## Question:

Given pointers to the head nodes of 2 linked lists that merge together at some point, find the node where the two lists merge. The merge point is where both lists point to the same node, i.e. they reference the same memory location. It is guaranteed that the two head nodes will be different, and neither will be NULL. If the lists share a common node, return that node's  ***data*** value.

**Note:** After the merge point, both lists will share the same node pointers.

**Example**

In the diagram below, the two lists converge at Node x:

[List #1] a--->b--->c

\

x--->y--->z--->NULL

/

[List #2] p--->q

**Function Description**

Complete the *findMergeNode* function in the editor below.

*findMergeNode* has the following parameters:

* *SinglyLinkedListNode pointer head1:* a reference to the head of the first list
* *SinglyLinkedListNode pointer head2:* a reference to the head of the second list

**Returns**

* *int:* the data value of the node where the lists merge

## Solution:

|  |
| --- |
| /\*  Insert Node at the end of a linked list  head pointer input could be NULL as well for empty list  Node is defined as  class Node {  int data;  Node next;  }  \*/  int FindMergeNode(Node headA, Node headB) {  // Complete this function  // Do not write the main method.    int countA=0;  int countB=0;    Node tempA=headA;  Node tempB=headB;  while(tempA!=null)  {  countA++;  tempA=tempA.next;  }      while(tempB!=null)  {  countB++;  tempB=tempB.next;  }  int diff=0;  if(countA>countB)  diff=countA-countB;  else  diff=countB-countA;  tempA=headA;  tempB=headB;  if(countA>countB)  {  while(diff >0)  {tempA=tempA.next;  diff--;}    }  else  {while(diff >0)  { tempB=tempB.next;  diff--;}      }    while(tempA!=null && tempB!=null)  {    tempA=tempA.next;  tempB=tempB.next;  if(tempA==tempB)  return tempA.data;    }  return 0;  } |