

Problem 160: Intersection of Two Linked Lists

Problem Information

Difficulty: Easy

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given the heads of two singly linked-lists

headA

and

headB

, return

the node at which the two lists intersect

. If the two linked lists have no intersection at all, return

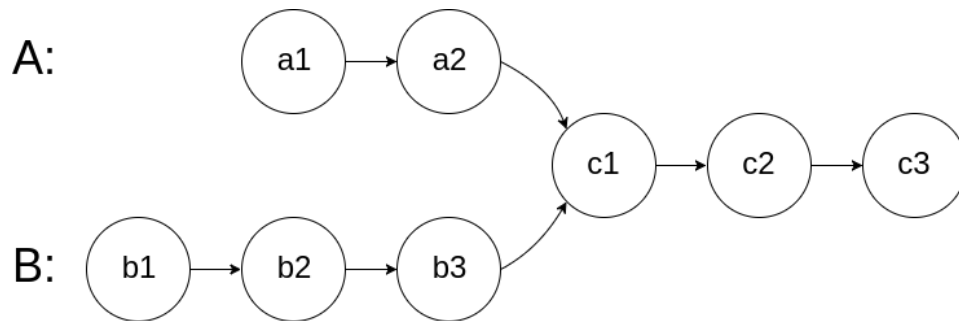
null

.

For example, the following two linked lists begin to intersect at node

c1

:



The test cases are generated such that there are no cycles anywhere in the entire linked structure.

Note

that the linked lists must

retain their original structure

after the function returns.

Custom Judge:

The inputs to the

judge

are given as follows (your program is

not

given these inputs):

`intersectVal`

- The value of the node where the intersection occurs. This is

0

if there is no intersected node.

listA

- The first linked list.

listB

- The second linked list.

skipA

- The number of nodes to skip ahead in

listA

(starting from the head) to get to the intersected node.

skipB

- The number of nodes to skip ahead in

listB

(starting from the head) to get to the intersected node.

The judge will then create the linked structure based on these inputs and pass the two heads,

headA

and

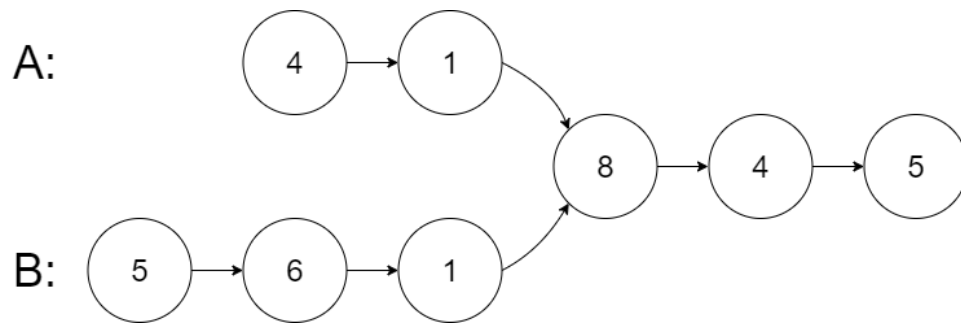
headB

to your program. If you correctly return the intersected node, then your solution will be

accepted

.

Example 1:



Input:

intersectVal = 8, listA = [4,1,8,4,5], listB = [5,6,1,8,4,5], skipA = 2, skipB = 3

Output:

Intersected at '8'

Explanation:

The intersected node's value is 8 (note that this must not be 0 if the two lists intersect). From the head of A, it reads as [4,1,8,4,5]. From the head of B, it reads as [5,6,1,8,4,5]. There are 2 nodes before the intersected node in A; There are 3 nodes before the intersected node in B. - Note that the intersected node's value is not 1 because the nodes with value 1 in A and B (2

nd

node in A and 3

rd

node in B) are different node references. In other words, they point to two different locations in memory, while the nodes with value 8 in A and B (3

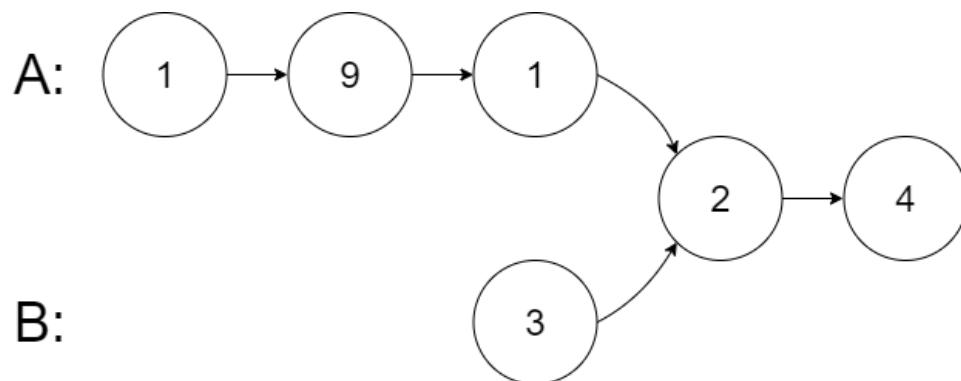
rd

node in A and 4

th

node in B) point to the same location in memory.

Example 2:



Input:

intersectVal = 2, listA = [1,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1

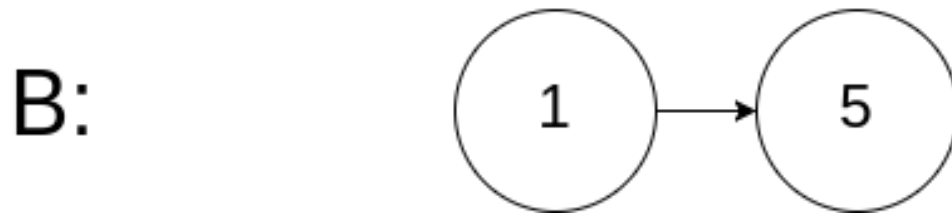
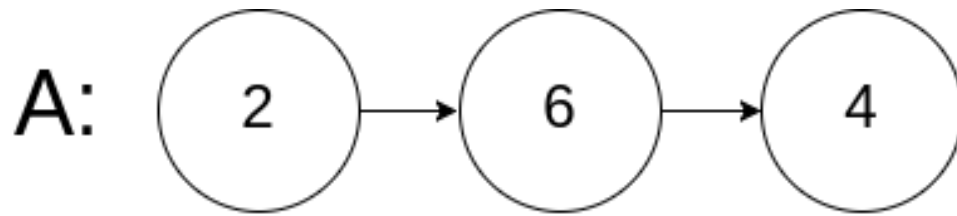
Output:

Intersected at '2'

Explanation:

The intersected node's value is 2 (note that this must not be 0 if the two lists intersect). From the head of A, it reads as [1,9,1,2,4]. From the head of B, it reads as [3,2,4]. There are 3 nodes before the intersected node in A; There are 1 node before the intersected node in B.

Example 3:



Input:

`intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2`

Output:

No intersection

Explanation:

From the head of A, it reads as [2,6,4]. From the head of B, it reads as [1,5]. Since the two lists do not intersect, `intersectVal` must be 0, while `skipA` and `skipB` can be arbitrary values.

Explanation: The two lists do not intersect, so return null.

Constraints:

The number of nodes of

`listA`

is in the

m

.

The number of nodes of

listB

is in the

n

.

$1 \leq m, n \leq 3 * 10$

4

$1 \leq \text{Node.val} \leq 10$

5

$0 \leq \text{skipA} \leq m$

$0 \leq \text{skipB} \leq n$

intersectVal

is

0

if

listA

and

listB

do not intersect.

`intersectVal == listA[skipA] == listB[skipB]`

if

listA

and

listB

intersect.

Follow up:

Could you write a solution that runs in

$O(m + n)$

time and use only

$O(1)$

memory?

Code Snippets

C++:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *   int val;
 *   ListNode *next;
 *   ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
```

```
}  
};
```

Java:

```
/**  
 * Definition for singly-linked list.  
 * public class ListNode {  
 *   int val;  
 *   ListNode next;  
 *   ListNode(int x) {  
 *     val = x;  
 *     next = null;  
 *   }  
 * }  
 */  
public class Solution {  
    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {  
  
    }  
}
```

Python3:

```
# Definition for singly-linked list.  
# class ListNode:  
#   def __init__(self, x):  
#     self.val = x  
#     self.next = None  
  
class Solution:  
    def getIntersectionNode(self, headA: ListNode, headB: ListNode) ->  
        Optional[ListNode]:
```

Python:

```
# Definition for singly-linked list.  
# class ListNode(object):  
#   def __init__(self, x):  
#     self.val = x  
#     self.next = None
```

```

class Solution(object):
def getIntersectionNode(self, headA, headB):
    """
    :type head1, head1: ListNode
    :rtype: ListNode
    """

```

JavaScript:

```

/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *   this.val = val;
 *   this.next = null;
 * }
 */

/**
 * @param {ListNode} headA
 * @param {ListNode} headB
 * @return {ListNode}
 */
var getIntersectionNode = function(headA, headB) {

};

```

TypeScript:

```

/**
 * Definition for singly-linked list.
 * class ListNode {
 *   val: number
 *   next: ListNode | null
 *   constructor(val?: number, next?: ListNode | null) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.next = (next===undefined ? null : next)
 *   }
 * }
 */

function getIntersectionNode(headA: ListNode | null, headB: ListNode | null):

```

```
ListNode | null {

};
```

C#:

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     public int val;
 *     public ListNode next;
 *     public ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode GetIntersectionNode(ListNode headA, ListNode headB) {

    }
}
```

C:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode *getIntersectionNode(struct ListNode *headA, struct ListNode
*headB) {

}
```

Go:

```
/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
```

```

func getIntersectionNode(headA, headB *ListNode) *ListNode {

}

```

Kotlin:

```

/**
 * Example:
 * var li = ListNode(5)
 * var v = li.`val`
 * Definition for singly-linked list.
 * class ListNode(var `val`: Int) {
 *     var next: ListNode? = null
 * }
 */

class Solution {
    fun getIntersectionNode(headA: ListNode?, headB: ListNode?): ListNode? {

    }
}

```

Swift:

```

/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     public var val: Int
 *     public var next: ListNode?
 *     public init(_ val: Int) {
 *         self.val = val
 *         self.next = nil
 *     }
 * }
 */

class Solution {
    func getIntersectionNode(_ headA: ListNode?, _ headB: ListNode?) -> ListNode?
    {

    }
}

```

Ruby:

```
# Definition for singly-linked list.
# class ListNode
# attr_accessor :val, :next
# def initialize(val)
# @val = val
# @next = nil
# end
# end

# @param {ListNode} headA
# @param {ListNode} headB
# @return {ListNode}
def getIntersectionNode(headA, headB)

end
```

PHP:

```
/**
 * Definition for a singly-linked list.
 * class ListNode {
 * public $val = 0;
 * public $next = null;
 * function __construct($val) { $this->val = $val; }
 * }
 */

class Solution {
/**
 * @param ListNode $headA
 * @param ListNode $headB
 * @return ListNode
 */
function getIntersectionNode($headA, $headB) {

}

}
```

Scala:

```

/**
 * Definition for singly-linked list.
 * class ListNode(var _x: Int = 0) {
 *   var next: ListNode = null
 *   var x: Int = _x
 * }
 */

object Solution {
  def getIntersectionNode(headA: ListNode, headB: ListNode): ListNode = {

  }
}

```

Solutions

C++ Solution:

```

/*
 * Problem: Intersection of Two Linked Lists
 * Difficulty: Easy
 * Tags: array, hash, linked_list
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *   int val;
 *   ListNode *next;
 *   ListNode(int x) : val(x), next(NULL) {}
 * };
 */

class Solution {
public:
  ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {

  }
}

```

```
};
```

Java Solution:

```
/**
 * Problem: Intersection of Two Linked Lists
 * Difficulty: Easy
 * Tags: array, hash, linked_list
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {

    }
}
```

Python3 Solution:

```
"""
Problem: Intersection of Two Linked Lists
Difficulty: Easy
Tags: array, hash, linked_list

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""
```

```

"""

# Definition for singly-linked list.
# class ListNode:
# def __init__(self, x):
# self.val = x
# self.next = None

class Solution:
def getIntersectionNode(self, headA: ListNode, headB: ListNode) ->
Optional[ListNode]:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

# Definition for singly-linked list.
# class ListNode(object):
# def __init__(self, x):
# self.val = x
# self.next = None

class Solution(object):
def getIntersectionNode(self, headA, headB):
    """
    :type head1, head1: ListNode
    :rtype: ListNode
    """

```

JavaScript Solution:

```

/**
 * Problem: Intersection of Two Linked Lists
 * Difficulty: Easy
 * Tags: array, hash, linked_list
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

```

```

/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *   this.val = val;
 *   this.next = null;
 * }
 */

/**
 * @param {ListNode} headA
 * @param {ListNode} headB
 * @return {ListNode}
 */
var getIntersectionNode = function(headA, headB) {

};

```

TypeScript Solution:

```

/**
 * Problem: Intersection of Two Linked Lists
 * Difficulty: Easy
 * Tags: array, hash, linked_list
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

/**
 * Definition for singly-linked list.
 * class ListNode {
 *   val: number
 *   next: ListNode | null
 *   constructor(val?: number, next?: ListNode | null) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.next = (next===undefined ? null : next)
 *   }
 * }
 */

```

```
function getIntersectionNode(headA: ListNode | null, headB: ListNode | null):
  ListNode | null {

};
```

C# Solution:

```
/*
 * Problem: Intersection of Two Linked Lists
 * Difficulty: Easy
 * Tags: array, hash, linked_list
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

/**
 * Definition for singly-linked list.
 * public class ListNode {
 * public int val;
 * public ListNode next;
 * public ListNode(int x) { val = x; }
 * }
 */
public class Solution {
  public ListNode GetIntersectionNode(ListNode headA, ListNode headB) {

  }
}
```

C Solution:

```
/*
 * Problem: Intersection of Two Linked Lists
 * Difficulty: Easy
 * Tags: array, hash, linked_list
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
```

```

*/

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *   int val;
 *   struct ListNode *next;
 * };
 */
struct ListNode *getIntersectionNode(struct ListNode *headA, struct ListNode
*headB) {

}

```

Go Solution:

```

// Problem: Intersection of Two Linked Lists
// Difficulty: Easy
// Tags: array, hash, linked_list
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *   Val int
 *   Next *ListNode
 * }
 */
func getIntersectionNode(headA, headB *ListNode) *ListNode {

}

```

Kotlin Solution:

```

/**
 * Example:
 * var li = ListNode(5)
 * var v = li.`val`

```

```

* Definition for singly-linked list.
* class ListNode(var `val`: Int) {
*   var next: ListNode? = null
* }
*/

class Solution {
fun getIntersectionNode(headA:ListNode?, headB:ListNode?):ListNode? {

}
}

```

Swift Solution:

```

/**
* Definition for singly-linked list.
* public class ListNode {
*   public var val: Int
*   public var next: ListNode?
*   public init(_ val: Int) {
*     self.val = val
*     self.next = nil
*   }
* }
*/

class Solution {
func getIntersectionNode(_ headA: ListNode?, _ headB: ListNode?) -> ListNode?
{

}
}

```

Ruby Solution:

```

# Definition for singly-linked list.
# class ListNode
#   attr_accessor :val, :next
#   def initialize(val)
#     @val = val
#     @next = nil
#   end
# end

```

```

# end
# end

# @param {ListNode} headA
# @param {ListNode} headB
# @return {ListNode}
def getIntersectionNode(headA, headB)

end

```

PHP Solution:

```

/**
 * Definition for a singly-linked list.
 * class ListNode {
 * public $val = 0;
 * public $next = null;
 * function __construct($val) { $this->val = $val; }
 * }
 */

class Solution {
/**
 * @param ListNode $headA
 * @param ListNode $headB
 * @return ListNode
 */
function getIntersectionNode($headA, $headB) {

}

}

```

Scala Solution:

```

/**
 * Definition for singly-linked list.
 * class ListNode(var _x: Int = 0) {
 * var next: ListNode = null
 * var x: Int = _x
 * }
 */

```

```
object Solution {  
  def getIntersectionNode(headA: ListNode, headB: ListNode): ListNode = {  
  
  }  
}
```