Problem 142: Linked List Cycle II

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given the

head

of a linked list, return

the node where the cycle begins. If there is no cycle, return

null

.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the

next

pointer. Internally,

pos

is used to denote the index of the node that tail's

next

pointer is connected to (

0-indexed

). It is

-1

if there is no cycle.

Note that

pos

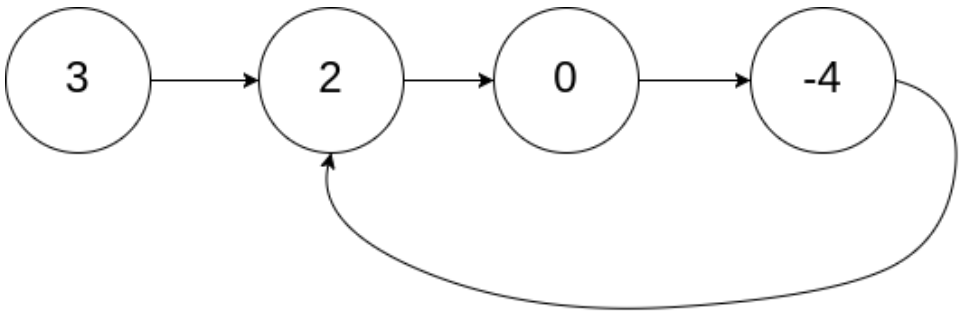
is not passed as a parameter

.

Do not modify

the linked list.

Example 1:



Input:

head = [3,2,0,-4], pos = 1

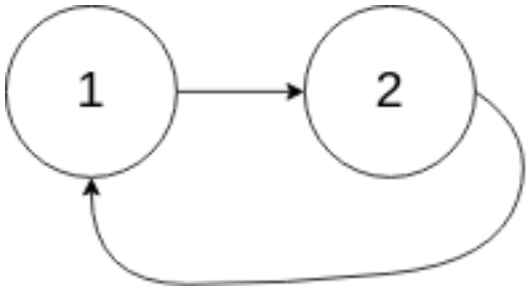
Output:

tail connects to node index 1

Explanation:

There is a cycle in the linked list, where tail connects to the second node.

Example 2:



Input:

head = [1,2], pos = 0

Output:

tail connects to node index 0

Explanation:

There is a cycle in the linked list, where tail connects to the first node.

Example 3:

Input:

head = [1], pos = -1

Output:

no cycle

Explanation:

There is no cycle in the linked list.

Constraints:

The number of the nodes in the list is in the range

[0, 10

4

]

.

-10

5

<= Node.val <= 10

5

pos

is

-1

or a

valid index

in the linked-list.

Follow up:

Can you solve it using

O(1)

(i.e. constant) 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 *detectCycle(ListNode *head) {

}
};
```

Java:

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

}
}
```

Python3:

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

class Solution:
def detectCycle(self, head: Optional[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 detectCycle(self, head):
"""
:type head: ListNode
:rtype: ListNode
"""
```

JavaScript:

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

/**
* @param {ListNode} head
* @return {ListNode}
*/
var detectCycle = function(head) {

};
```

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 detectCycle(head: 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;
* next = null;
* }
* }
*/
public class Solution {
public ListNode DetectCycle(ListNode head) {

}
}
```

C:

```
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
```

```
* struct ListNode *next;
* };
*/
struct ListNode *detectCycle(struct ListNode *head) {

}
```

Go:

```
/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func detectCycle(head *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 detectCycle(head: 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 detectCycle(_ head: 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} head
# @return {ListNode}
def detectCycle(head)

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 $head
* @return ListNode
*/
function detectCycle($head) {

}
}
```

Scala:

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

object Solution {
def detectCycle(head: ListNode): ListNode = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Linked List Cycle II
* Difficulty: Medium
* 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 *detectCycle(ListNode *head) {

}
};
```

Java Solution:

```
/**
* Problem: Linked List Cycle II
* Difficulty: Medium
* 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 {
* int val;
* ListNode next;
* ListNode(int x) {
* val = x;
* next = null;
* }
* }
*/
public class Solution {
```

```
public ListNode detectCycle(ListNode head) {

}
}
```

Python3 Solution:

```
"""
Problem: Linked List Cycle II
Difficulty: Medium
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 detectCycle(self, head: Optional[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 detectCycle(self, head):
"""
:type head: ListNode
:rtype: ListNode
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Linked List Cycle II
* Difficulty: Medium
* 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} head
* @return {ListNode}
*/
var detectCycle = function(head) {

};
```

TypeScript Solution:

```
/**
* Problem: Linked List Cycle II
* Difficulty: Medium
* 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 detectCycle(head: ListNode | null): ListNode | null {

};
```

C# Solution:

```
/*
* Problem: Linked List Cycle II
* Difficulty: Medium
* 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;
* next = null;
* }
* }
*/
public class Solution {
```

```
public ListNode DetectCycle(ListNode head) {

}
}
```

C Solution:

```
/*
* Problem: Linked List Cycle II
* Difficulty: Medium
* 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 *detectCycle(struct ListNode *head) {

}
```

Go Solution:

```
// Problem: Linked List Cycle II
// Difficulty: Medium
// 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 detectCycle(head *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 detectCycle(head: 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 detectCycle(_ head: ListNode?) -> ListNode? {

}
}
```

Ruby Solution:

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

# @param {ListNode} head
# @return {ListNode}
def detectCycle(head)

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 $head
* @return ListNode
*/
function detectCycle($head) {

}
```

```
}
```

Scala Solution:

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

object Solution {
def detectCycle(head: ListNode): ListNode = {

}
}
```