```
/*Author: Bochen (mddboc@foxmail.com)
Last Modified: Tue Apr 10 22:28:44 CST 2018*/

/* Write a program to find the node at which the intersection of two singly linked
lists begins.


For example, the following two linked lists:

A:          a1 → a2
                   ↘
                     c1 → c2 → c3
                   ↗
B:     b1 → b2 → b3
begin to intersect at node c1.


Notes:

If the two linked lists have no intersection at all, return null.
The linked lists must retain their original structure after the function returns.
You may assume there are no cycles anywhere in the entire linked structure.
Your code should preferably run in O(n) time and use only O(1) memory. */


import java.util.*;
import java.lang.Math;
import java.lang.System;
import java.lang.Integer;


public class Main {

    public static void main(String[] args) throws ArithmeticException {

        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(2);
        root.left.left = new TreeNode(3);
        root.left.right = new TreeNode(4);
        root.right.left = new TreeNode(4);
        root.right.right = new TreeNode(3);

        boolean result = new Solution().isSymmetric(root);

        System.out.println(result);
    }

}


class ListNode {
    int val;
    ListNode next;

    ListNode(int x) {
        val = x;
    }
}


class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int x) {
        val = x;
    }
}


class Solution {
```

```
    
    //version 2: better one
    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        
        int aLength = 0, bLength = 0;
        ListNode aPointer = headA, bPointer = headB;
        
        while ( aPointer != null ) {
            aLength++;
            aPointer = aPointer.next;
        }
        while ( bPointer != null ) {
            bLength++;
            bPointer = bPointer.next;
        }
        
        if ( aLength < bLength ) {
            return getIntersectionNodeHelper(headA, headB, bLength - aLength);
        } else {
            return getIntersectionNodeHelper(headB, headA, aLength - bLength);
        }
        
    }
    
    private ListNode getIntersectionNodeHelper(ListNode headA, ListNode headB, int deltaLength) {
        
        for (int i = deltaLength; i > 0; i-- ) {
            
            headB = headB.next;
        }
        
        while (true) {
            
            if ( headA == headB ) {
                return headA;
            }
            
            headA = headA.next;
            headB = headB.next;
        }
    }
    
    
    //version 1
    /* public ListNode getIntersectionNode(ListNode headA, ListNode headB) {

        ListNode nodeA = headA, nodeB = headB;

        while (true) {

            if (nodeA == nodeB) {
                return nodeA;
            }

            nodeA = (nodeA == null) ? headB : nodeA.next;
            nodeB = (nodeB == null) ? headA : nodeB.next;
        }
    } */
}
```