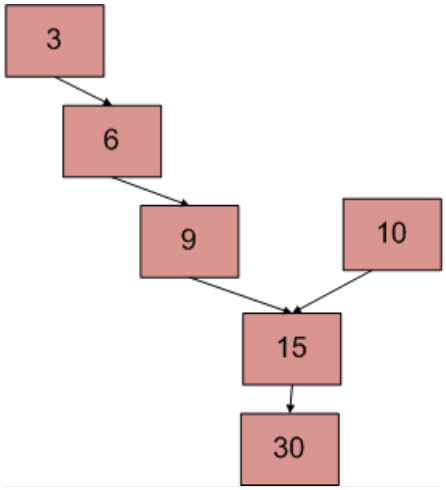
Write a function to get the intersection point of two Linked Lists.

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



Above diagram shows an example with two linked list having 15 as intersection point.

Method 1(Simply use two loops)

Use 2 nested for loops. Outer loop will be for each node of the 1st list and inner loop will be for 2nd list. In the inner loop, check if any of nodes of 2nd list is same as the current node of first linked list. Time complexity of this method will be O(mn) where m and n are the number of nodes in two lists.

Method 2 (Mark Visited Nodes)

This solution requires modifications to basic linked list data structure. Have a visited flag with each node. Traverse the first linked list and keep marking visited nodes. Now traverse second linked list, If you see a visited node again then there is an intersection point, return the intersecting node. This solution works in O(m+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. Traverse the first linked list and store the addresses of visited nodes in a hash. Now traverse the second linked list and if you see an address that already exists in hash then return the intersecting node.

Method 3(Using difference of node counts)

1) Get count of the nodes in first list, let count be c1.
2) Get count of the nodes in second list, let count be c2.
3) Get the difference of counts d = abs(c1 – c2)
4) Now traverse the bigger list from the first node till d nodes so that from here onwards both the lists have equal no of nodes.
5) Then we can traverse both the 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)

C

```
#include<stdio.h>
#include<stdlib.h>

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

/* Function to get the counts of node in a linked list */
int getCount(struct node* head);
```

```
/* 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, struct node* head1, struct node* head2);

/* function to get the intersection point of two linked
   lists head1 and head2 */
int getIntesectionNode(struct node* head1, struct node* head2)
{
  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, struct node* head1, struct node* head2)
{
  int i;
  struct node* current1 = head1;
  struct node* current2 = head2;

  for(i = 0; i < d; i++)
  {
    if(current1 == NULL)
    {  return -1; }
    current1 = current1->next;
  }

  while(current1 !=  NULL && current2 != NULL)
  {
    if(current1 == current2)
      return current1->data;
    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(struct node* head)
{
  struct node* current = head;
  int count = 0;

  while (current != NULL)
  {
    count++;
    current = current->next;
  }

  return count;
}

/* IGNORE THE BELOW LINES OF CODE. THESE LINES
   ARE JUST TO QUICKLY TEST THE ABOVE FUNCTION */
int main()
{
```

```
{
  /*
    Create two linked lists

    1st 3->6->9->15->30
    2nd 10->15->30

    15 is the intersection point
  */

  struct node* newNode;
  struct node* head1 =
            (struct node*) malloc(sizeof(struct node));
  head1->data  = 10;

  struct node* head2 =
            (struct node*) malloc(sizeof(struct node));
  head2->data  = 3;

  newNode = (struct node*) malloc (sizeof(struct node));
  newNode->data = 6;
  head2->next = newNode;

  newNode = (struct node*) malloc (sizeof(struct node));
  newNode->data = 9;
  head2->next->next = newNode;

  newNode = (struct node*) malloc (sizeof(struct node));
  newNode->data = 15;
  head1->next = newNode;
  head2->next->next->next  = newNode;

  newNode = (struct node*) malloc (sizeof(struct node));
  newNode->data = 30;
  head1->next->next= newNode;

  head1->next->next->next = NULL;

  printf("\n The node of intersection is %d \n",
          getIntesectionNode(head1, head2));

  getchar();
}
```

Java

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

class LinkedList {

    static Node head1, head2;

    static class Node {

        int data;
        Node next;

        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;
            }
            current1 = current1.next;
        }
        while (current1 != null && current2 != null) {
            if (current1.data == current2.data) {
                return current1.data;
            }
            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(15);
        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);

        System.out.println("The node of intersection is " + list.getNode());

    }
}

// This code has been contributed by Mayank Jaiswal
```

Time Complexity: O(m+n)
Auxiliary Space: O(1)

Method 4(Make circle in first list)

Thanks to Saravanan Man for providing below solution.

1. Traverse the first linked list(count the elements) and make a circular linked list. (Remember last node so that we can break the circle later on).
2. Now view the problem as find the loop in the second linked list. So the problem is solved.
3. Since we already know the length of the loop(size of first linked list) we can traverse those many number of nodes in second list, and then start another pointer from the beginning of second list. we have to traverse until they are equal, and that is the required intersection point.
4. remove the circle from the linked list.

Time Complexity: O(m+n)
Auxiliary Space: O(1)

Method 5 (Reverse the first list and make equations)

Thanks to Saravanan Mani for providing this method.

```
1) Let X be the length of the first linked list until intersection point.
   Let Y be the length of the second linked list until the intersection point.
   Let Z be the length of the linked list from intersection point to End of
   the linked list including the intersection node.
   We Have
           X + Z = C1;
           Y + Z = C2;
2) Reverse first linked list.
3) Traverse Second linked list. Let C3 be the length of second list - 1.
     Now we have
        X + Y = C3
     We have 3 linear equations. By solving them, we get
       X = (C1 + C3 – C2)/2;
       Y = (C2 + C3 – C1)/2;
       Z = (C1 + C2 – C3)/2;
      WE GOT THE INTERSECTION POINT.
4)  Reverse first linked list.
```

Advantage: No Comparison of pointers.
Disadvantage : Modifying linked list(Reversing list).

Time complexity: O(m+n)
Auxiliary Space: O(1)

Method 6 (Traverse both lists and compare addresses of last nodes) This method is only to detect if there is an intersection point or not. (Thanks to NeoTheSaviour for suggesting this)

```
1) Traverse the list 1, store the last node address
2) Traverse the list 2, store the last node address.
3) If nodes stored in 1 and 2 are same then they are intersecting.
```

Time complexity of this method is O(m+n) and used Auxiliary space is O(1)

Method 7 (Use Hashing)

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

1) Create an empty hash table such that node address is used as key and a binary value present/absent is used as value.
2) Traverse the first linked list and insert all nodes' addresses in hash table.
3) Traverse the second list. For every node check if it is present in hash table. If we find a node in hash table, return the node.