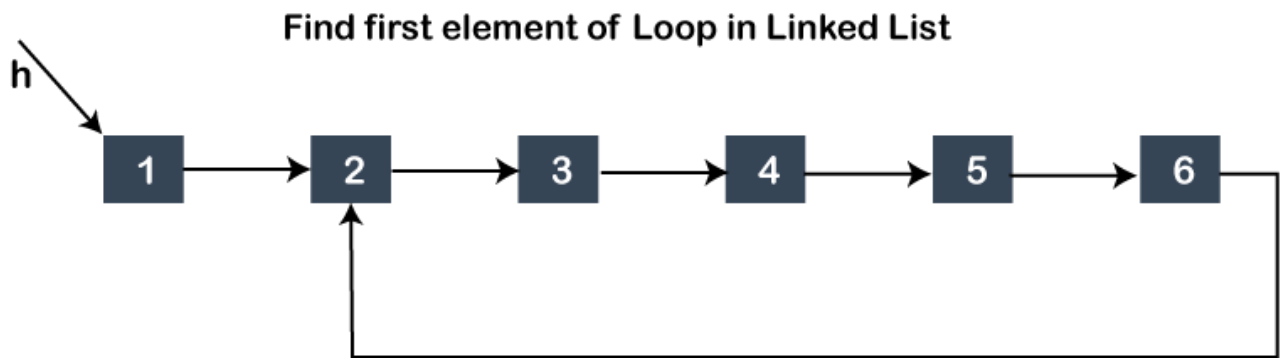


[Home](#)[Data Structure](#)[C](#)[C++](#)[C#](#)[Java](#)[SQL](#)[HTML](#)[CSS](#)[JavaScript](#)[Ajax](#)[↑ SCROLL TO TOP](#)

Detect loop in a Linked list

A loop in a linked list is a condition that occurs when the linked list does not have any end. When the loop exists in the linked list, the last pointer does not point to the Null as observed in the singly linked list or doubly linked list and to the head of the linked list observed in the **circular linked list**. When the loop exists, it points to some other node, also known as the linked list cycle.

Let's understand the loop through an example.



In the above figure, we can observe that the loop exists in the linked list. Here, the problem statement is that we have to detect the node, which is the start of the loop. The solution to solve this problem is:

- First, we detect the loop in the linked list.
- Detect the start node of the loop.

Detecting a loop

First, we will detect a loop in the linked list. To understand this, we will look at the algorithm for detecting a loop.

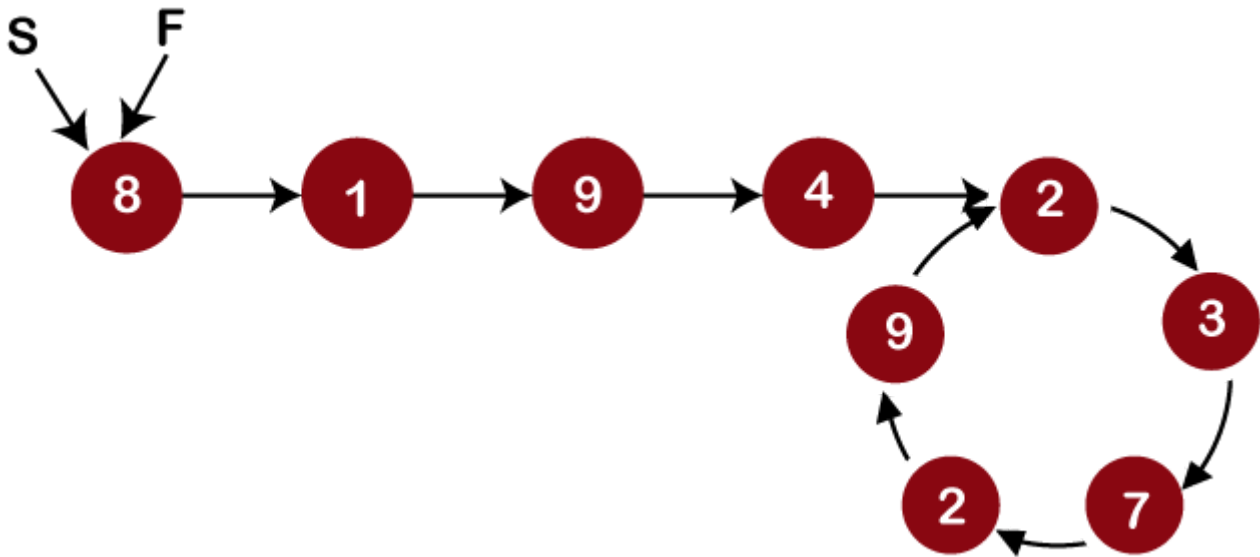
Step 1: First, we will initialize two pointers, i.e., S as a slow pointer and F as a fast pointer. Initially, both the pointers point to the first node in the linked list.

Step 2: Move the 'S' pointer one node at a time while move the 'F' pointer two nodes at a time.

Step 3: If at some point that both the pointers, i.e., 'S' and 'F', point to the same node, then there is a loop in the linked list; otherwise, no loop exists.

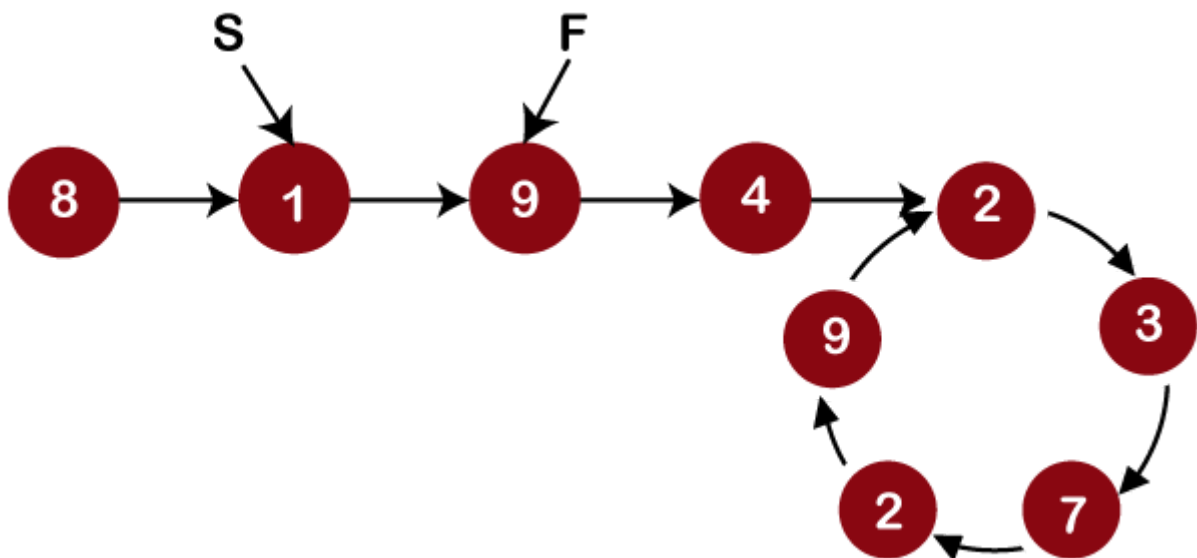
Let's implement the above algorithm for more clarity.

↑ SCROLL TO TOP

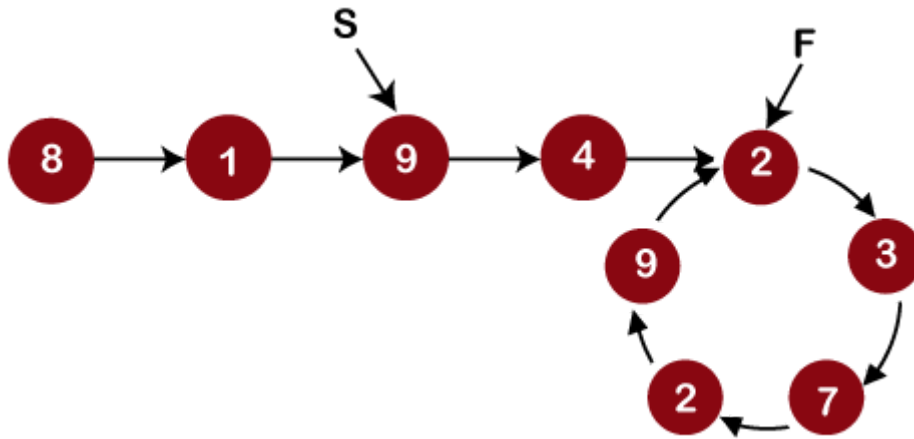


As we can observe in the above figure that both the pointers, i.e., S and F point to the first node. Now, we will move the 'S' pointer by one and the 'F' pointer by two until they meet. If the 'F' pointer reaches the end node means that there is no loop in the linked list.

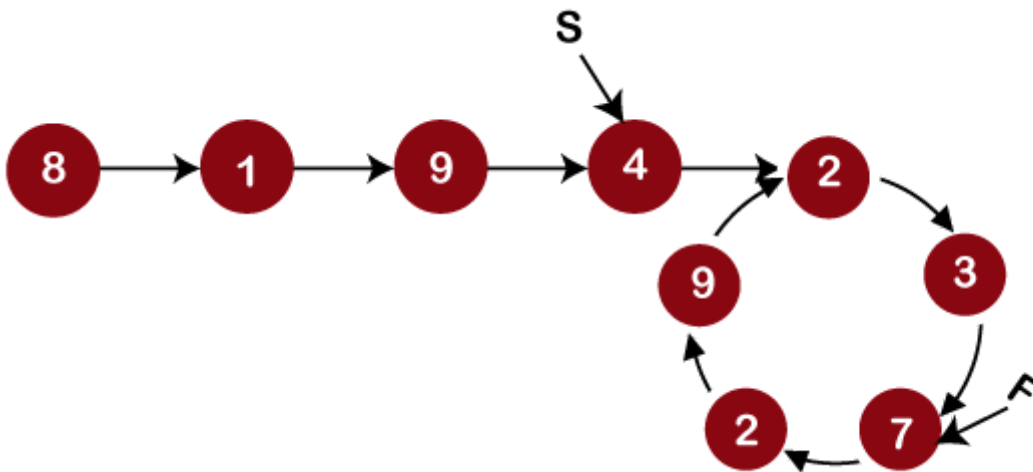
The 'S' pointer moves by one whereas the 'F' pointer moves by two, so 'S' pointer points to node 1, and the 'F' pointer points to node 9 shown as below:



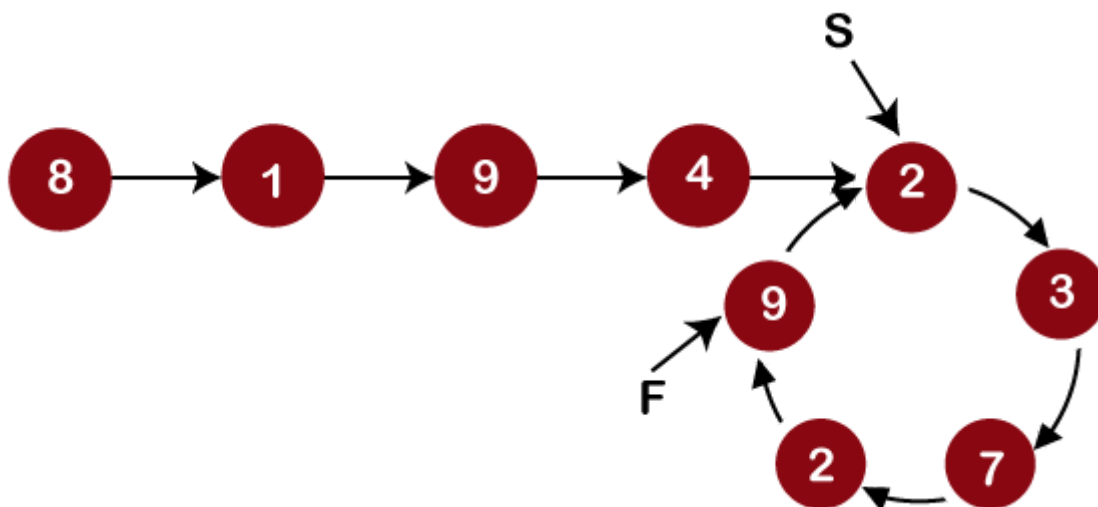
Since both the pointers do not point to the same node and 'F' pointer does not reach the end node so we will again move both the pointers. Now, pointer 'S' will move to the node 9, and pointer 'F' will move to node 2, shown as below:

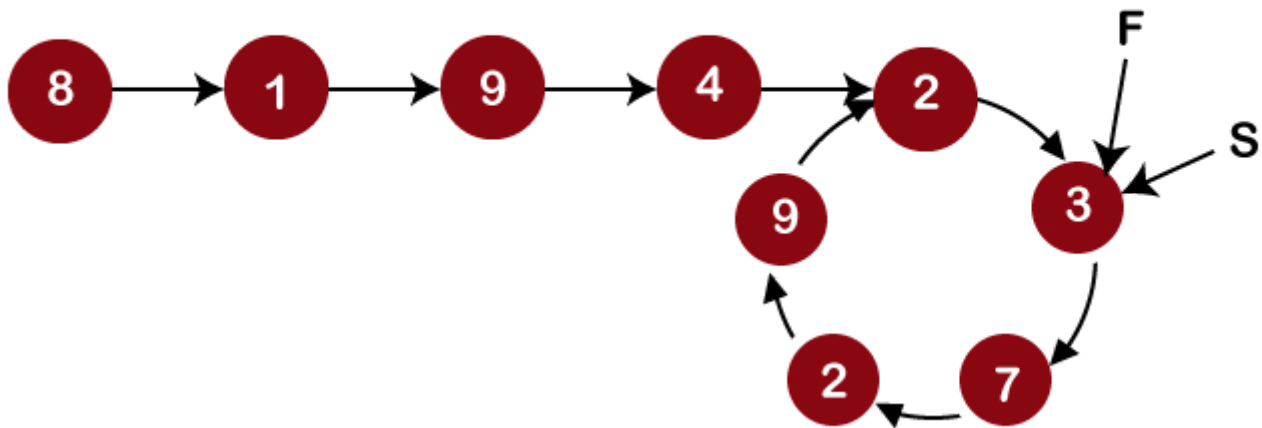


Since both the pointers do not point to the same node, so again, we will increment the pointers. Now, 'S' will point to node 4, and 'F' will point to the node 7 shown as below:



Since both the pointers do not point to the same node, so again, we will increment the pointers. Now, 'S' will point to the node 2, and 'F' will point to node 9 shown as below:





As we can observe in the above figure, both pointers point to the same node, i.e., 3; therefore, the loop exists in the linked list.

Detecting start of the loop

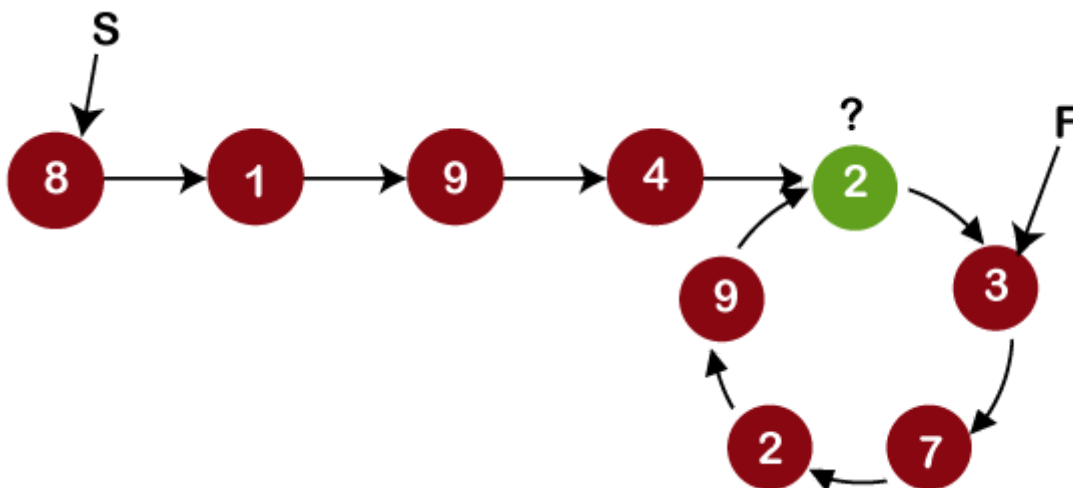
Here, we need to detect the origination of the loop. We will consider the same example which we discussed in detecting the loop. To detect the start of the loop, consider the below algorithm.

Step 1: Move 'S' to the start of the list, but 'F' would remain point to node 3.

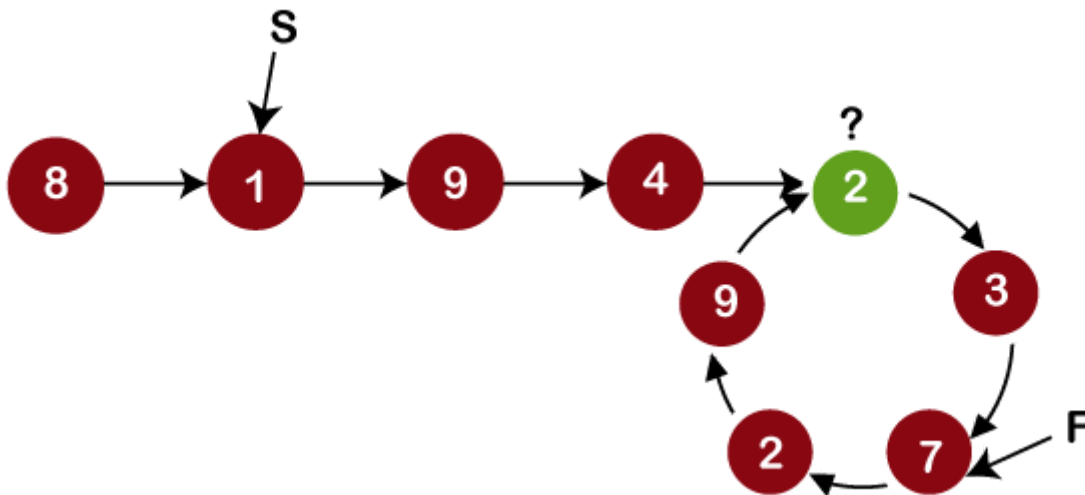
Step 2: Move 'S' and 'F' forward one node at a time until they meet.

Step 3: The node where they meet is the start of the loop.

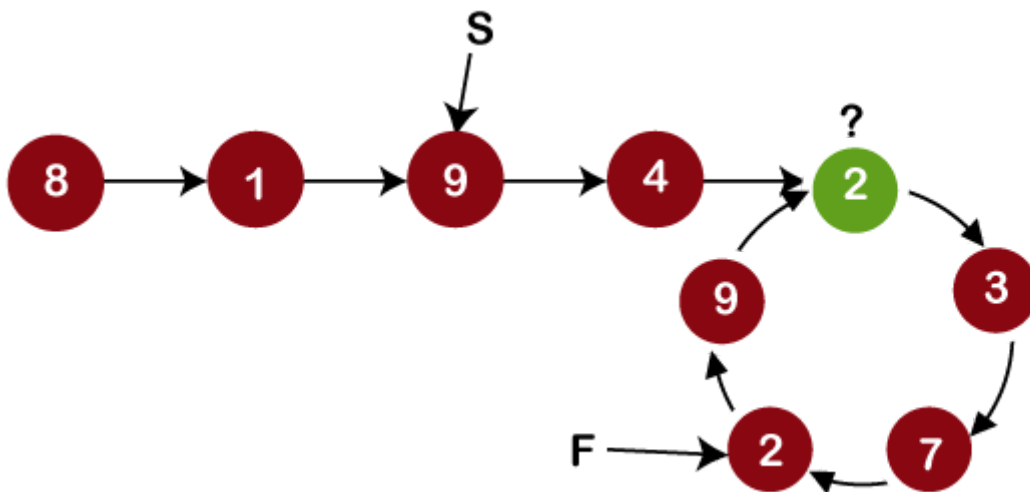
Let's visualize the above algorithm for more clarity.



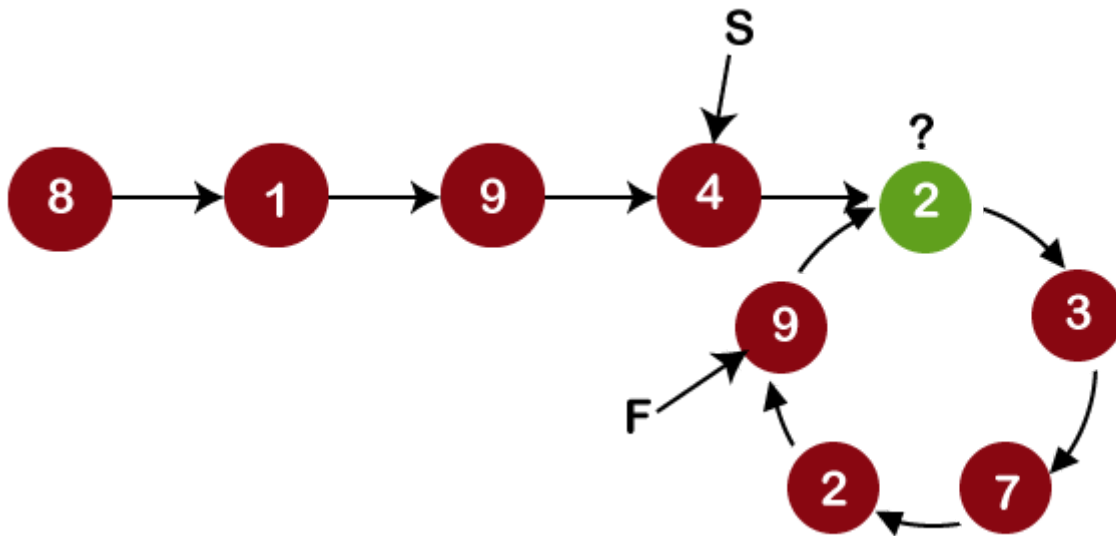
First, we increment the pointer 'S' and 'F' by one; 'S' and 'F' would point to node 1 and node 7, respectively, shown as below:



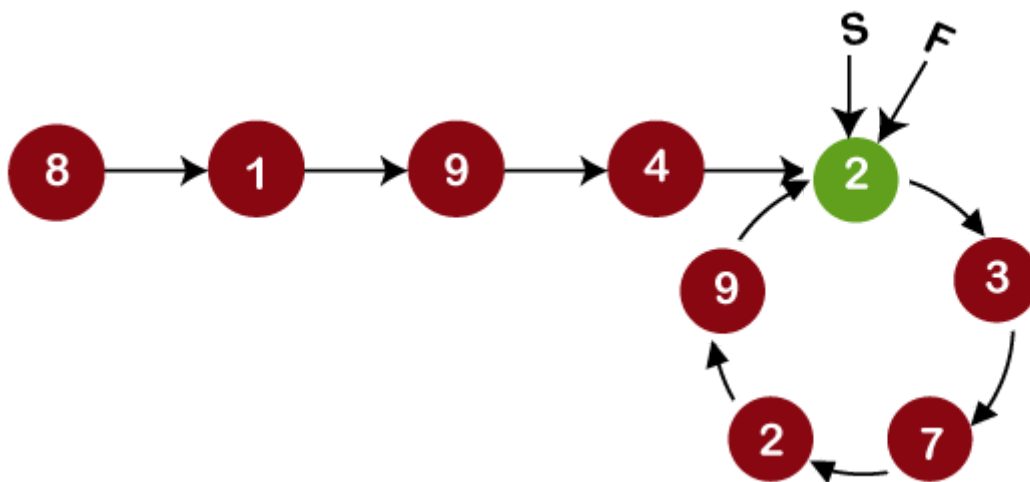
Since both the node are not met, so we again increment the pointers by one node shown as below:



As we can observe in the above figure that the 'S' pointer points to node 9 and 'F' pointer points to the node 2. So again, we increment both the pointers by one node. Now, 'S' would point to the node 4, and 'F' would point to the node 9, shown as below:



As we can observe in the above figure that both the pointers do not point to the same node, so again, we will increment both the pointers by one node. Now, pointer 'S' and 'F' point to node 2 shown as below:



Since both the pointers point to the same node, i.e., 2; therefore, we conclude that the starting node of the loop is node 2.

Why this algorithm works?

Consider 'l' that denotes the length of the loop, which will measure the number in the links.

L: length of the loop

As we can observe in the above figure that there are five links.

Consider 'm' as the distance of the start of the loop from the beginning of the list. In other words, 'm' can be defined as the distance from the starting node to the node from where the

↑ **SCROLL TO TOP** in the above figure, m is 4 because the starting node is 8 and the node

from the loop gets started is 2.

Consider 'k' is the distance of the meeting point of 'S' and 'F' from the start of the loop when they meet for the first time while detecting the loop. In the above linked list, the value of 'k' would be 1 because the node where both fast and slow pointers meet the first time is 3, and the node from where the loop has been started is 2.

When 'S' and 'F' meet for the first time then,

Let's assume that the total distance covered by the slow pointer is **distance_S**, and the total distance covered by the fast pointer is **distance_F**.

$$\text{distance_S} = m + p \cdot l + k$$

where the total distance covered by S is the sum of the distance from the beginning of the list to the start of the loop, the distance covered by the slow pointer in the loop, and the distance from the start of the loop to the node where both the pointers meet.

$$\text{distance_F} = m + q \cdot l + k \quad (q > p \text{ since the speed of 'S' is greater than the speed of 'F'})$$

As we know that when 'S' and 'F' met the first time, then 'F' traverses twice as faster as the 'S' pointer; therefore, the distance covered by 'F' would be two times the distance covered by 'S'. Mathematically, it can be represented as:

$$\text{distance_F} = 2\text{distance_S}$$

The above equation can be written as:

$$m + q \cdot l + k = 2(m + p \cdot l + k)$$

Solving the above equation, we would get:

$$m + k = (q - 2p) \cdot l, \text{ which implies that } m + k \text{ is an integer multiple of } l \text{ (length of the loop).}$$

Implementation of detecting loop in C.

```
int detectloop(struct node *list)
{
    struct node *S = list, *F=list;
    while(S && F && F->next)
    {
        S = S->next;
        F = F->next->next;
    }
    if(S == F)
        return 1;
    else
        return 0;
}
```

↑ SCROLL TO TOP


```
if(F==S)
{
    printf("loop exists");
    return 1;
}
}
return 0;
}
```

In the above code, detectloop() is the name of the function which will detect the loop in the **linked list**. We have passed the list pointer of type struct node pointing to the head node in the linked list. Inside the detectloop() function, we have declared two pointers, i.e., 'S' and 'F' of type struct node and assign the reference of the head node to these pointers. We define the while loop in which we are checking whether the 'S', 'F' and F->next are NULL or not. If they are not Null, then the control will go inside the while loop. Within the while loop, 'S' pointer is incremented by one node and 'F' pointer is incremented by two. If both 'F' and 'S' gets equal; means that the loop exists in the linked list.

[← Prev](#)[Next →](#)

 [For Videos Join Our Youtube Channel: Join Now](#)

Feedback
















- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share








[Learn Latest Tutorials](#)

[↑ SCROLL TO TOP](#)













 Splunk tutorial Splunk	 SPSS tutorial SPSS	 Swagger tutorial Swagger	 T-SQL tutorial Transact-SQL
 Tumblr tutorial Tumblr	 React tutorial ReactJS	 Regex tutorial Regex	 Reinforcement learning tutorial Reinforcement Learning
 R Programming tutorial R Programming	 RxJS tutorial RxJS	 React Native tutorial React Native	 Python Design Patterns Python Design Patterns
 Python Pillow tutorial Python Pillow	 Python Turtle tutorial Python Turtle	 Keras tutorial Keras	

Preparation

















 Aptitude Aptitude	 Logical Reasoning Reasoning	 Verbal Ability Verbal Ability	 Interview Questions Interview Questions
 Company Interview Questions Company Questions			

Trending Technologies

[↑ SCROLL TO TOP](#)

 Artificial Intelligence Tutorial Artificial Intelligence	 AWS Tutorial AWS	 Selenium tutorial Selenium	 Cloud Computing tutorial Cloud Computing
 Hadoop tutorial Hadoop	 ReactJS Tutorial ReactJS	 Data Science Tutorial Data Science	 Angular 7 Tutorial Angular 7
 Blockchain Tutorial Blockchain	 Git Tutorial Git	 Machine Learning Tutorial Machine Learning	 DevOps Tutorial DevOps

B.Tech / MCA

 DBMS tutorial DBMS	 Data Structures tutorial Data Structures	 DAA tutorial DAA	 Operating System tutorial Operating System
 Computer Network tutorial Computer Network	 Compiler Design tutorial Compiler Design	 Computer Organization and Architecture Computer Organization	 Discrete Mathematics Tutorial Discrete Mathematics
 Ethical Hacking Tutorial Ethical Hacking	 Computer Graphics Tutorial Computer Graphics	 Software Engineering Tutorial Software Engineering	 html tutorial Web Technology
 Cyber Security tutorial Cyber Security	 Automata Tutorial Automata	 C Language tutorial C Programming	 C++ tutorial C++

↑ SCROLL TO TOP

