nd linked end up getting Null node ie last one.

• Reply • Share ›

Christopher • 2 months ago

If the tail nodes of the two lists are equal, there must have intersected somewhere. If the tail nodes are stored in the list, the complexity is O(1).

• Reply • Share ›

Rahul → Christopher • 2 months ago

In normal list which has only single pointer for next item, to reach the tail node you have to traverse complete list, which will take O(m+n) for both list.

• Reply • Share ›

prashant maheshwari • 2 months ago

What will be the time complexity for the solution where "hash maps" are used?? wouldn't it be worse than O(m+n)?

• Reply • Share ›

Jenil Jain • 2 months ago

We can reverse both the linked list and then traverse through the reversed list to check where they diverge. That will be the intersection.

int getIntersection(struct node* list1, struct node* list2)
{
reverse(list1);
reverse(list2); // using recursive method with O(n) time and O(1) space complexity

while(list1 && list2){
if(list1->data == list2->data){
list1 = list1->next;
list2 = list2->next;
}
else
return list1->data;
}
return -1;
}

• Reply • Share ›

Himani Kashyap Jenil Jain • 2 months ago

If lists are:
1-2-3-4-10-11
5-6-7-10-11
It means that the lists intersect at 10.
On reversing list1 it will become:
11-10-4-3-2-1
and list2 will become:
5-6-7-10-4-3-2-1
because when you reversed the list1, list2 also got altered because both were sharing same nodes after node 10.
So this will not work.

1 • Reply • Share ›

Jenil Jain Himani Kashyap • 2 months ago

Yeah right!!

• Reply • Share ›

dev • 4 months ago

can anyone explain how to compare the address of two nodes ?

• Reply • Share ›

Manish Tiwari • 4 months ago

how will we find the intersection point using stack?

• Reply • Share ›

satyadev yalavarthy Manish Tiwari • 3 months ago

1. Push 1st list to stack1
2. push 2nd list to stack2
3. pop out each element from both stacks and compare until elements are not same.

In step 3 we are traversing lists from reverse until intersection.

1 • Reply • Share ›

GFN • 4 months ago

what if both linked list size are equal i.e. c1 = c2 ?

• Reply • Share ›

Teena GFN • 4 months ago

Then start from current position.

• Reply • Share ›

yashwanth reddy • 5 months ago

this can be solved even with stack

• Reply • Share ›

Leena Singh • 5 months ago

i am writing this code in java. it is not comparing the addresses,it gives -1. when i compare data then only it returns the intersection point. please check out my problem.

• Reply • Share ›

Abhinav Singh Leena Singh • 4 months ago

Don't use compareTo() for comparing addresses. Use ==.

• Reply • Share ›

Kishore Leena Singh • 4 months ago

Which method? I mean method #?

• Reply • Share ›

kartik mittal • 5 months ago

if we traverse both lists and for each node , we XOR the addresses , then first 0 result from XOR will give us the intersection point, isnt it fine ?

• Reply • Share ›

Rajan Kalra kartik mittal • 4 months ago

If I have understood your method right then time complexity would be O(n^2) for comparing all nodes of list A with each node of list B.

• Reply • Share ›

Kishore kartik mittal • 4 months ago

But how do you access the address?

• Reply • Share ›

Abhisek Mandal • 6 months ago

Using the node count difference of the two linked lists...

```
int FindMergeNode(Node *headA, Node *headB)
{

    int An = countNodes(headA);

    int Bn = countNodes(headB);

    Node *it=nullptr,*itA=nullptr,*itB=nullptr;

    int ctr=0;

    if(An>Bn){

        for(it=headA; ctr<(An-Bn); ctr++, it=it->next);

        headA = it;
```

see more

• Reply • Share ›

Mrinal Kr Das Abhisek Mandal • 5 months ago

is it ok..abhisek

• Reply • Share ›

Prince Bharti • 6 months ago

java version(using method 3) https://ideone.com/v0mEhU

• Reply • Share ›

Anirudh Gupta • 6 months ago

Traverse both lists and store the addresses in stacks.
Then compare the addresses and pop stacks. The intersection point is the next node of first unmatached address.

1 • Reply • Share ›

Rohit Saluja Anirudh Gupta • 5 months ago

Well this method is almost similar to the first method and is not a very good solution as it increases the space complexity plus also has same time complexity as of Method 1

^ | ˅ • Reply • Share ›

Sagar ➦ Anirudh Gupta • 6 months ago

It increases space complexity. But kudos for the new method.

^ | ˅ • Reply • Share ›

Nishant Jain ➦ Sagar • 5 months ago

@Sagar @Anirudh Gupta
Instead of using the stack, you can reverse both the linked list and compare the reversed linked list to find out the difference.

^ | ˅ • Reply • Share ›

manish • 6 months ago

want solution for the below:

Write a function to which takes head nodes of 3 Linked Lists and returns head node of another 4th Linked List, where each node contains the value which is sum of 3 Lists.

for example

H1: 2->7>3

H2: 4->5->9->5

H3: 3->1->2

Output : 5->1->8->0 (as is derived as 273+4595+312=5180)

^ | ˅ • Reply • Share ›

Saurabh Singh ➦ manish • 5 months ago

another method could be store each data item in stack . . .pop itout and multiply with a factor of 10^n . . . and rest same as below

^ | ˅ • Reply • Share ›

Saurabh Singh ➦ manish • 5 months ago

Reverse the linked list H1 : 3>7>2
Start reading the data item fro each node starting from head
For each node multiply it with a factor of 10^n where n is the no of node(like for 7 its 2 and 3 for node data 3)
keep adding and get the result

^ | ˅ • Reply • Share ›

Abhinav Bahiya → manish • 5 months ago

use arrays with linked lists first copy the contents of 1st linked list to an array element like
2*100+7*10+3*1 (Hundred's +Ten's +one's)

same to be done with 2nd and 3rd linked list .

then add the array elements (which u will get after performing the above task)

then in the sum variable
you can use modulas and division operation to get the individual no. then take a linked list and enter that in the data part simutaneously.

By:
Abhinav Bahiya
Chitkara University, Chandigarh
abhinavbahiya@gmail.com

• Reply • Share ›

coolk • 6 months ago

Slight optimization of method 3, for usecase where lengths are equal.

http://ideone.com/j9DAjZ

• Reply • Share ›

Manishi • 7 months ago

In method 2 there is no need of traversing the list 1,while creating a structure we will mark all the flags of list1 as 1.Thus we need to traverse the list 2 only to get to know whether the node's flag in list 2 is already set.
this will reduce to the complexity from o(m+n) to o(n) or o(m).
Please correct if I am wrong.

• Reply • Share ›

Gautham Kumaran • 7 months ago

java code for method 4

https://github.com/gautham20/g...

• Reply • Share ›

Gautham Kumaran • 7 months ago

can someone provide a better or detailed explanation for method 5?

1 • Reply • Share ›

pravir ahuja • 7 months ago

MAY BE THIS CODE IS FLAWED FOR THE CASE WHEN THERE IS NO

INTERSECTION (correct me if i'm wrong)
in the code of method 3 what will be o/p if the linked lists dont have any intersections, according to the above code -1 should be returned by the function and the o/p should be==
the node of the intersection is -1
can someone tell why -1 is not getting returned and why am i not getting the above o/p of -1
the program is halting and not ending for such a case i.e., the code after after the printf statement in main is not getting executed for such a case

• Reply • Share ›

manu agrawal • 7 months ago

another implementation using two stacks:

https://ideone.com/9KKqKI

2 • Reply • Share ›

SlickHackz • 7 months ago

In method 5, how to find the length of the first linked list until intersection point ?

• Reply • Share ›

Hitesh Saini • 7 months ago

we can also use the hashing
A HASH function which will store the address of each node and if there is a collision of addresses then this is the node of intersection
Time Complexity will be O(k)
where k=max(m,n);
"m" and "n" are the lengths of two given linked list

Here is my getintersectionfunction

```
int getIntesectionNode(struct node* head1, struct node* head2)
{
struct node *temp1=head1,*temp2=head2,*ans=NULL;
while(temp1!=NULL||temp2!=NULL)
{
if(temp1!=NULL)
{
if(hashmap[temp1]==0)
{
hashmap[temp1]=1;
```

see more

1 • Reply • Share ›

coderr ➦ Hitesh Saini • 5 months ago

Your link seems to be broken. Can you re post it maybe?
Also, I'm not familiar with hash maps. I want to implement it. Can you suggest some good sources to study from?

• Reply • Share ›

Hitesh Saini ➦ coderr • 5 months ago

http://ideone.com/dDRyf0

watch tutorials on youtube.
google your queries you will get all your answers..:)

• Reply • Share ›

coderr ➦ Hitesh Saini • 5 months ago

Thanks a lot for re posting the code.
And yeah i will do that. Thanks :)

1 • Reply • Share ›

Jitin Maherchandani • 7 months ago

ANOTHER METHOD(EASIER TO UNDERSTAND):-
We can traverse both the list till the end is reached and then traverse back together, compare the values of nodes while we are traversing one by one and when we find different node value we can predict the intersection value of the two lists.

• Reply • Share ›

Anand Barnwal ➦ Jitin Maherchandani • 7 months ago

So how do you traverse back ?
If you use two stacks to store the pointers of both list nodes and then while popping and comparing it, will increase the space complexity
(i.e.O(m + n))

1 • Reply • Share ›

Ayush ➦ Jitin Maherchandani • 7 months ago

we cannot traverse back without using double pointers in a list

• Reply • Share ›

ss ➦ Ayush • 7 months ago

we can do back traversing using stack

1 • Reply • Share ›

Saurabh Kumar ➦ Ayush • 7 months ago

use stack to store the address and thereafter match the stack top of two list... difference in stack address point to the intersecting point

• Reply • Share ›

Ayush • 7 months ago

In the given example can anyone explain what will be the value of X, Y &Z according to method 5
ll1= 1->2->3->4->5->6
ll2= 8->4->5->6
i.e intersection point is 4

• Reply • Share ›

Load more comments