Saturday, January 8, 2022 7:23 PM

Remove loop in Linked List

Medium Accuracy: 47.96% Submissions: 100k+ Points: 4

Given a linked list of **N** nodes such that it may contain a loop.

A loop here means that the last node of the link list is connected to the node at position X. If the link list does not have any loop, X=0.
Remove the loop from the linked list, if it is present.

Example 1:

```
Input:
N = 3
value[] = {1,3,4}
X = 2
Output: 1
Explanation: The link list looks like
1 -> 3 -> 4
     ^    |
     |____|
A loop is present. If you remove it
successfully, the answer will be 1.
```

Example 2:

```
Input:
N = 4
value[] = {1,8,3,4}
X = 0
Output: 1
Explanation: The Linked list does not
contains any loop.
```

Example 3:

```
Input:
N = 4
value[] = {1,2,3,4}
```

X = 1
Output: 1
Explanation: The link list looks like
1 -> 2 -> 3 -> 4
|_______________|
A loop is present.
If you remove it successfully,
the answer will be 1.

Your Task:

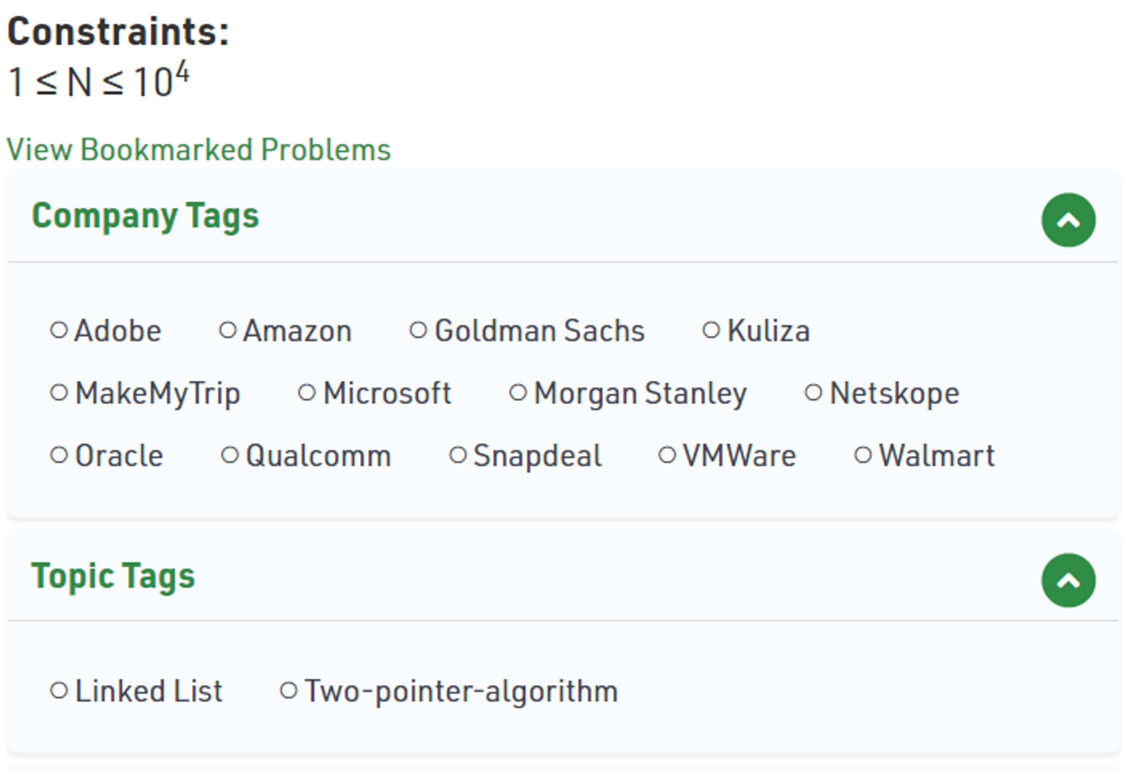
You don't need to read input or print anything. Your task is to complete the function **removeLoop**() which takes the head of the linked list as the input parameter. Simply remove the loop in the list (if present) without disconnecting any nodes from the list.

Note: The generated output will be **1** if your submitted code is correct.

Expected time complexity: O(N)

Expected auxiliary space: O(1)

From <https://practice.geeksforgeeks.org/problems/remove-loop-in-linked-list/1#>



Constraints:
$1 \leq N \leq 10^4$

View Bookmarked Problems

Company Tags

- Adobe
- Amazon
- Goldman Sachs
- Kuliza
- MakeMyTrip
- Microsoft
- Morgan Stanley
- Netskope
- Oracle
- Qualcomm
- Snapdeal
- VMWare
- Walmart

Topic Tags

- Linked List
- Two-pointer-algorithm

Related Courses

- DSA-Self Paced with Doubt Assistance
- Data Structures and Algorithms - Self Paced
- Amazon SDE Preparation Test Series
- Placement 100
- Complete Interview Preparation - Self Paced
- Complete Interview Preparation With Doubt Assistance
- Microsoft SDE Preparation Test Series
- Must Do Coding Questions - Self Paced
- Competitive Programming - Live
- DSA Live for Working Professionals - Live
- Data Structures with C++ Live
- Advance DSA Training Program - Live
- Advance DSA Training Course
- Competitive Programing (Basic to Advanced) - Self Paced

Related Interview Experiences

- Vmware interview experience set 9 internship rd
- Walmart labs interview experience set 2 on campus
- Makemytrip interview experience set 8 on campus
- Adobe interview experience shecodes software engineer

```
public class removeLoop {
    public static void removeLoop(Node head){
        // code here
        // remove the loop without losing any nodes
        Node fast=head;
        Node slow=head;

        boolean loop=false ;
```

```java
while(slow!=null && fast!=null && fast.next!=null)
{
    fast=fast.next.next;
    slow=slow.next;
    if(fast==slow)
    {
        loop=true;
        break;
    }

}

//  now // 1 2 3 4 5
//           ^|____|^
//
//
 int count=1;
  boolean removed=false;


if(loop)
{


    fast=fast.next;

    while(fast!=slow)
    {

        count++;
         fast=fast.next;

    }


    slow=head;
    fast=head;



   for(int i=0;i<count;i++)
        {


            fast=fast.next;


        }

  while(fast!=slow)
{
    slow=slow.next;
    fast=fast.next;

}
while(fast.next!=slow)
{
    fast=fast.next;
```

```
        }
        fast.next=null;


        }



    }
}
```

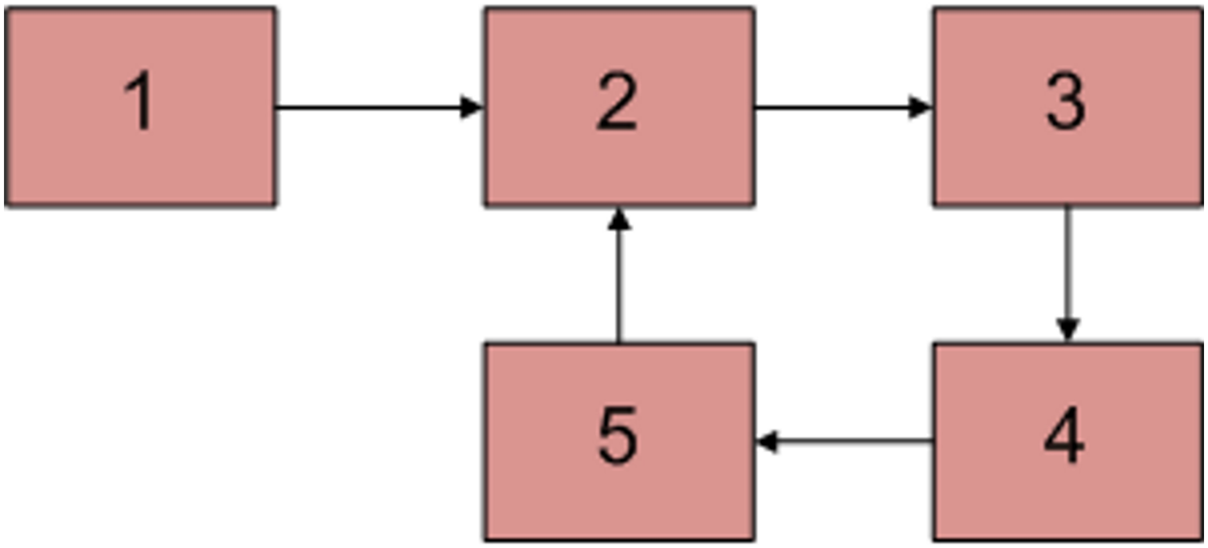
Find first node of loop in a linked list

- Difficulty Level : Medium
- Last Updated : 20 Oct, 2021

Write a function *findFirstLoopNode()* that checks whether a given Linked List contains a loop. If the loop is present then it returns point to the first node of the loop. Else it returns NULL.

Example :

```
Input : Head of below linked list
```



```
Output : Pointer to node 2
```

Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.

We have discussed Floyd's loop detection algorithm. Below are steps to find the first node of the loop.

1. If a loop is found, initialize a slow pointer to head, let fast pointer be at its position.

2. Move both slow and fast pointers one node at a time.

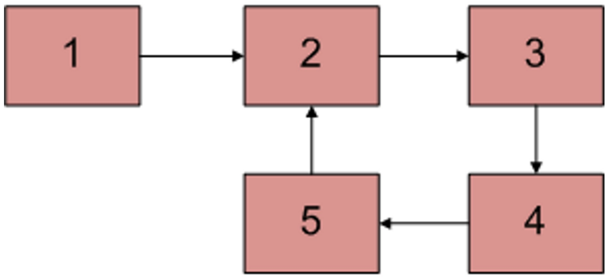
3. The point at which they meet is the start of the loop.

From <https://www.geeksforgeeks.org/find-first-node-of-loop-in-a-linked-list/>

The following is a GFG article/problem that can be used to understand the concepts used in this problem. It may or may not be exactly applicable to the problem statement given here.

Access Full Video Editorial for this Problem by becoming a Premium Member. Click Here !

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. If the list doesn't contain loop then it returns false. Below diagram shows a linked list with a loop. *detectAndRemoveLoop()* must change the below list to 1->2->3->4->5->NULL.



Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

We also recommend to read following post as a prerequisite of the solution discussed here.

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 above 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). Store the address of this in a pointer variable say ptr2. After that start from the head of the Linked List and check for nodes one by one if they are reachable from ptr2. Whenever 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 the pointer to the previous of this node.

Output:

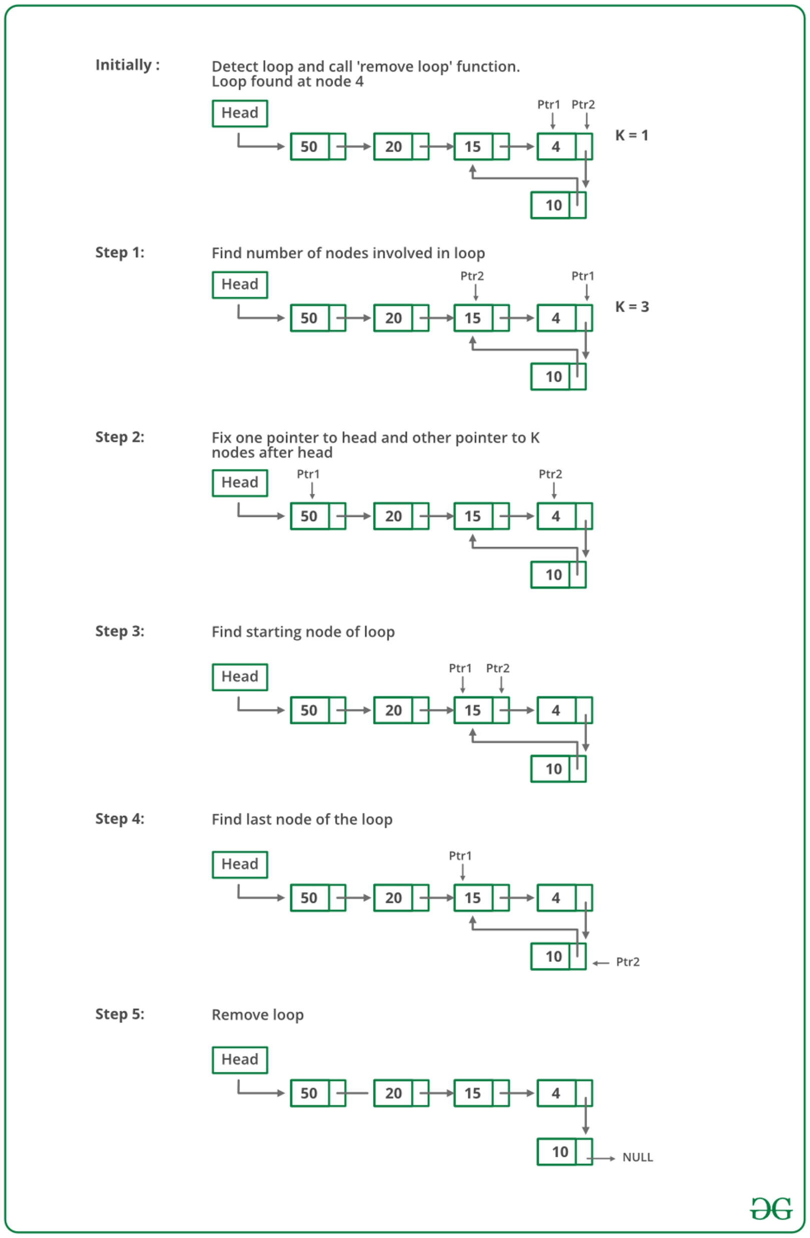
```
Linked List after removing loop
50 20 15 4 10
```

Method 2 (Better Solution)

1. This method is also dependent on Floyd's Cycle detection algorithm.
2. Detect Loop using Floyd's Cycle detection algorithm and get the pointer to a loop node.
3. Count the number of nodes in loop. Let the count be k.
4. Fix one pointer to the head and another to a kth node from the head.
5. Move both pointers at the same pace, they will meet at loop starting node.
6. Get a pointer to the last node of the loop and make next of it as NULL.

Thanks to WgpShashank for suggesting this method.

Below image is a dry run of 'remove loop' function in the code :



Initially :
Detect loop and call 'remove loop' function. Loop found at node 4
Head
50
20
15
4
10
Ptr1
Ptr2
K = 1
Step 1:
Find number of nodes involved in loop
Ptr2
Ptr1
K = 3
Step 2:
Fix one pointer to head and other pointer to K nodes after head
Ptr1
Ptr2
Step 3:
Find starting node of loop
Ptr1
Ptr2
Step 4:
Find last node of the loop
Ptr1
Ptr2
Step 5:
Remove loop
NULL

Below is the implementation of the above approach:

C++CJava

```
// Java program to detect and remove loop in linked list

class LinkedList {

static Node head;

static class Node {

int data;
Node next;

Node(int d)
{
data = d;
next = null;
}
}

// Function that detects loop in the list
int detectAndRemoveLoop(Node node)
{
Node slow = node, fast = node;
while (slow != null && fast != null && fast.next != null) {
slow = slow.next;
fast = fast.next.next;

// If slow and fast meet at same point then loop is present
if (slow == fast) {
removeLoop(slow, node);
return 1;
}
}
return 0;
}

// Function to remove loop
void removeLoop(Node loop, Node head)
{
Node ptr1 = loop;
Node ptr2 = loop;

// Count the number of nodes in loop
```

```
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
while (ptr2.next != ptr1) {
ptr2 = ptr2.next;
}

/* Set the next node of the loop ending node
to fix the loop */
ptr2.next = null;
}

// Function to print the linked list
void printList(Node node)
{
while (node != null) {
System.out.print(node.data + " ");
node = node.next;
}
}

// Driver program to test above functions
public static void main(String[] args)
{
LinkedList list = new LinkedList();
list.head = new Node(50);
list.head.next = new Node(20);
list.head.next.next = new Node(15);
list.head.next.next.next = new Node(4);
list.head.next.next.next.next = new Node(10);

// Creating a loop for testing
head.next.next.next.next.next = head.next.next;
list.detectAndRemoveLoop(head);
System.out.println("Linked List after removing loop : ");
list.printList(head);
}
}

// This code has been contributed by Mayank Jaiswal
```

Run

Python3C#Javascript

Output:

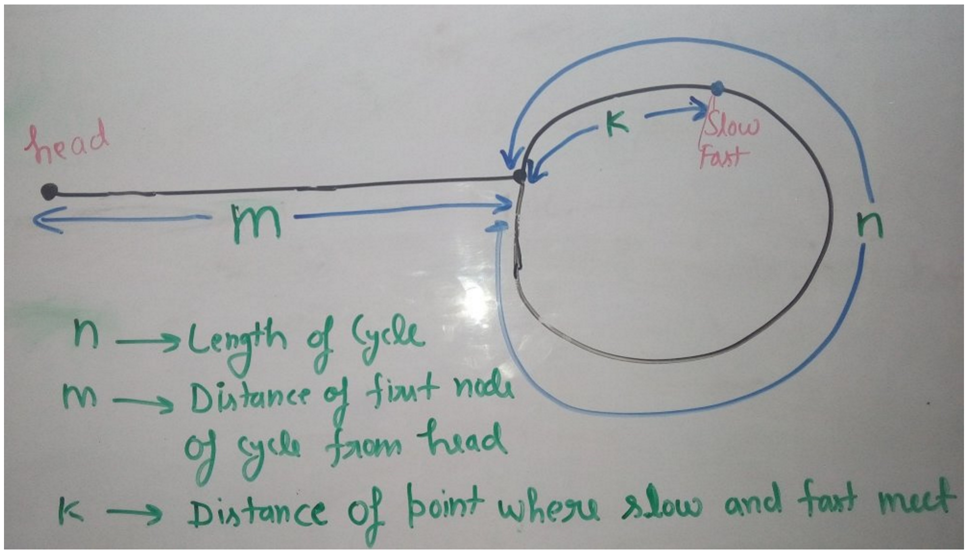
```
Linked List after removing loop
50 20 15 4 10
```

Method 3 (Optimized Method 2: Without Counting Nodes in Loop)

We do not need to count number of nodes in Loop. After detecting the loop, if we start slow pointer from head and move both slow and fast pointers at same speed until fast don't meet, they would meet at the beginning of the loop.

How does this work?

Let slow and fast meet at some point after Floyd's Cycle finding algorithm. Below diagram shows the situation when cycle is found.


head
m
K
Slow
Fast
n
n → Length of Cycle
m → Distance of first node of cycle from head
K → Distance of point where slow and fast meet

We can conclude below from above diagram

```
Distance traveled by fast pointer = 2 * (Distance traveled
by slow pointer)


(m + n*x + k) = 2*(m + n*y + k)


Note that before meeting the point shown above, fast

was moving at twice speed.


x -->  Number of complete cyclic rounds made by

fast pointer before they meet first time


y -->  Number of complete cyclic rounds made by

slow pointer before they meet first time
```

From above equation, we can conclude below

```
    m + k = (x-2y)*n
```

```
Which means m+k is a multiple of n.

Thus we can write, m + k = i*n or m = i*n - k.

Hence, distance moved by slow pointer: m, is equal to distance moved by fast
pointer:

i*n - k or (i-1)*n + n - k (cover the loop completely i-1 times and start from n-
k).
```

So if we start moving both pointers again at **same speed** such that one pointer (say slow) begins from head node of linked list and other pointer (say fast) begins from meeting point. When slow pointer reaches beginning of loop (has made m steps), fast pointer would have made also moved m steps as they are now moving same pace. Since m+k is a multiple of n and fast starts from k, they would meet at the beginning. Can they meet before also? No because slow pointer enters the cycle first time after m steps.

C++JavaPython3C#Javascript

Output:

```
Linked List after removing loop
50 20 15 4 10
```

Method 4 Hashing: Hash the address of the linked list nodes

We can hash the addresses of the linked list nodes in an unordered map and just check if the element already exists in the map. If it exists, we have reached a node which already exists by a cycle, hence we need to make the last node's next pointer NULL.

C++Java

```
// Java program to detect and remove loop in a linked list
```

```
import java.util.*;

public class LinkedList {

static Node head; // head of list

/* Linked list Node*/
static class Node {
int data;
Node next;
Node(int d)
{
data = d;
next = null;
}
}

/* Inserts a new Node at front of the list. */
static public void push(int new_data)
{
/* 1 & 2: Allocate the Node &
Put in the data*/
Node new_node = new Node(new_data);

/* 3. Make next of new Node as head */
new_node.next = head;

/* 4. Move the head to point to new Node */
head = new_node;
}

// Function to print the linked list
void printList(Node node)
{
while (node != null) {
System.out.print(node.data + " ");
node = node.next;
}
}

// Returns true if the loop is removed from the linked
// list else returns false.
static boolean removeLoop(Node h)
{
HashSet<Node> s = new HashSet<Node>();
Node prev = null;
while (h != null) {
```

```
// If we have already has this node
// in hashmap it means their is a cycle and we
// need to remove this cycle so set the next of
// the previous pointer with null.

if (s.contains(h)) {
prev.next = null;
return true;
}

// If we are seeing the node for
// the first time, insert it in hash
else {
s.add(h);
prev = h;
h = h.next;
}
}

return false;
}

/* Driver program to test above function */
public static void main(String[] args)
{
LinkedList llist = new LinkedList();

llist.push(20);
llist.push(4);
llist.push(15);
llist.push(10);

/*Create loop for testing */
llist.head.next.next.next.next = llist.head;

if (removeLoop(head)) {
System.out.println("Linked List after removing loop");
llist.printList(head);
}
else
System.out.println("No Loop found");
}
}

// This code is contributed by Animesh Nag.
```
Run
C#Javascript

Output

```
Linked List after removing loop
50 20 15 4 10
```

We Thank Shubham Agrawal for suggesting this solution.

Thanks to Gaurav Ahirwar for suggesting the above solution.

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

From <https://practice.geeksforgeeks.org/problems/remove-loop-in-linked-list/1#>