Detect loop using floyd cycle detection

Detect loop in a linked list using Floyd's Cycle-Finding Algorithm:

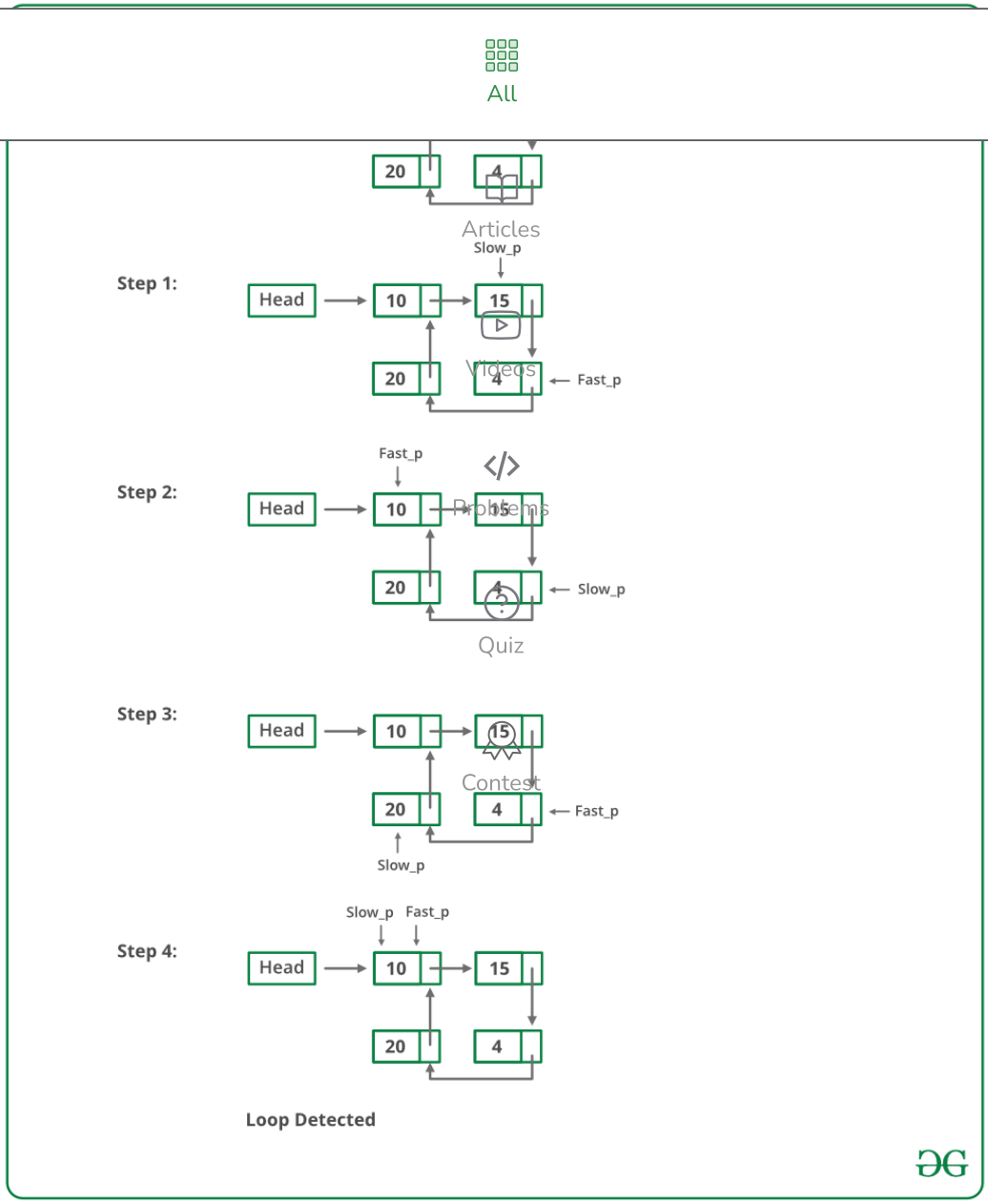
> *This algorithm is used to find a loop in a linked list. It uses two pointers one moving twice as fast as the other one. The faster one is called the faster pointer and the other one is called the slow pointer.*

Follow the steps below to solve the problem:

- Traverse linked list using two pointers.
- Move one pointer(slow_p) by one and another pointer(fast_p) by two.
- If these pointers meet at the same node then there is a loop. If pointers do not meet then the linked list doesn't have a loop.

Illustration:

The below image shows how the detect loop function works in the code:



Implementation of Floyd's Cycle-Finding Algorithm:

C++ Java

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

import java.io.*;

class LinkedList {
    Node head; // head of list
```

```
    Node next;

        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;
}

void detectLoop()
{
    Node slow_p = head, fast_p = head;
    int flag = 0;
    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) {
            flag = 1;
            break;
        }
    }
    if (flag == 1)
        System.out.println("Loop Found");
    else
        System.out.println("No Loop");
```

```
    {
        llist.push(4);
        llist.push(15);
        llist.push(10);

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

        llist.detectLoop();
    }
}
```


Output

```
Loop Found
```

Time complexity: O(N), Only one traversal of the loop is needed.
Auxiliary Space: O(1).