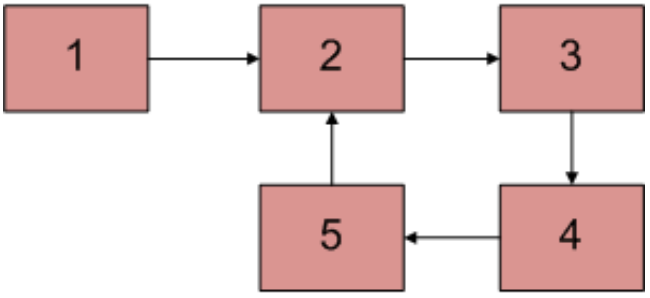
# Write a program function to detect loop in a linked list

Given a linked list, check if the the linked list has loop or not. Below diagram shows a linked list with a loop.




**We strongly recommend that you click here and practice it, before moving on to the solution.**

Following are different ways of doing this

**Use Hashing:**

Traverse the list one by one and keep putting the node addresses in a Hash Table. At any point, if NULL is reached then return false and if next of current node points to any of the previously stored nodes in Hash then return true.

**Mark Visited Nodes:**

This solution requires modifications to basic linked list data structure. Have a visited flag with each node. Traverse the linked list and keep marking visited nodes. If you see a visited node again then there is a loop. This solution works in O(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. Just store the addresses of visited nodes in a hash and if you see an address that already exists in hash then there is a loop.

**Floyd's Cycle-Finding Algorithm:**

This is the fastest method. Traverse linked list using two pointers. Move one pointer by one and other pointer by two. If these pointers meet at some node then there is a loop. If pointers do not meet then

linked list doesn't have loop.

**Implementation of Floyd's Cycle-Finding Algorithm:**

## C/C++

```
// C program to detect loop in a linked list
#include<stdio.h>
#include<stdlib.h>

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

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;
}

int detectloop(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 == fast_p)
        {
           printf("Found Loop");
           return 1;
        }
    }
    return 0;
}

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

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

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

    return 0;
}
```

Run on IDE

## Java

```
// Java program to detect loop in a linked list
class LinkedList
{
    Node head;  // head of list

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

    /* Inserts a new Node at front of the list. */
    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;
    }

    int detectLoop()
    {
        Node slow_p = head, fast_p = head;
        while (slow_p != null && fast_p != null && fast_p.next != null) {
            slow_p = slow_p.next;
            fast_p = fast_p.next.next;
            if (slow_p == fast_p) {
                System.out.println("Found loop");
                return 1;
            }
        }
        return 0;
    }

    /* Drier program to test above functions */
    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;

        llist.detectLoop();
    }
}
/* This code is contributed by Rajat Mishra. */
```

Run on IDE

## Python

```
# Python program to detect loop in the 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

    # 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


    def detectLoop(self):
        slow_p = self.head
        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 == fast_p:
                print "Found Loop"
                return

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

# Create a loop for testing
llist.head.next.next.next.next = llist.head
llist.detectLoop()

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

Run on IDE

Output:

```
Found loop
```

**Time Complexity:** O(n)

**Auxiliary Space:** O(1)

**Asked in: Amazon, DE Shaw, Lybrate, Mahindra Comviva, MAQ Software, Qualcomm, Snapdeal**

**References:**

http://en.wikipedia.org/wiki/Cycle_detection

http://ostermiller.org/find_loop_singly_linked_list.html

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



## Company Wise Coding Practice  Topic Wise Coding Practice

186 Comments Category: Linked Lists Tags: Linked Lists , loop

## 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)

2.4 Average Difficulty : **2.4/5.0**
Based on **108** 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.

**186 Comments** **GeeksforGeeks** 1 **Login**

**Recommend** 8 **Share** **Sort by Newest**

Join the discussion…

**Girish Grover** • 4 days ago

What if i want the element also where the loop is generated.

• Reply • Share ›

**Kunal_sachdeva** • 2 months ago

```
ListNode* Solution::detectCycle(ListNode* A) {
unordered_map<listnode *,int="">hash;
ListNode *tmp=A;
hash[tmp]=1;
tmp=tmp->next;
while(tmp!=NULL)
{
if(hash[tmp]==1)
{
return tmp;
}
else
{
hash[tmp]=1;
tmp=tmp->next;
}

}
tmp=NULL;
return (tmp);

}
```

• Reply • Share ›

**Himanshi_Jain** • 2 months ago

why is it that we are moving them by 1 and 2 steps only..if we move by 2 and 3 steps

then why the time complexity will increase

• Reply • Share ›

**Chaitanya Reddy** ➚ Himanshi_Jain • 2 months ago

u could check this out
http://stackoverflow.com/quest...

1 • Reply • Share ›

**Himanshi_Jain** ➚ Chaitanya Reddy • 2 months ago

thanq..

• Reply • Share ›

**Sarthak Mehra** • 3 months ago

```
boolean loop(node *head)
{

node *back=head;
node *front=head;

while(front && front->next)
{
front=front->next->next;
if(back==front)
return true;
else
back=back->next;

}

return false;
}
```

The above code evaluates result faster.The code mentioned in the content can reach a condition where there is an endless or a very long time taking race between front and back nodes.This is the list where content code fails:
5->4->1->2
^----------^

• Reply • Share ›

**Junaid Alam** • 3 months ago

Simple code using Hash Map please let me know if there is any error...:)
http://code.geeksforgeeks.org/...

• Reply • Share ›

**Prashant Kumar** ➚ Junaid Alam • 2 months ago

add pointers to the set instead,code will be cleaner

• Reply • Share ›

**devendra yadav** • 3 months ago

please write code for hash table mathod...thanx in advance

 • Reply • Share ›

**Ravi Shankar** ➚ devendra yadav • 3 months ago

```
function boolean hasLoop(Node head)
{
HashSet hs = new HashSet();
Node curr = head;
do
{
if (hs.contains(curr))
return true;
hs.add(curr);
} while (curr= curr.next());
return false;
}
```

1  • Reply • Share ›

**devendra yadav** ➚ Ravi Shankar • 3 months ago

thnq shankar

 • Reply • Share ›

**Burninggoku** • 3 months ago

what if we are incr. one pointer by 1 and other by 3...will they meet always.....is there should be like gcd(step1,step2)==1 then its ok???

 • Reply • Share ›

**Anas Ahmed Qazi** • 3 months ago

There may be some test case where floyd's algo will take huge time to converge??? So how is it best algo

 • Reply • Share ›

**Ashish Agarwal** • 5 months ago

Inside the while loop we advance the pointers but do not check if again they are NULL. If you try the same code for NO LOOP condition it gives NPE. Hence we need to check again for not null condition before we compare the data.

```
while (slow_p != null && fast_p != null && fast_p.next != null) {
slow_p = slow_p.next;
fast_p = fast_p.next.next;
if(slow_p != null && fast_p != null) {
if (slow_p == fast_p) {
System.out.println("Found loop");
return 1;
```

return 1;
}
}
}

• Reply • Share ›

**Harsh Vardhan Ladha** • 6 months ago

How can we know the node at which both pointer will meet for the first time ?

I.e., The time when we will see slow_p == fast_p for the first time, how can we know this node before hand? any equation in terms of n and p which will give the meeting point of both pointers.

n is number of linear nodes
p is number of nodes in a loop.

• Reply • Share ›

**Pranshu Kumar** • 7 months ago

while(slow_p && fast_p && fast_p->next )

can be rewritten as

while(fast_p && fast_p->next)

slow pointer is always behind fast pointer and hence will never be null.

8 • Reply • Share ›

**aastha singh** • 8 months ago

http://stackoverflow.com/quest...

• Reply • Share ›

**hazara mullah** • 8 months ago

If you increase the value of the fast pointer by a value higher than two you might skip the value in which both pointers overlap and end up having to iterate more times. By having the fast pointer incremented by two you guarantee that you'll be maximising your efficiency in the case of worst case scenario :) Hope that helps!

you could do that. Instead of moving the fast pointer 2 times faster than the slow pointer, you can move it N times as well. but doing that would increase the time complexity of the algorithm. Read this link: http://stackoverflow.com/quest...

its linear because we can detect the loop in O(length of the loop + position of meeting point of both the pointers ) This is because each time they move, The difference between them is reduced by '1' , Hence even if they move in cycle, the complexity will still be linear.

Draw a list with loop and have 2 pointers starting from head with different speeds as mentioned( fast and slow ptrs ) .you can find the answer. A similar example : Assume a 5km run in a ground and people initially start from the same position, after a while

a 5km run in a ground and people initially start from the same position, after a while, each one may be covering different number of rounds but their physical distance in the ground may be equal. guess How ? I leave it for u

**see more**

2 ▲ ▼ • Reply • Share ›

**Rishi Raj** • 8 months ago

does it matter the order of increment of slow_ptr & fast_ptr in the loop process? can we increment slow_ptr after fast_ptr in the loop

1 ▲ ▼ • Reply • Share ›

**Nikhil Agrawal** ➚ Rishi Raj • 8 months ago

it doesnt matter

▲ ▼ • Reply • Share ›

**Rishi Raj** ➚ Nikhil Agrawal • 7 months ago

thank u for ur reply taking ur time

▲ ▼ • Reply • Share ›

**Mehul Hirpara** • 9 months ago

Go recursive ...

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

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

1 ▲ ▼ • Reply • Share ›

**Kowshika** ➚ Mehul Hirpara • 8 months ago

But you have altered the given linked list which is not desirable

▲ ▼ • Reply • Share ›

**Mehul Hirpara** ➚ Kowshika • 7 months ago

Nope. I am retaining whole list again while coming out of recursion.

The overall idea is to traverse the list until last node is found. If it is found that means no loop otherwise loop is there. Now, if loop is there and if we try to traverse than we may end up with infinite loop. So, what I did is I disconnected each visited node temporary to avoid infinite looping and at the last node we can clearly see that node is pointing itself, which means some loop was there. Now the beauty if recursion is while coming out of from depth of list, we can have all the previous node address available to use and so I could able to reconstruct entire list again.

1 ▲ ▼ • Reply • Share ›

**Shubham Garg** ➚ Mehul Hirpara • 3 months ago

Fantastic work..

• Reply • Share ›

**venky** ➚ Mehul Hirpara • 4 months ago

Nice work! Its working..

• Reply • Share ›

**Jayesh** • 9 months ago

Detailed explanation with Java program.

http://javabypatel.blogspot.in...

• Reply • Share ›

**gizmogaurav** ➚ Jayesh • 8 months ago

very nicely explained

• Reply • Share ›

**Swapnil Pandey** • a year ago

Can anyone explain why the fast pointer is incremented by only 2 and not by 3 or any other number ?

1 • Reply • Share ›

**Ramendu Shandilya** ➚ Swapnil Pandey • a year ago

If you increase the value of the fast pointer by a value higher than two you might skip the value in which both pointers overlap and end up having to iterate more times. By having the fast pointer incremented by two you guarantee that you'll be maximising your efficiency in the case of worst case scenario :) Hope that helps!

2 • Reply • Share ›

**Swapnil Pandey** ➚ Ramendu Shandilya • a year ago

Thanks :)

• Reply • Share ›

**Harish** • a year ago

```
void loopCir(node* head)
{
node *s,*f;

for(s=head,f=head->next;((s!=f)&&(f!=NULL)&&(f->next!=NULL));s=s->next,f=f->next->next);

if((s->next==head)&&(f->next==head))
cout<<"Circular"<<endl; else="" if(s="=" f)="" cout<<"loop"<<endl;="" else="" cout<<"flat"<<endl;="" }="">
```

• Reply • Share ›

**shaunak** • a year ago

what about this logic .....i know its working only for a specific type of linklist but we can modify always the constrution of linklist in the program

1 • Reply • Share ›

**shaunak** • a year ago

```
#include<stdio.h>

int main()

{

int i,n,x,count;

typedef struct node

{

int info;

struct node*link;

}N;

printf("enter the no of nodes\n");

scanf("%d",&n);
```

see more

• Reply • Share ›

**Sandeep Raju** • a year ago

this doesn't work when the last node is pointing to itself!

• Reply • Share ›

**Kartik** ➚ Sandeep Raju • a year ago

Worked for me http://code.geeksforgeeks.org/...

• Reply • Share ›

**pk** • a year ago

Is while(fast!=NULL && fast->next!=NULL) only not sufficient?

• Reply • Share ›

**Rahul Sarkar** ➚ pk • a year ago

suppose slow_p is null.
inside while loop,if we perform this operation => slow_p = slow_p->next;
then it will give null pointer exception.
Therefore we have used the condition slow_p!=null in while loop

Therefore we have used the condition slow_p!=null in while loop.

• Reply • Share ›

**pk** ➚ Rahul Sarkar • a year ago

HI Rahul
Fast!=null and fast->next!=null should take care of everything. Can you give me example where providing only those conditions won't work.
Thanks

• Reply • Share ›

**Rahul Sarkar** ➚ pk • a year ago

in this example,if we dont write fast->next!=null,it will not work

```
void printMiddle(struct node *head)
{
struct node *slow_ptr = head;
struct node *fast_ptr = head;

if (head!=NULL)
{
while (fast_ptr != NULL && fast_ptr->next != NULL)
{
fast_ptr = fast_ptr->next->next;
slow_ptr = slow_ptr->next;
}
printf("The middle element is [%d]\n\n", slow_ptr->data);
}
}
```

• Reply • Share ›

**pk** ➚ Rahul Sarkar • a year ago

I don't think your example is pertinent here to the discussion.
Please give me example which fits context of my question.

• Reply • Share ›

**manish** • a year ago

just valid for this example .what if loop is between 1st and last element

• Reply • Share ›

**Zhengzhi Liu** ➚ manish • a year ago

This will work the same,if loop is between 1st and last element ,slow_ptr and fast_ptr will meet when the slow_ptr move N times(N is the count of element).

• Reply • Share ›

**Kanika Jain** • a year ago

why the space complexity is O(1) ?

• Reply • Share ›

**unknown** ➚ Kanika Jain • a year ago

Space complexity is O(1) when the memory required for an algorithm is independent of its input size. Here, only 2 pointers (fast and slow) are used, which is constant. Hence the space complexity is O(1).

1 • Reply • Share ›

**Kanika Jain** ➚ unknown • a year ago

ty :)

• Reply • Share ›

**Anand** • a year ago

can this problem be solved by comparing addresses of each node instead of data?

• Reply • Share ›

**Mysterious Mind** ➚ Anand • a year ago

Can you please elaborate?

• Reply • Share ›

**makzimus** ➚ Anand • a year ago

Definitely. In fact, the third algorithm does exactly the same

• Reply • Share ›

**Karthick Ram** ➚ Anand • a year ago

Yes you can , but the time complexity would be O(n^2) .

• Reply • Share ›

Load more comments