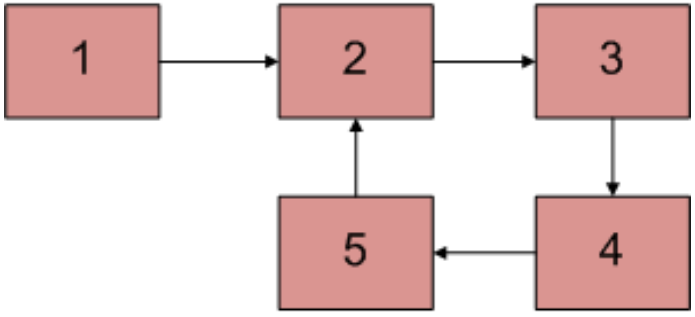
# Detect and Remove Loop in a Linked List

Write a function *detectAndRemoveLoop()* that checks whether a given Linked List contains loop and if loop is present then removes the loop and returns true. And if the list doesn't contain loop then returns false. Below diagram shows a linked list with a loop. *detectAndRemoveLoop()* must change the below list to 1->2->3->4->5->NULL.




We recommend to read following post as a prerequisite.

Write a C function to detect loop in a linked list

Before trying to remove the loop, we must detect it. Techniques discussed in the above post can be used to detect loop. To remove loop, all we need to do is to get pointer to the last node of the loop. For example, node with value 5 in the above diagram. Once we have pointer to the last node, we can make the next of this node as NULL and loop is gone.
We can easily use Hashing or Visited node techniques (discussed in the aobve mentioned post) to get the pointer to the last node. Idea is simple: the very first node whose next is already visited (or hashed) is the last node.
We can also use Floyd Cycle Detection algorithm to detect and remove the loop. In the Floyd's



algo, the slow and fast pointers meet at a loop node. We can use this loop node to remove cycle. There are following two different ways of removing loop when Floyd's algorithm is used for Loop detection.

**Method 1 (Check one by one)**
We know that Floyd's Cycle detection algorithm terminates when fast and slow pointers meet at a common point. We also know that this common point is one of the loop nodes (2 or 3 or 4 or 5 in the above diagram). We store the address of this in a pointer variable say ptr2. Then we start from the head of the Linked List and check for nodes one by one if they are reachable from ptr2. When we find a node that is reachable, we know that this node is the starting node of the loop in Linked List and we can get pointer to the previous of this node.

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

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

/* Function to remove loop. Used by detectAndRemoveLoop() */
void removeLoop(struct node *, struct node *);

/* This function detects and removes loop in the list
  If loop was there in the list then it returns 1,
  otherwise returns 0 */
int detectAndRemoveLoop(struct node *list)
{
    struct node  *slow_p = list, *fast_p = list;

    while (slow_p && fast_p && fast_p->next)
    {
        slow_p = slow_p->next;
        fast_p  = fast_p->next->next;

        /* If slow_p and fast_p meet at some point then there
           is a loop */
        if (slow_p == fast_p)
        {
            removeLoop(slow_p, list);

            /* Return 1 to indicate that loop is found */
```



## Popular Posts

```
            return 1;
        }
    }

    /* Return 0 to indeciate that ther is no loop*/
    return 0;
}

/* Function to remove loop.
 loop_node --> Pointer to one of the loop nodes
 head -->  Pointer to the start node of the linked list */
void removeLoop(struct node *loop_node, struct node *head)
{
   struct node *ptr1;
   struct node *ptr2;

   /* Set a pointer to the beging of the Linked List and
      move it one by one to find the first node which is
      part of the Linked List */
   ptr1 = head;
   while(1)
   {
     /* Now start a pointer from loop_node and check if it ever
       reaches ptr2 */
     ptr2 = loop_node;
     while(ptr2->next != loop_node && ptr2->next != ptr1)
     {
         ptr2 = ptr2->next;
     }

     /* If ptr2 reahced ptr1 then there is a loop. So break the
        loop */
     if(ptr2->next == ptr1)
        break;

     /* If ptr2 did't reach ptr1 then try the next node after ptr1 */
     else
       ptr1 = ptr1->next;
   }

   /* After the end of loop ptr2 is the last node of the loop. So
     make next of ptr2 as NULL */
   ptr2->next = NULL;
}

/* UTILITY FUNCTIONS */
/* Given a reference (pointer to pointer) to the head
```

```
  of a list and an int, pushes a new node on the front
  of the 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 linked list */
void printList(struct node *node)
{
    while(node != NULL)
    {
        printf("%d  ", node->data);
        node = node->next;
    }
}

/* Drier program to test above function*/
int main()
{
    /* Start with the empty list */
    struct node* head = NULL;

    push(&head, 10);
    push(&head, 4);
    push(&head, 15);
    push(&head, 20);
    push(&head, 50);

    /* Create a loop for testing */
    head->next->next->next->next->next = head->next->next;

    detectAndRemoveLoop(head);

    printf("Linked List after removing loop \n");
    printList(head);
```



## Recent Comments

Abhi You live US or India?

Google (Mountain View) interview · 43 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

```
    getchar();
    return 0;
}
```

**Method 2 (Efficient Solution)**

This method is also dependent on Floyd's Cycle detection algorithm.

1) Detect Loop using Floyd's Cycle detection algo and get the pointer to a loop node.

2) Count the number of nodes in loop. Let the count be k.

3) Fix one pointer to the head and another to kth node from head.

4) Move both pointers at the same pace, they will meet at loop starting node.

5) Get pointer to the last node of loop and make next of it as NULL.

Thanks to WgpShashank for suggesting this method.

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

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

/* Function to remove loop. */
void removeLoop(struct node *, struct node *);

/* This function detects and removes loop in the list
  If loop was there in the list then it returns 1,
  otherwise returns 0 */
int detectAndRemoveLoop(struct node *list)
{
    struct node  *slow_p = list, *fast_p = list;

    while (slow_p && fast_p && fast_p->next)
    {
        slow_p = slow_p->next;
        fast_p  = fast_p->next->next;

        /* If slow_p and fast_p meet at some point then there
           is a loop */
        if (slow_p == fast_p)
        {
            removeLoop(slow_p, list);
```

```
            /* Return 1 to indicate that loop is found */
            return 1;
        }
    }

    /* Return 0 to indeciate that ther is no loop*/
    return 0;
}

/* Function to remove loop.
 loop_node --> Pointer to one of the loop nodes
 head -->  Pointer to the start node of the linked list */
void removeLoop(struct node *loop_node, struct node *head)
{
    struct node *ptr1 = loop_node;
    struct node *ptr2 = loop_node;

    // Count the number of nodes in loop
    unsigned int k = 1, i;
    while (ptr1->next != ptr2)
    {
        ptr1 = ptr1->next;
        k++;
    }

    // Fix one pointer to head
    ptr1 = head;

    // And the other pointer to k nodes after head
    ptr2 = head;
    for(i = 0; i < k; i++)
      ptr2 = ptr2->next;

    /*  Move both pointers at the same pace,
      they will meet at loop starting node */
    while(ptr2 != ptr1)
    {
        ptr1 = ptr1->next;
        ptr2 = ptr2->next;
    }

    // Get pointer to the last node
    ptr2 = ptr2->next;
    while(ptr2->next != ptr1)
       ptr2 = ptr2->next;
```

```
    /* Set the next node of the loop ending node
      to fix the loop */
    ptr2->next = NULL;
}

/* UTILITY FUNCTIONS */
/* Given a reference (pointer to pointer) to the head
  of a list and an int, pushes a new node on the front
  of the 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 linked list */
void printList(struct node *node)
{
    while(node != NULL)
    {
        printf("%d  ", node->data);
        node = node->next;
    }
}

/* Drier program to test above function*/
int main()
{
    /* Start with the empty list */
    struct node* head = NULL;

    push(&head, 10);
    push(&head, 4);
    push(&head, 15);
    push(&head, 20);
    push(&head, 50);
```

```
    /* Create a loop for testing */
    head->next->next->next->next->next = head->next->next;

    detectAndRemoveLoop(head);

    printf("Linked List after removing loop \n");
    printList(head);

    getchar();
    return 0;
}
```

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



## Related Tpoics:

- 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



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

## 101 Comments GeeksforGeeks

Sort by Newest

Join the discussion…

**kinshuk chandra** • 3 days ago

Nice post and really good explanation. Here is my java code to find the loop in

```
/**
 * Returns the node at the start of a loop in the given circular linke
 * list. A circular list is one in which a node's next pointer points
 * to an earlier node, so as to make a loop in the linked list. For
 * instance:
 *
 * input: A -> B -> C -> D -> E -> C [the same C as earlier]
 * output: C
 *
 *
 * @param linkedList
 *            list to be tested
 * @return the node at the start of the loop if there is a loop, null
 * otherwise
 */
```

see more

• Reply • Share ›

**Rajesh M D** • 7 days ago

For METHOD 2:
we can eliminate one while loop of K iterations. Just for moving to last node, w
one more pointer as ptr2_prev to point to previous pointer of prt2. Here is the b

```
/* Move both pointers at the same pace,
they will meet at loop starting node */

struct node* ptr2_prev = NULL; // add this line

while(ptr2 != ptr1)
{
ptr1 = ptr1->next;
ptr2_prev = ptr2;
ptr2 = ptr2->next;

}

// Get pointer to the last node
ptr2 = ptr2->next; // comment this line
```

see more

• Reply • Share ›

**Aditya Gaurav** • 16 days ago

let y be the length of no. of nodes in loop & x be that in linear chain (excluding t
In both cases (i) y>x and (ii) y<x after="" the="" loop="" detection="" and="" pla
(one="" at="" beg="" of="" list="" and="" another="" at="" yth="" node="" from="
exactly="" x="" steps="" to="" meet="" at="" loop="" entry="" node...="">

**AMIT JAMBOTKAR** • 24 days ago

For Java Lovers

```
public class LinkedList<e> implements Cloneable{

Node<e> head = null;

class Node<t> {

T value;

Node<t> nextReference;

public Node(T value) {

this.value = value;

this.nextReference = null;

}

public Node(T value, Node<t> ref) {
```

see more



**Rahul Maheshwari** • a month ago

Why you guys counting number of nodes.. Look my solution .. Its easy and rob

```
void remove(struct node *slow , struct node *head){

struct node *ptr1 = head ; struct node *ptr2 = head;

while(ptr2->next != slow){
```

```
ptr2 = ptr2->next;

}

while(slow != ptr1){

ptr1=ptr1->next;

ptr2 = slow;

slow = slow->next;

}
```

see more

Reply • Share ›

**AMIT JAMBOTKAR** Rahul Maheshwari • 25 days ago

How your solution will guarantee that your pointing at last node only spe
ptr2->next = NULL;

Reply • Share ›

**Vishal** • 2 months ago

```
Node* detectAndRemoveLoop(Node *head)
{
Node *slow,*fast,*temp;
temp = slow = fast = head;
while(slow && fast &&fast->next)
{
slow = slow->next;
fast = fast->next->next;

if(slow == fast)
break;
```

```
}
if(slow == fast && (slow && fast &&fast->next) )
{
fast->next->next = NULL;
}

return head;
}
```

Reply • Share ›

**Abhijit** • 4 months ago

Why do we need all the three pointers in while loop? i.e while(slow && fast &&

Reply • Share ›

**Himanshu Dagar** ➦ Abhijit • 3 months ago

no
but you can omit slow frm here
But fast and fast->next both are necessary otherwise ur program may

Reply • Share ›

**Vishal Hemnani** ➦ Abhijit • 4 months ago

Needed to avoid NullPointer when we do fast = fast->next->next ..

Reply • Share ›

**Somashekhar Ganjigatti** • 4 months ago

what if only two nodes are there - how to find loop - thanks

Reply • Share ›

**Abhi** ➦ Somashekhar Ganjigatti • 4 months ago

This works for two nodes also. First loop it wont be detected, but in ne
points again to head, hence in the succeeding loop it will be caught.

Reply • Share ›

**gaurav** • 5 months ago

what is problem with this code

```
#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

struct node

{

int data;

struct node *next;

};

void push(struct node **start,int num)

{
```

see more

• Reply • Share ›

**Arjun Rana** • 5 months ago

I think when we got the common node using Floyd's Algorithm... then using a v
we easily remove the node...

I post my source code :

```
#include<stdio.h>

#include<conio.h>
```

#include<conio.h>

int count =0;

struct node {

int data;

struct node* link;

}node;

struct node* head = NULL;

see more

• Reply • Share ›

**shri1729** • 5 months ago

when u got the last node using Floyd's Algorithm .. why dont u make that node
void removeLoop(struct node *loop_node, struct node *head) function.

• Reply • Share ›

**Gaurav Reddy** • 5 months ago

Alternative way :
- Use floy'd algo to find the cycle using slow pointer.
- At the node where we detect the loop make its next NULL
(we have a two list with a intersection point)
- Find the point of intersection of two lists, which is the head of loop
- now revert back to the original list and make the ptr->next == intersection_nc

1 • Reply • Share ›

**Rajesh M D** ➦ Gaurav Reddy • 7 days ago

very smart dude.. i liked it, but two problems are there.
FIRST:

for very rare case, what if we detects the loop at previous of intersectic
NULL . we will break the intersection there only. then it wont be two list
Eg:
1->2->3->4->5->6->7->8
.....|..... |
.....13<-12 <-11<-10 <- 9

what if slow and fast pointer point to 13.
Then while making slow->next as NULL, we are breaking the loop (13-

Second:
when two list is large say list1 of 'm' length and list2 of 'n' length. then i
intersection which is not efficient compared to method 2.

Reply • Share ›

**samsammy** Gaurav Reddy • 5 months ago
Good One....

Reply • Share ›

**Rohit Ramsen** samsammy • a month ago
I think its better then above mentioned both ways to remove the

Reply • Share ›

**ashish** Rohit Ramsen • 17 days ago
i don get it

Reply • Share ›

**anil** • 6 months ago
void removeLoop(struct node *loop_node, struct node *head)
{
struct node *ptr1 = loop_node;
struct node *ptr2 = loop_node;

```
// Count the number of nodes in loop
unsigned int k = 0, i;
while (ptr1->next != ptr2)
{
ptr1 = ptr1->next;
k++;
}
for(i = 0; i < k; i++)
ptr2 = ptr2->next;

ptr2->next = NULL;
}
```

1 ▲ | ▼ • Reply • Share ›

**Sandeep** • 6 months ago

Not tested it but think it will work

```
slow=fast=ptr=head;

while(ptr!=fast->next->next) {

while(ptr!=fast->next->next) {

slow=slow->next;

fast=fast->next->next;

if(slow==fast)

break;

}

slow=fast=ptr=ptr->next;
```

}

fast->next->next=NULL;

1 ^ | v • Reply • Share ›

**Sandeep** ➚ Sandeep • 6 months ago

missed if statement after inner while
if(ptr==fast->next->next)
break;

^ | v • Reply • Share ›

**ravi** • 7 months ago

why 2nd one is efficient?????????????

^ | v • Reply • Share ›

**sumit** • 7 months ago

what is time complexity of both method ??

^ | v • Reply • Share ›

**Swastik Sahu** • 9 months ago

Or maintain a pointer(say *t) to keep track of the node prev to the node pointec
When fast pointer == slow pointer, t->next = NULL.
done.

^ | v • Reply • Share ›

**Dinesh Khowal** • 9 months ago

where r u freeing the memory!

^ | v • Reply • Share ›

**Dinesh Khowal** • 9 months ago

where r u freeing the memory!

^ | v • Reply • Share ›

**Kush Pandey** • 10 months ago

According to me we can apply the logic of circular linked list to detect and rem
The function is

```
 node *detectandremove(node *start)
{
while(start!=0)
{
if(start->next==start)
{
printf("Loop found");
start->next=0;
break;
}
start=start->next;
}
return(start);
}
```

• Reply • Share ›

**Karthikeya Yakkali** • 10 months ago

Method 2 is very good and simple. its simply awesome..Thanx alot for the met

• Reply • Share ›

**Susheel Pandey** • 11 months ago

I have suggestion here---------
we can replace this code snippet from the method 2.
by ----(see after this snippet)--
---

they will meet at loop starting node */

they will meet at loop starting node */
while(ptr2!= ptr1)
{
ptr1 = ptr1->next;
ptr2 = ptr2->next;
}
// Get pointer to the last node.
ptr2 = ptr2->next;.
while(ptr2->next!= ptr1).
ptr2 = ptr2->next;.
/* Set the next node of the loop ending node.

to fix the loop */.
ptr2->next = NULL;.

see more

1 ▲ | ▼ • Reply • Share ›

**ultimate_coder** • 11 months ago

Method 2 :
Why we need to skip one pointer k forward?
I think we can find the starting node of the loop using following pointers.
1. ptr1=head
2. ptr2=loop_node // where they coincide

And then increment both of them one by one.
Now above step will give us the starting node of the loop.

So, now we can also find out the ending node of the loop without moving one p

I think it is more optimized version of Method 2.
Correct me if i am wrong.

2 ▲ | ▼ • Reply • Share ›

**Pratik Shah** ultimate_coder • 5 months ago

This wont work if 1->2->(again 1) . It ll end up in a infinite loop. becaus
will be on head. ie 1.

Hence its necessary to go ahead and calculate the no of nodes in the l

Reply • Share ›

**hariprasaadssalem** ultimate_coder • 10 months ago

I thought the same thing. It must work.

Reply • Share ›

**ultimate_coder** ultimate_coder • 11 months ago

It also don't need to find loop length.

Reply • Share ›

**Avaneesh Kumar** • a year ago

DETECT AND REMOVE LOOP IN SINGLY LINKED LIST (VERY COMMON E
BUT I HAVE A WELL SOLUTION EASY TO UNDERSTAND :), AS YOU KNO\

C++ CODE.

```
// program to degtect and remove loop in singly linked list.
/*** AUTHOR @ AVANEESH KUMAR2013 , BIET JHANSI, prmrs111@live.cor
#include<stdio.h>
#include<cstring>
#include<iostream.h>
#include<string.h>
#include<map>
#include<deque>
#include<queue>
#include<stack>
#include<sstream>
#include<iostream>
```

#include<iomanip>
#include<cstdio>

see more

• Reply • Share ›

**Fresher_1** • a year ago

Hello Geeks,
I use the following case for the checking the answer.
1->2->3->4->5->6->3...!
that is 3 is looping point,
here k=4.
ptr1=1 and ptr2= 4 now we move it by one by one so
but they never meets coz they have diff of one in the loops.
please correct me if i`m wrong ?

• Reply • Share ›

**Sreenivas Doosa** Fresher_1 • a year ago

@Fresher_1:

In your example, kth node from head is 5, (not 4, dont consider the hea

Please have a closer look at the code :)

• Reply • Share ›

**abhishek08aug** • a year ago

Intelligent :D

• Reply • Share ›

**aman1234** abhishek08aug • 11 months ago

what happened to you ? why do you reply with intelligent comment on e

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

1 ▲ | ▼ • Reply • Share ›

**Soumya Sengupta** • a year ago

@geeksforgeeks.....cnt we store the address of the node @ which loop starts move the pointr @newnode unless it reaches its originally occupied position..v location.....
and equate that pointr to null...

pls reply???

▲ | ▼ • Reply • Share ›

**Ankit** • a year ago

Why we cannot put NULL after loop_node, I mean if there is a loop that must b of pointing to NULL.. last element is now pointing to some other node..
So we just have to do this;
last_node->next==NULL;

:-/

▲ | ▼ • Reply • Share ›

**Sreenivas Doosa** ➦ Ankit • a year ago

@Ankit,

The loop_node may not be the last node in the loop. It is one of the nod

See this example
1->2->3->4->5->6
Lets say next of node 6 is 3.

In this example, if you apply floyd`s loop detection algo.. the loop_node

▲ | ▼ • Reply • Share ›

**Sonal** ➚ Sreenivas Doosa • 3 months ago

Thanks Sreenivas.. I too had the same doubt

^ | v • Reply • Share ›

**Ankit** • a year ago

Why we cannot just put null after link_pointer. i mean

2-->3-->4-->5-->6--|

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

^ | v • Reply • Share ›

**Saira** • a year ago

One solution is,
when u detect the loop using fast and slow pointer, then move the slow pointe
slow = head.. Then move the slow and fast pointer at the same pace i.e. slow
must meet at Loop start.

^ | v • Reply • Share ›

**vedverma1** ➚ Saira • 9 months ago

1-2-3-4-5
| |
9-8-7-6
If we consider this linklist den slowpointer and fastpointer will meet at n
with head and move both pointerswith same pace, den they will never
know.

^ | v • Reply • Share ›

**ultimate_coder** ➚ vedverma1 • 9 months ago

```
dude it cant be singly linked list. your node 4 has two
```

Reply • Share ›

vedverma1 ultimate_coder • 9 months ago

dude , its display error, my linklist was like 1->2->3->4->
back to node2. now recheck the logic.

Reply • Share ›

indra2gurjar Saira • 11 months ago

@Saira :
IT NOT possible when loop length is << n,where n is distance from hea
plz explain if i'm wrong or misinterpreted your solution.

Reply • Share ›

Load more comments

Subscribe Add Disqus to your site