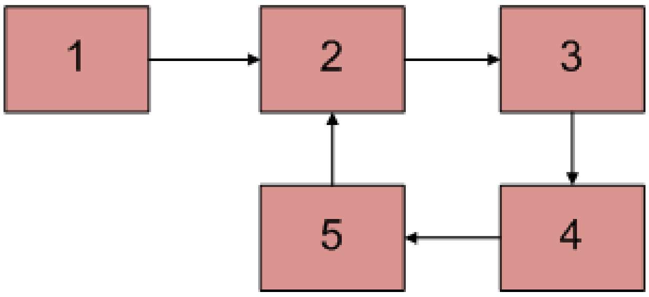
Detect and Remove Loop in a Linked List

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 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). 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.

C

```c
#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 */
            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 */
         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;
}

/* Function to print linked list */
void printList(struct node *node)
{
    while (node != NULL)
    {
        printf("%d  ", node->data);
        node = node->next;
    }
}

struct node *newNode(int key)
{
    struct node *temp = new struct node;
    temp->data = key;
    temp->next = NULL;
    return temp;
}

/* Drier program to test above function*/
int main()
{
    struct node *head = newNode(50);
    head->next = newNode(20);
    head->next->next = newNode(15);
    head->next->next->next = newNode(4);
    head->next->next->next->next = newNode(10);

    /* 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);
    return 0;
}
```

Run on IDE

Java

```
// 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 curr) {
        Node ptr1 = null, ptr2 = null;

        /* 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 = curr;
        while (1 == 1) {

            /* Now start a pointer from loop_node and check if it ever
             reaches ptr2 */
            ptr2 = loop;
            while (ptr2.next != loop && 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 */
            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;
    }

    // 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 on IDE

Python

```
# Python program to detect and remove loop in linked list

# Node class
class Node:

    # Constructor to initialize the node object
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:

    # Function to initialize head
    def __init__(self):
        self.head = None

    def detectAndRemoveLoop(self):
        slow_p = fast_p = self.head
        while(slow_p and fast_p and fast_p.next):
            slow_p = slow_p.next
            fast_p = fast_p.next.next

            # If slow_p and fast_p meet at some poin
            # then there is a loop
            if slow_p == fast_p:
                self.removeLoop(slow_p)

                # Return 1 to indicate that loop if found
                return 1

        # Return 0 to indicate that there 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
    def removeLoop(self, loop_node):

        # Set a pointer to the beginning of the linked
        # list and move it one by one to find the first
        # node which is part of the linked list
        ptr1 = self.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 and ptr2.next !=ptr1):
                ptr2 = ptr2.next

            # If ptr2 reached ptr1 then there is a loop.
            # So break the loop
            if ptr2.next == ptr1 :
                break
            ptr1 = ptr1.next

        # After the end of loop ptr2 is the lsat node of
        # the loop. So make next of ptr2 as NULL
        ptr2.next = None
    # Function to insert a new node at the beginning
    def push(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        self.head = new_node

    # Utility function to prit the linked LinkedList
    def printList(self):
        temp = self.head
        while(temp):
            print temp.data,
            temp = temp.next


# Driver program
llist = LinkedList()
llist.push(10)
llist.push(4)
llist.push(15)
llist.push(20)
llist.push(50)

# Create a loop for testing
llist.head.next.next.next.next.next = llist.head.next.next

llist.detectAndRemoveLoop()

print "Linked List after removing loop"
llist.printList()

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Run on IDE

Output:

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

Method 2 (Better 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.

C

```
#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;
}

/* Function to print linked list */
void printList(struct node *node)
{
    while (node != NULL)
    {
        printf("%d  ", node->data);
        node = node->next;
    }
}

struct node *newNode(int key)
{
    struct node *temp = new struct node;
    temp->data = key;
    temp->next = NULL;
    return temp;
}

/* Driver program to test above function*/
int main()
{
    struct node *head = newNode(50);
    head->next = newNode(20);
    head->next->next = newNode(15);
    head->next->next->next = newNode(4);
    head->next->next->next->next = newNode(10);

    /* 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);
    return 0;
}
```

Run on IDE

Java

```
// 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
        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;
    }

    // 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 on IDE

Python

```
# Python program to detect and remove loop in linked list

# Node class
class Node:

    # Constructor to initialize the node object
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:

    # Function to initialize head
    def __init__(self):
        self.head = None

    def detectAndRemoveLoop(self):
        slow_p = fast_p = self.head

        while(slow_p and fast_p and 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:
                self.removeLoop(slow_p)

                # Return 1 to indicate that loop is found
                return 1

        # Return 0 to indicate that there 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
    def removeLoop(self, loop_node):
        ptr1 = loop_node
        ptr2 = loop_node

        # Count the number of nodes in loop
        k = 1
        while(ptr1.next != ptr2):
            ptr1 = ptr1.next
            k += 1

        # Fix one pointer to head
        ptr1 = self.head
        # And the other pointer to k nodes after head
        ptr2 = self.head
        for i in range(k):
            ptr2 = ptr2.next

        # Move both pointers at the same place
        # 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 = None

    # Function to insert a new node at the beginning
    def push(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        self.head = new_node

    # Utility function to prit the linked LinkedList
    def printList(self):
        temp = self.head
        while(temp):
            print temp.data,
            temp = temp.next


# Driver program
llist = LinkedList()
llist.push(10)
llist.push(4)
llist.push(15)
llist.push(20)
llist.push(50)

# Create a loop for testing
llist.head.next.next.next.next.next = llist.head.next.next

llist.detectAndRemoveLoop()

print "Linked List after removing loop"
llist.printList()

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Run on IDE

Output:

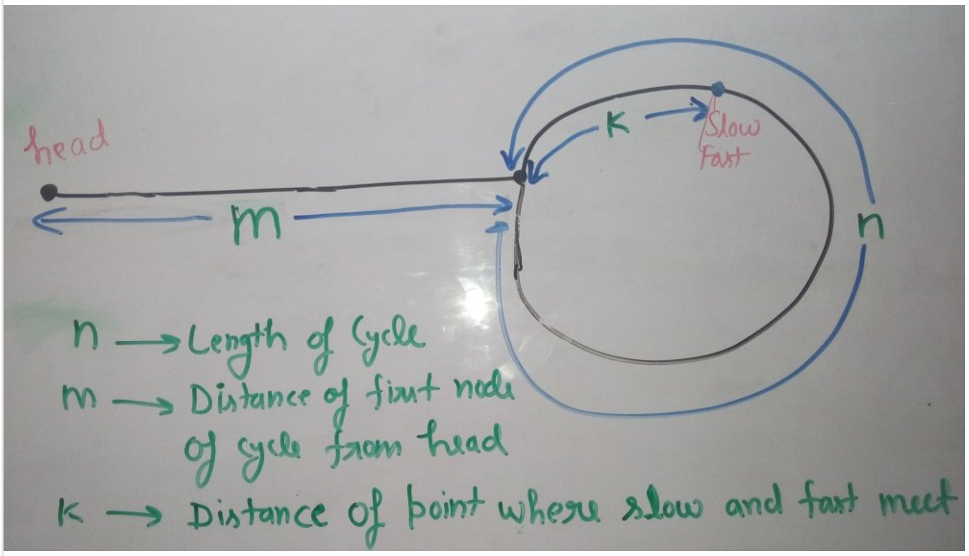
```
Linked List after removing loop
50 20 15 4 10
```

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 linked list.

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.
```

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 linked list (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++

```
// C++ program to detect and remove loop
#include<bits/stdc++.h>
using namespace std;

struct Node
{
    int key;
    struct Node *next;
};

Node *newNode(int key)
{
    Node *temp = new Node;
    temp->key = key;
    temp->next = NULL;
    return temp;
}

// A utility function to print a linked list
void printList(Node *head)
{
    while (head != NULL)
    {
        cout << head->key << " ";
        head = head->next;
    }
    cout << endl;
}

void detectAndRemoveLoop(Node *head)
{
    Node *slow = head;
    Node *fast = head->next;

    // Search for loop using slow and fast pointers
    while (fast && fast->next)
    {
        if (slow == fast)
            break;
        slow = slow->next;
        fast = fast->next->next;
    }

    /* If loop exists */
    if (slow == fast)
    {
        slow = head;
        while (slow != fast->next)
        {
            slow = slow->next;
            fast = fast->next;
        }

        /* since fast->next is the looping point */
        fast->next = NULL; /* remove loop */
    }
}

/* Driver program to test above function*/
int main()
{
    Node *head = newNode(50);
    head->next = newNode(20);
    head->next->next = newNode(15);
    head->next->next->next = newNode(4);
    head->next->next->next->next = newNode(10);

    /* 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);

    return 0;
}
```

Run on IDE

Java

```
// 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
    void detectAndRemoveLoop(Node node) {
        Node slow = node;
        Node fast = node.next;

        // Search for loop using slow and fast pointers
        while (fast != null && fast.next != null) {
            if (slow == fast) {
                break;
            }
            slow = slow.next;
            fast = fast.next.next;
        }

        /* If loop exists */
        if (slow == fast) {
            slow = node;
            while (slow != fast.next) {
                slow = slow.next;
                fast = fast.next;
            }

            /* since fast->next is the looping point */
            fast.next = null; /* remove loop */

        }
    }

    // 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 on IDE

Python

```
# Python program to detect and remove loop

# Node class
class Node:

    # Constructor to initialize the node object
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:

    # Function to initialize head
    def __init__(self):
        self.head = None

    # Function to insert a new node at the beginning
    def push(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        self.head = new_node


    def detectAndRemoveLoop(self):
        slow = self.head
        fast = self.head.next

        # Search for loop using slow and fast pointers
        while(fast is not None):
            if fast.next is None:
                break
            if slow == fast :
                break
            slow = slow.next
            fast = fast.next.next

        # if loop exists
        if slow == fast :
            slow = self.head
            while(slow != fast.next):
                slow = slow.next
                fast = fast.next

            # Sinc fast.next is the looping point
            fast.next = None # Remove loop


    # Utility function to print the linked LinkedList
    def printList(self):
        temp = self.head
        while(temp):
            print temp.data,
            temp = temp.next


# Driver program
llist = LinkedList()
llist.head = Node(50)
llist.head.next = Node(20)
llist.head.next.next = Node(15)
llist.head.next.next.next = Node(4)
llist.head.next.next.next.next = Node(10)

#Create a loop for testing
llist.head.next.next.next.next.next =  llist.head.next.next

llist.detectAndRemoveLoop()

print "Linked List after removing looop"
llist.printList()

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Run on IDE

Output:

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

Thanks to Gaurav Ahirwar for suggesting above solution.

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

Company Wise Coding Practice **Topic Wise Coding Practice**

291 Comments Category: Linked Lists

Related Posts:

- Convert a Binary Tree to a Circular Doubly Link List
- Subtract Two Numbers represented as Linked Lists
- Rearrange a given list such that it consists of alternating minimum maximum elements
- Flatten a multi-level linked list | Set 2 (Depth wise)
- Decimal Equivalent of Binary Linked List
- Merge K sorted linked lists
- Check if a linked list is Circular Linked List
- Delete middle of linked list

(Login to Rate and Mark)

3.4 Average Difficulty : **3.4/5.0**
Based on **120** vote(s)

☐ Add to TODO List
☐ Mark as DONE

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

291 Comments **GeeksforGeeks** 1 **Login**

Recommend 10 **Share** **Sort by Newest**

Join the discussion…

AB • 6 days ago

another simplified solution
http://code.geeksforgeeks.org/...

• Reply • Share ›

Shashank • 10 days ago

It should be slow>next=NULL

• Reply • Share ›

Shashank Dash • 10 days ago

It should be slow=slow>next; in/*remove loop comment*/

• Reply • Share ›

sw • 12 days ago

Thank you for the explanations. Like some others, I did not at first get the explanation but the approach gave me the idea to try out. Here is how I understood the third method.

The insight for me was to break the distance m into.

m = A * n + b

that is, it is an integral multiple of the loop length plus a few remaining nodes. b is obviously less than n. (0 .. n-1); and A = (0, 1, 2 ...)

The above break down already helps us understand the conclusion of the algorithm. If p1 is at head, and p2 is at k inside the loop, then after every n increments, p2 is back at k, and p1 advances along m by n nodes. Ultimately, it will consume all multiples of n and will only have b more nodes left to cover to reach the start of the loop. At this point p2 is back at k.

Then according to the algorithm, advancing both pointers, they meet at the start of the loop. This means both pointers must be only b units away from start of the loop. p1

see more

1 • Reply • Share ›

Piyush • 20 days ago

public Node detectAndRemoveLoop(Node node) {
Node n1 = head;
Node n = head;
while(n1!=null){
while(n!=null){
if(n1!=n.next)
n=n.next;
else{

```
n.next = null;
return n;
}

}
n1=n1.next;
n= n1;
}
return n;
}
```

• Reply • Share ›

amandeep gupta • 21 days ago

if someone is having problem with this line
"When slow pointer reaches beginning of linked list (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."
I have a mathematical proof:
we derived m+k=n(x-2y)
Let x-2y = T . it is pretty obvious that T>=0(also try to imagine when is T=0 and what would be the value of m and k for t=0) now
m+k=T*n
m=T*n - k
m=(T-1)*n +n-k
m=(T-1)*n + (n-k)
so imagine slow at head and fast at k steps away from start of loop
interpretation:
so when slow would have taken m steps implies fast would have taken (n-k) steps + (T-1) cycles of loop.
Hence they will meet at starting point of loop.

• Reply • Share ›

Fbian • 23 days ago

Could somebody please explain, What is the problem if we have an extra integer data VISITED for a node which will be one once it is visited and zero if not , then In the end check if a node's VISITED data is one, If it is one make previous node-> next = NULL

• Reply • Share ›

david ➦ Fbian • 15 days ago

Extra memory needed to store the boolean value. If the listed was say a million nodes long, you would need a million extra bytes for the boolean. Using floyd's, you would only need two extra pointers

1 • Reply • Share ›

Fbian ➦ david • 14 days ago

Thanks david

Reply • Share ›

Shiva • a month ago

the program using method 1 fails if the loop is 1->2->3->4->5->6->7->8->9->3......

the output for the above loop should be 1->2->3->4->5->6->7->8->9 but instead the output that im getting is
1->2->3->4->5->6->7->8

Reply • Share ›

Manish Chauhan • a month ago

```
LinkedList list = new LinkedList();
list.head = new Node(1);
list.head.next = new Node(2);
list.head.next.next = new Node(3);
list.head.next.next.next = new Node(4);
list.head.next.next.next.next = new Node(5);
list.head.next.next.next.next.... = new Node(6);

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

output:

```
Linked List after removing loop :
1 2 3 4 5 6
```

// the output is wrong for method 2.

Reply • Share ›

Ayush Aggarwal • 2 months ago

We can find the loop and the last pointer which is causing the loop (5 in the above example) in O(n) in a very easy way as well
Just store the hexadecimal adress of the pointer as an integer in a map , and check if an adress is repeated twice.
Also we can maintain a previous node to just save the node previous than the node which was repeated and hence finding the pointer that was causing the loop

Reply • Share ›

Mayank Garg • 2 months ago

Better code

Reply • Share ›

Kevin Pandya • 2 months ago

For all the people who are confused why in the method 3 Node *fast = head->next; is there. Here is the explanation:
Since the fast pointer has already taken one step the distance equation is now
2*(m +n*x +k) = m-1 + n*y +k; which simplifies to
m+k+1 = n*(y-2x).
Here one extra is coming. So now when slow pointer has traveled m nodes fast pointer has also traveled m nodes. So total nodes fast node has traveled from the junction is m+k which is one less than the number of nodes in the cycle n. So this method will work. But seriously excellent work by a person who has invented this method

4 ^ | ⌄ • Reply • Share ›

SOURAV SARMA → Kevin Pandya • 23 days ago

Can't understand your explanation.
What is mean by ' Since the fast pointer has already taken one step the distance.. ' , 1. where does is says that the fast pointer has already taken on step, as per the algo, fast pointer will start from K and slow from head.

2. How this method will work if m+k is one less that number of nodes.

Appreciate your clarification.
Thanks.

^ | ⌄ • Reply • Share ›

tyrion • 2 months ago

in method 1, detectandremoveloop function returns an integer, but in main function there is no variable to catch the value.doesn't it create an error.

^ | ⌄ • Reply • Share ›

Avata This comment was deleted.

Aarushi Arora → Guest • 2 months ago

Your code fails for a linked list with even number of nodes, for example : 1->2->3->4->5->6->1.
It returns 1->NULL.
In fact, it will fail for any linked list where the slow and fast pointers do not meet at the last node of the loop.

1 ^ | ⌄ • Reply • Share ›

aamer • 3 months ago

I have got another effecient solution of this problem which first detects the loop using slow and fast references and then the point at which the two pointers meet, I break the linked list by putting the next field of the node as null. Now the problem boils down to finding the merge point of two linked lists that i find using very simple code. In this way, the code is efficiently working. This is the link to the code.

http://ideone.com/Hg1XcY

Reply • Share ›

Premchand • 4 months ago

With more Optimization

```
void detectAndRemoveLoop(Node node) {
Node slow = node;
Node fast = node.next;
Node temp=node;
// Search for loop using slow and fast pointers
while (fast != null && fast.next != null) {
if (slow == fast) {
fast=temp;
fast.next=null;//Remove the Loop
break;
}
slow = slow.next;
temp=fast.next;
fast = fast.next.next;
}

}
```

Reply • Share ›

RocketBoy → Premchand • 3 months ago

This will lead to some of the node that are unreachable

Reply • Share ›

Dhananjai • 4 months ago

the pointer 'head' is not passed by reference,so when we change the pointers 'ptr2' and 'ptr1' are changed ,how does it change the original linked list

Reply • Share ›

Mayank Sinha • 5 months ago

in the above program...does the code related to detect and remove

```
while (fast && fast->next)

{

if (slow == fast)

break;

slow = slow->next;

fast = fast->next->next;
}
```
confirms that we find the loop in single iteration over the loop ???

Reply • Share ›

deepu thomas • 5 months ago

I have another solution for detecting loop with hashMap worst case is O(n).

```
public boolean findLink() {

Map<node<t>, Node<t>> adressMap = new HashMap<node<t>, Node<t>>();
Node<t> temp = mHead;

while (temp != null) {
// check if already exists a next value
Node<t> loop = adressMap.get(temp.next);
if (loop != null) {
System.out.println("Loop in " + temp.data + " and " + loop.data);
return true;
}
adressMap.put(temp, temp.next);
temp = temp.next;
}
System.out.println("No Loop");
return false;
}
```

Reply • Share ›

Dharmik Thakkar • 5 months ago

What about storing the prev_ptr of slow_ptr in loop so that when slow_ptr is equal to fast_ptr, prev_ptr points to node causing the loop and prev_ptr.next can be changed to null.

```
boolean floydCylce(){
Node p1 = head;
Node p2 = head;
Node prev = null;
while(p1!=null && p2!=null && p2.next!=null){
prev = p1;
p1 = p1.next;
p2 = p2.next.next;
if(p1 ==p2){
prev.next = null;
return true;
}
}
return false;
}
```
Is something wrong in this implementation?

Is something wrong in this implementation?

^ | ˅ • Reply • Share ›

Rashmi Mishra ➦ Dharmik Thakkar • 4 months ago

Did you run your code. with your logic the prev will reach til 3 and you will set next pointer of 3 as null and when you will print you will get 1 2 3 rather than 1 2 3 4 5.

^ | ˅ • Reply • Share ›

Dharmik Thakkar ➦ Rashmi Mishra • 4 months ago

Thanks. Only in some cases that would work. Understood my mistake.

^ | ˅ • Reply • Share ›

adi mehra • 5 months ago

Nice Explanation. Please correct this line of the last paragraph- "When slow pointer reaches beginning of linked list (has made m steps)." It should be changed to " When slow pointer reaches beginning of the loop of the linked list (has made m steps). "

^ | ˅ • Reply • Share ›

Vipul Sharma ➦ adi mehra • 5 months ago

yes.exactly

^ | ˅ • Reply • Share ›

Avantika Chabbra • 6 months ago

Solution 2,
Move both pointers at the same pace, they will meet at loop starting node.

I don't understand , why will they meet at the first node?

^ | ˅ • Reply • Share ›

dragon_king ➦ Avantika Chabbra • a month ago

It has been proven mathematically that n divides m+k, so lets assume that the new pointer that has begun its journey from the head pointer has advanced and reached the starting point of the loop(i.e has traveled a distance of m) so now the other pointer that is located at the meet point (k distance away from the starting point of the loop) has also traveled m distance(so this pointer is at a distance of m+k from the starting point of the loop) but since m+k is divisible by n , it means that it has completed (m+k)/n rotations around the loop and has again reached back to the start point of the loop. So , now both the pointers are at the starting point of the loop.

^ | ˅ • Reply • Share ›

Davinder Pal ➦ Avantika Chabbra • 5 months ago

Explanation is given in Method 3

^ | ˅ • Reply • Share ›

pual • 7 months ago
http://code.geeksforgeeks.org/...
1 ^ | v • Reply • Share ›

Sreejith Menon • 7 months ago
I am a little confused about the proof of correctness of method 2 and method 3. Are we assuming there can be no loops in between?
^ | v • Reply • Share ›

Sreejith Menon ➦ Sreejith Menon • 7 months ago
Never mind.. I found the answer in the discussion.
^ | v • Reply • Share ›

Bharath BG • 8 months ago
Can anyone please explain me how this works in a list which has loop like this

1,2,3,4,5,6,7,8,9,8,9,10

There is definitely loop and by the time fast(second) pointer reaches 10 slow(first) pointer will still be at 5?
So no loop detected right?
^ | v • Reply • Share ›

Bhargav Jhaveri ➦ Bharath BG • 7 months ago
The second 8 will not be in picture as the loop is because of the presence of node 9[the second one].
^ | v • Reply • Share ›

Jitender Yadav • 8 months ago
On what basis it has been decided that the second pointer needs to be incremented by 2 ?? I mean why it can't be incremented by 3 or 4 or any other number ??
^ | v • Reply • Share ›

gauri shankar gaur ➦ Jitender Yadav • 7 months ago
http://stackoverflow.com/quest...
1 ^ | v • Reply • Share ›

Shashank Singh • 8 months ago
Can anyone please explain how does moving one pointer single step and other pointer two steps ensures they will meet if loop is present
. I mean how do we prove it?
It would be helpful if you provide any link to an article,video,etc also.
^ | v • Reply • Share ›

Nikhil Kumar Singh ➦ Shashank Singh • 8 months ago

Consider you and your friends are moving in a circle ..Imagine your friend starts 1 sec after you start, and your speed is faster than your friend's speed..Now keep on moving ...You will meet your friend at some point in time ..If you were moving in a straight path you would have never met your friend ...This is how you find that there is a cycle...Simple isn't it ??

• Reply • Share ›

Bharath BG → Nikhil Kumar Singh • 8 months ago

How this works in a list which has loop like this
1,2,3,4,5,6,7,8,9,8,9,10
There is definitely loop and by the time fast(second) pointer reaches 10 slow(first) pointer will still be at 5?
So no loop detected right?

• Reply • Share ›

Nikhil Kumar Singh → Bharath BG • 8 months ago

The loop can only be present at the end. It can't be present in the middle. This is because the linked list has only one next pointer so having a loop at the middle will rule this out. Since the point at 9 will have to have next pointing to both 10 and 8 which is not possible.

Hope you got that right

• Reply • Share ›

Shashank Singh → Nikhil Kumar Singh • 8 months ago

Thanks for the explanation!

• Reply • Share ›

.NetGeek • 9 months ago

C# Code: http://ideone.com/Iknkzr

• Reply • Share ›

nishant sinha • 9 months ago

can anybody explain get pointer to last node part in function remove loop of method 2.tnx in advance

• Reply • Share ›

Mehul Hirpara • 9 months ago

@GeeksforGeeks, following is simple recursive method to detect and remove loop in a linked list.

http://code.geeksforgeeks.org/...

• Reply • Share ›

girlwhoCodes • 9 months ago

girlwhoCodes • 9 months ago

In third solution, why we need fast= head->next and not fast=head just like in other solutions?

• Reply • Share ›

Kevin Pandya girlwhoCodes • 2 months ago

You can check my comment for the explaination

• Reply • Share ›

Yash Desai girlwhoCodes • 7 months ago

so that is cause, you want to find the 2 connected nodes, one. If yo don't use head->next, you will find just one node and to remove it, you need the last node's address.

• Reply • Share ›

Load more comments