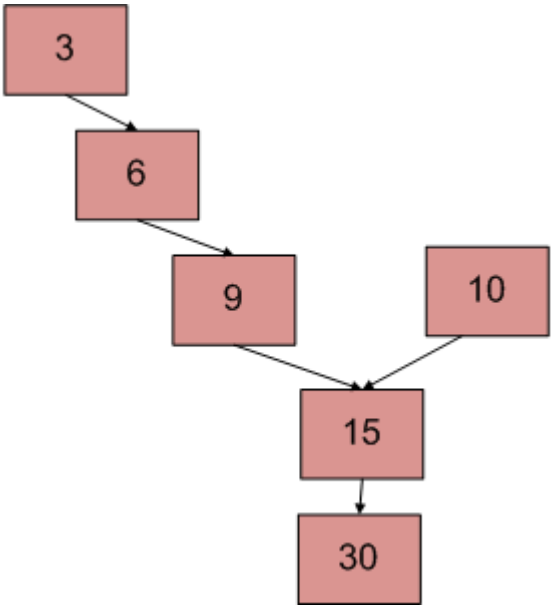
Intersection of two linked list

There are two singly linked lists in a system. By some programming error, the end node of one of the linked lists got linked to the second list, forming an inverted Y-shaped list. Write a program to get the point where two linked lists merge.

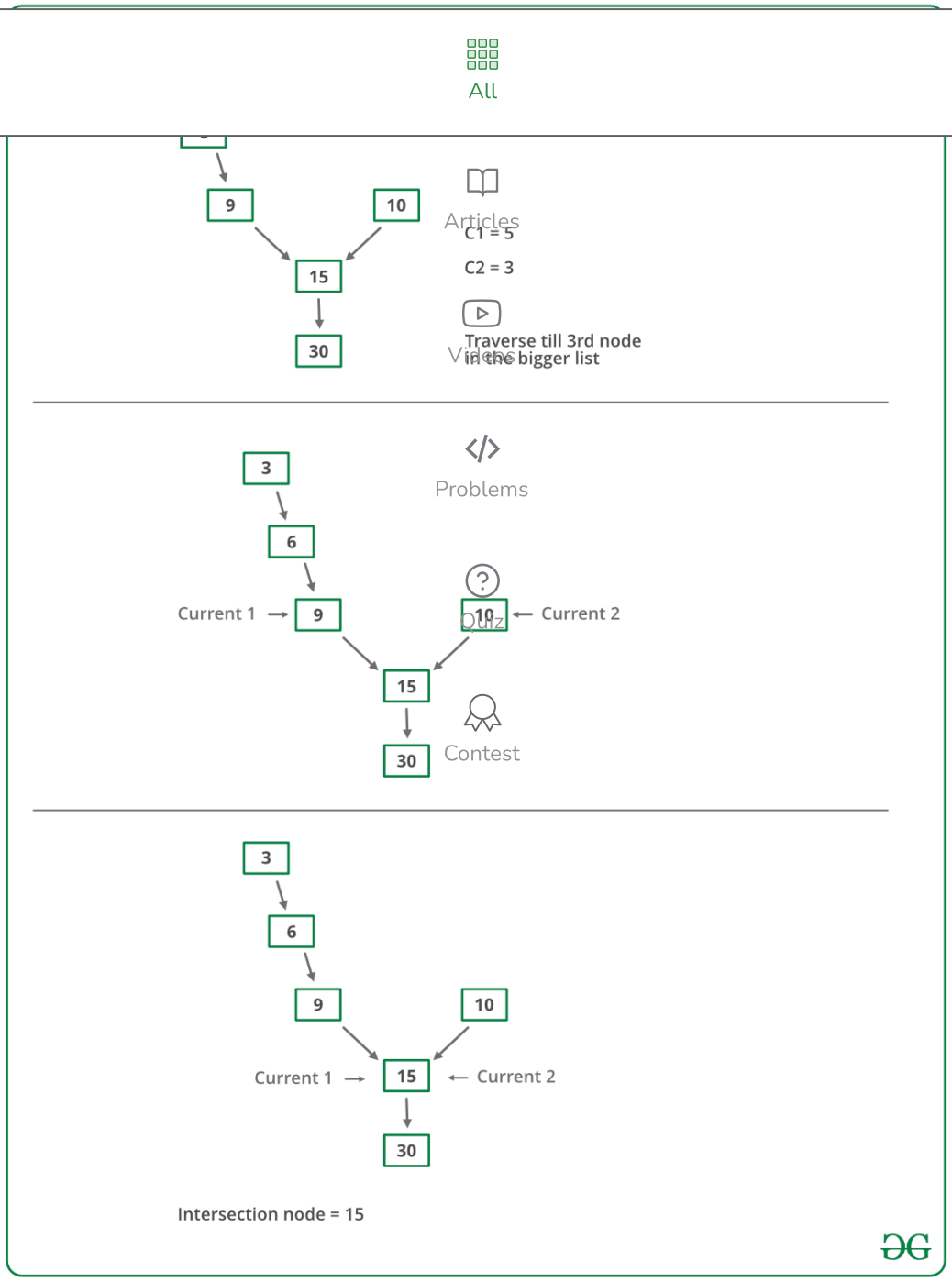


The above diagram shows an example with two linked lists having 15 as intersection points.

Method : Using the difference in node counts

- Get the count of the nodes in the first list, let the count be c1.
- Get the count of the nodes in the second list, let the count be c2.
- Get the difference of counts **d = abs(c1 – c2)**
- Now traverse the bigger list from the first node to d nodes so that from here onwards both the lists have an equal no of nodes
- Then we can traverse both lists in parallel till we come across a common node. (Note that getting a common node is done by comparing the address of the nodes)

Below image is a dry run of the above approach:



Below is the implementation of the above approach :

C++ Java

```
// Java program to get intersection point of two linked list

class LinkedList {

    static Node head1, head2;
```

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

/*function to get the intersection point of two linked
lists head1 and head2 */
int getNode()
{
    int c1 = getCount(head1);
    int c2 = getCount(head2);
    int d;

    if (c1 > c2) {
        d = c1 - c2;
        return _getIntesectionNode(d, head1, head2);
    }
    else {
        d = c2 - c1;
        return _getIntesectionNode(d, head2, head1);
    }
}

/* function to get the intersection point of two linked
lists head1 and head2 where head1 has d more nodes than
head2 */
int _getIntesectionNode(int d, Node node1, Node node2)
{
    int i;
    Node current1 = node1;
    Node current2 = node2;
    for (i = 0; i < d; i++) {
        if (current1 == null) {
            return -1;
        }
```

```
        while (current1 != null && current2 != null) {

            current1 = current1.next;
            current2 = current2.next;
        }

        return -1;
    }

    /*Takes head pointer of the linked list and
    returns the count of nodes in the list */
    int getCount(Node node)
    {
        Node current = node;
        int count = 0;

        while (current != null) {
            count++;
            current = current.next;
        }

        return count;
    }

    public static void main(String[] args)
    {
        LinkedList list = new LinkedList();

        // creating first linked list
        list.head1 = new Node(3);
        list.head1.next = new Node(6);
        list.head1.next.next = new Node(9);
        list.head1.next.next.next = new Node(15);
        list.head1.next.next.next.next = new Node(30);

        // creating second linked list
        list.head2 = new Node(10);
        list.head2.next = new Node(15);
        list.head2.next.next = new Node(30);
```


Track Progress

91 of **132** Complete. (69%)

```
    }
```

Output

```
The node of intersection is 15
```

Method : Use Hashing

Basically, we need to find a common node of two linked lists. So we hash all nodes of the first list and then check the second list.

1) Create an empty hash set.

2) Traverse the first linked list and insert all nodes' addresses in the hash set.

3) Traverse the second list. For every node check if it is present in the hash set. If we find a node in the hash set, return the node.

C++ | Java

```
// Java program to get intersection point of two linked list
import java.util.*;
class Node {
    int data;
    Node next;
    Node(int d)
    {
        data = d;
        next = null;
    }
}
class LinkedListIntersect {
    public static void main(String[] args)
    {
        // list 1
        Node n1 = new Node(1);
        n1.next = new Node(2);
        n1.next.next = new Node(3);
        n1.next.next.next = new Node(4);
        n1.next.next.next.next = new Node(5);
```

```
        // list 2
        n2.next.next.next = n1.next.next.next;
        Print(n1);
        Print(n2);
        System.out.println(MegeNode(n1, n2).data);
    }

    // function to print the list
    public static void Print(Node n)
    {
        Node cur = n;
        while (cur != null) {
            System.out.print(cur.data + " ");
            cur = cur.next;
        }
        System.out.println();
    }

    // function to find the intersection of two node
    public static Node MegeNode(Node n1, Node n2)
    {
        // define hashset
        HashSet<Node> hs = new HashSet<Node>();
        while (n1 != null) {
            hs.add(n1);
            n1 = n1.next;
        }
        while (n2 != null) {
            if (hs.contains(n2)) {
                return n2;
            }
        }
        return null;
    }
```

This method required O(n) additional space and is not very efficient if one list is large.