GeeksforGeeks

A computer science portal for geeks

GeeksQuiz

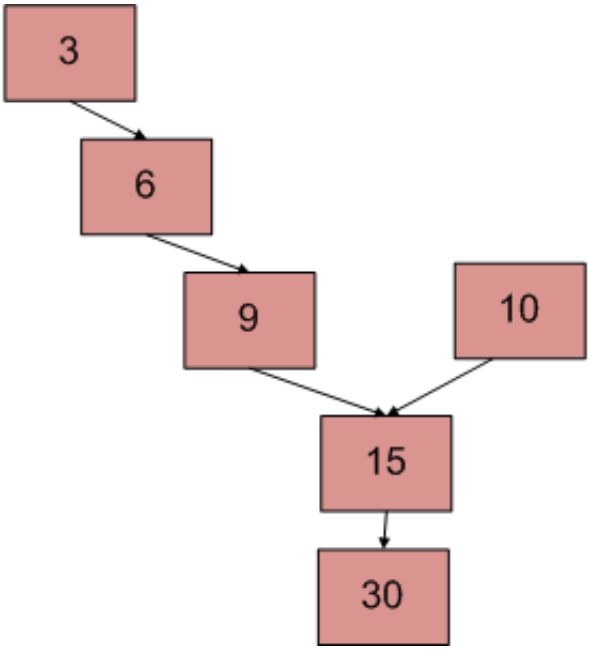
Login/Register

- Home
- Algorithms
- DS
- GATE
- Interview Corner
- Q&A
- C
- C++
- Java
- Books
- Contribute
- Ask a Q
- About

Array
Bit Magic
C/C++
Articles
GFacts
Linked List
MCQ
Misc
Output
String
Tree
Graph

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();
}
```

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)

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

when executing with another list of numbers it's showing intersection node as 15 itself !!!!!

• Reply • Share ›

Madan • a month ago

"Traverse the first linked list and store the addresses of visited nodes in a hash." In method 2, what does this mean? How would we do this in Java?

• Reply • Share ›

siddharth • a month ago

Very good explanation.

• Reply • Share ›

Rushikesh Deshpande • a month ago

Why method 6 cannot be used to actually find the intersection point?

• Reply • Share ›

Stack ➚ Rushikesh Deshpande • 18 days ago

We can use method 6 only to check if the address of the last nodes of both the list matches or not. If it matches there is an intersection but we cannot know for sure where is the intersection point, is it the last node or much before it. In the given example above address of node 30 will be checked it matches in both the list so there is an intersection but we see that the intersection point is actually node 15.

• Reply • Share ›

Hinata Hyuga • 2 months ago

Method 2 will useful even if the common part of the lists contains a circle.

• Reply • Share ›

Ashish P Magar • 2 months ago

Method 4: In the case of loop the pointers always meet before the intersection(try any example) so no need for extra logic. The intersection point will always be next pointer to the node where 2 pointers meet from finding the loop logic

• Reply • Share ›

harish rajagopal • 3 months ago

How about you use an additional stack for each LinkedList<node>, traverse through them and push the Node's into respective stacks O(n) + O(m).

Next you have a while loop where you iterate until you reach the base of either stack, in the process you will find a match at some point if an intersection did exist, else you exhaust on or both stacks and you break out of the loop.

The logic behind this approach is that it is easier to find the intersection point from the end of the LinkedList rather than the front and since it is a singly LinkedList we use a stack to do exactly

LinkedList rather than the front and since it is a singly LinkedList we use a stack to do exactly that.

9 ^ | v • Reply • Share ›

Gaurav Tomar • 3 months ago

If there do not merge, Method 5 will fail.

^ | v • Reply • Share ›

Rajesh Kamalanathan • 5 months ago

store the address of a linked list in a array and traverse the next list by comparing the address of the list.. O(n^2).. :P best i hope

^ | v • Reply • Share ›

Anonymous ➚ Rajesh Kamalanathan • 4 months ago

traverse both linked list and push addresses into different stacks and then pop address until they are equal. And you'll get Intersection point from that.

2 ^ | v • Reply • Share ›

Rajesh Kamalanathan ➚ Anonymous • 4 months ago

gud idea....

^ | v • Reply • Share ›

random_dude • 5 months ago

Or we can store all the nodes of the first list in a hash and then traverse the second list till the key matches..

3 ^ | v • Reply • Share ›

prashant patel ➚ random_dude • 18 days ago

we will be traversing first linked list Plus we will be creating a tree(hash map is a tree), then we will be traversing second linked list Plus we will be performing search operation for each node of second linked list in tree(hash map).
Doesn't it seems to be less efficient ?

^ | v • Reply • Share ›

prabhas ➚ random_dude • 4 months ago

Excellent

^ | v • Reply • Share ›

kaushik Lele • 5 months ago

I could not understand logic of "Method 3(Using difference of node counts)"
IF there are two lists, both having 10 node .... then the difference in length is zero. But there can be different combinations possible like only 1 node common in both lists, 2 nodes common, 3 nodes common .... 10 nodes common.

So what advantage does difference calculation give ?

• Reply • Share ›

kaushik Lele ➚ kaushik Lele • 5 months ago

May be I am thinking of worst case scenario. I think it will be advantageous in non-worst scenarios.

e.g. one list is of length 10 and other is of 4.
So at the max last 4 nodes can have common.

So we can skip first 6 nodes of bigger list and just compare last 4 nodes in parallel.

So we skip first 6 (which is 10-4 i.e difference)

• Reply • Share ›

neeraj kumar ➚ kaushik Lele • 4 months ago

You are thinking exact the same way what method 3 says.

• Reply • Share ›

hassan ashraf • 5 months ago

Just go to the end of both list. If the address of last nodes of both list are same they have a intersection somewhere. But this cant be used to find the point of intersection. For finding the point of intersection traverse both the list putting addresses in separate stacks until all nodes addresses has been pushed. Then keep popping addresses from both and at the point where they are not equal,the next is the point is the point of intersection.

2 • Reply • Share ›

Aaditya Sriram • 6 months ago

Use two stacks, push nodes till end for both, pop out one by one and the last same node is point of intersection.

5 • Reply • Share ›

fresh_man ➚ Aaditya Sriram • 6 months ago

Yes, it can be done like you said,but would be Auxiliary Space: O(m+n)

So this is not so desirable when it can be done in constant memory space O(1)

• Reply • Share ›

Anon ➚ Aaditya Sriram • 6 months ago

What if the lists doesn't intersect ?

• Reply • Share ›

Junchao Gu ➚ Anon • 6 months ago

I think **@Aaditya Sriram**'s way is ok if nodes contain different values if they are not the same nodes--because compare value only does not mean the same

not the same nodes--because compare value only does not mean the same node

• Reply • Share ›

Umang jain • 6 months ago

/*Make an interating pointer that goes forward every time till the end, and then jumps to the beginning of the opposite list, and so on. Create two of these, pointing to two heads. Advance each of the pointers by 1 every time, until they meet.*/

```
int FindMergeNode(Node *headA, Node *headB)
{
Node *p1 = headA;
Node *p2 = headB;
while(p1 != p2){
p1 = p1->next;
p2 = p2->next;
if(p1 == NULL)
p1 = headB;
if(p2 == NULL)
p2 = headA;
}
return p1->data;
}
```

• Reply • Share ›

Ankit Dave • 6 months ago

we can use two stacks and then keep popping from both the stacks till we get similar elements. last common element is the answer

2 • Reply • Share ›

Anil Kumar K K • 6 months ago

Not sure how does Method 4 solve the problem ? Could you please elaborate it with code ?

• Reply • Share ›

The_Geek ➚ Anil Kumar K K • 6 months ago

I assume that u have understood method 3, so Method 4 is the another version of method 3 approach. What we have to do is make the larger list circular, and compute its length, Then traverse smaller list for length no. of nodes. For better understanding, consider this example. we have 4 nodes in 1st(larger) list having data, 1,2,3,4. Second list have 3 nodes, having data 7,3,4, i.e. intersection point is node having data value as 3. So now length=4 as per above discussion. So we will traverse 2nd list for 4 nodes. this way our pointer will reach on node having data value=2. Now if u have noticed both lists have pointers just 1 node away from intersecting node. So we will start another pointer from 2nd, i.e. smaller list, and compare parallel, this way we will get node having data=3 as intersecting node.

Hope u got it !

• Reply • Share ›

Gaurav Tomar ➚ The_Geek • 3 months ago

What if either of the list already has a cycle int it.

• Reply • Share ›

reeetesh11 • 7 months ago

we can reverse both lists and traverse both the lists together until they have different node ...

1 • Reply • Share ›

Gaurav Tomar ➚ reeetesh11 • 3 months ago

nice idea

• Reply • Share ›

Guest • 7 months ago

we can use Recursion too here
here " one " and " two " are both linked list's first pointer
and the pointer " answer " represents the Intersection point .

```
linked* intersection(linked *one,linked *two)
{
if(two==one) { return NULL; }
if(one->next!=NULL) { one=one->next; }
if(two->next!=NULL) { two=two->next; }

answer=intersection(one,two);

if(two==one) { answer=two; }
return answer;
}
```

• Reply • Share ›

Umang Jain • 7 months ago

/*Make an interating pointer that goes forward every time till the end, and then jumps to the beginning of the opposite list, and so on. Create two of these, pointing to two heads. Advance each of the pointers by 1 every time, until they meet.*/

```
int FindMergeNode(Node *headA, Node *headB)
{
Node *p1 = headA;
Node *p2 = headB;
while(p1->data != p2->data){
p1 = p1->next;
p2 = p2->next;
```

```
if(p1 == NULL)
p1 = headB;
if(p2 == NULL)
p2 = headA;
}
return p1->data;
}

// This solves in O(n) time with 0(1) space assuming n > m
```

• Reply • Share ›

sivanraj ➚ Umang Jain • 2 months ago

What happens if all the nodes have the same value?

• Reply • Share ›

codejam123 ➚ Umang Jain • 6 months ago

exactly bro!
i was thinking the same and couldnot find why this post wasnt made

• Reply • Share ›

Junchao Gu ➚ Umang Jain • 6 months ago

are you sure this is O(n). and btw, you cannot say two nodes are equal if data equal

• Reply • Share ›

parit13 • 7 months ago

using map or hashing to get O(n) time but O(n) space.
method-
1) insert all the node(key,address) of list first in map.
2) for second list check if the address already present in map or not. if not
insert node of second list in map.
if address already present then print the corresponding key associated with that address. that will be the merge point of that list.
thanks

• Reply • Share ›

DS+Algo=Placement • 8 months ago

@GeeksforGeeks

Does method 5 even guarantee that there is an intersection point. We need to apply method 6 to check if there is intersection point and if there is, then method 5 should be used.

• Reply • Share ›

Abhishek Kumar Singh ➚ DS+Algo=Placement • 7 months ago

In that case u wil get z=0;
fine isnt it?

• Reply • Share ›

Guest • 8 months ago

@GeeksforGeeks the above methods will give an output of 15 for the following input :
3->6->9->15->30
10->15->25->12
instead of just checking the value of current data,a while loop should check if all values are the same after the first common value

• Reply • Share ›

Arjun Yadiki ➚ Guest • 7 months ago

Did you miss the point that we were traversing a linked list and comparing the addresses (pointers). If a node is common between two linked lists, then everything that follows will be present in both the singly linked lists.

• Reply • Share ›

Abhishek Kumar Singh ➚ Guest • 7 months ago

Check the addreses of every node onstead.

• Reply • Share ›

DS+Algo=Placement • 8 months ago

method 6 doesn't tell about intersection point.

2 • Reply • Share ›

samthebest • 8 months ago

method 4 logic ? anyone regarding maintaining two pinters on the 2nd list with difference as that of 1st list?

• Reply • Share ›

The_Geek ➚ samthebest • 6 months ago

I assume that u have understood method 3, so Method 4 is the another version of method 3 approach. What we have to do is make the larger list circular, and compute its length, Then traverse smaller list for length no. of nodes. For better understanding, consider this example. we have 4 nodes in 1st(larger) list having data, 1,2,3,4. Second list have 3 nodes, having data 7,3,4, i.e. intersection point is node having data value as 3. So now length=4 as per above discussion. So we will traverse 2nd list for 4 nodes. this way our pointer will reach on node having data value=2. Now if u have noticed both lists have pointers just 1 node away from intersecting node. So we will start another pointer from 2nd, i.e. smaller list, and compare parallel, this way we will get node having data=3 as intersecting node.
Hope u got it !!

• Reply • Share ›

vamsi • 8 months ago

is this works for two linked lists having only one common node
two linked lists intersecting at only one node

1->2->3->4->5->6->7->8->9
11->12->13->14->5->16->17

Reply • Share ›

vaibhavatul47 vamsi • 8 months ago

same node cannot have two different next pointers.
in your case, next of 5 will either be 6 or 16.

2 Reply • Share ›

Ashrith • 8 months ago

@geeksforgeeks http://ideone.com/VzHxwP

Push nodes into 2 stacks.
Pop all common nodes.
Last common node is point of intersection

Source :DSA made easy by Narasimha Karumanchi

5 Reply • Share ›

vipinkaushal • 9 months ago

method 5 will not work if given linked lists are disjoint

Reply • Share ›

Load more comments

- 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

- Goku

  They are considering 0 based indexing instead...

  Write a function to get Nth node in a Linked List · 38 minutes ago

- lebron

  since the array size is 5, it takes constant...

  K’th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case Linear Time) · 4 hours ago

- lebron

  merge sort

  K’th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case Linear Time) · 4 hours ago

- Shubham Sharma

  You saved my time :)

  Searching for Patterns | Set 2 (KMP Algorithm) · 5 hours ago

- Prakhar

  Why so many LOCs, if I'm not wrong (please...

  Largest Sum Contiguous Subarray · 5 hours ago

- Aayush Gupta

  For R4 Q3, Another solution would be to use a...

  Amazon Interview Experience | Set 168 · 6 hours ago