

Problem 141: Linked List Cycle

Problem Information

Difficulty: Easy

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given

head

, the head of a linked list, determine if the linked list has a cycle in it.

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.

Note that

pos

is not passed as a parameter

.

Return

true

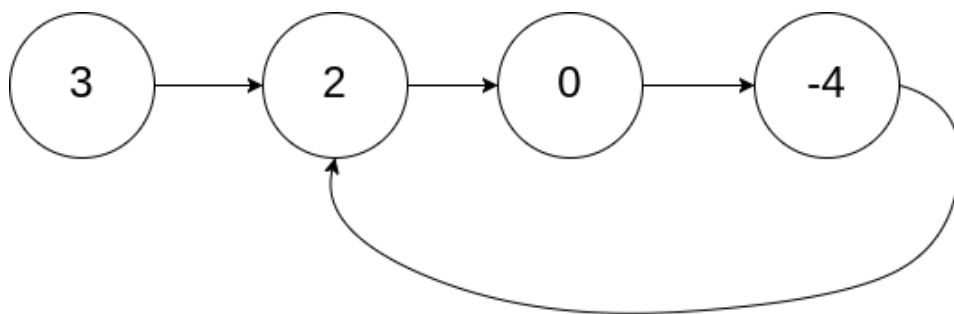
if there is a cycle in the linked list

. Otherwise, return

false

.

Example 1:



Input:

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

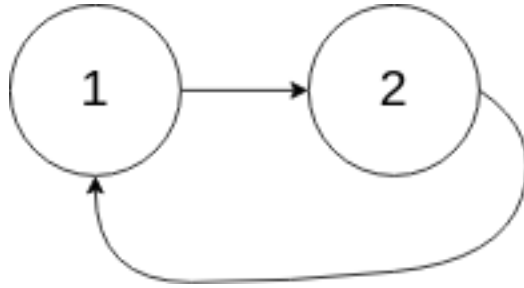
Output:

true

Explanation:

There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

Example 2:



Input:

head = [1,2], pos = 0

Output:

true

Explanation:

There is a cycle in the linked list, where the tail connects to the 0th node.

Example 3:



Input:

head = [1], pos = -1

Output:

false

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

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

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:
    bool hasCycle(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 boolean hasCycle(ListNode head) {

    }
}
```

Python3:

```

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

class Solution:
    def hasCycle(self, head: Optional[ListNode]) -> bool:

```

Python:

```

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

class Solution(object):
    def hasCycle(self, head):
        """
        :type head: ListNode
        :rtype: bool
        """

```

JavaScript:

```

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

/**
 * @param {ListNode} head
 * @return {boolean}
 */
var hasCycle = 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 hasCycle(head: ListNode | null): boolean {

};

```

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 bool HasCycle(ListNode head) {

    }
}

```

C:

```

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

```

```

*/
bool hasCycle(struct ListNode *head) {

}

```

Go:

```

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
func hasCycle(head *ListNode) bool {

}

```

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 hasCycle(head: ListNode?): Boolean {

}

}

```

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 hasCycle(_ head: ListNode?) -> Bool {

}
}

```

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 {Boolean}
def hasCycle(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 $head
 * @return Boolean
 */
function hasCycle($head) {

}
}

```

Scala:

```

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

object Solution {
  def hasCycle(head: ListNode): Boolean = {

  }
}

```

Solutions

C++ Solution:

```

/*
 * Problem: Linked List Cycle
 * 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) {
 * // TODO: Implement optimized solution
 * return 0;
 * }
 * };
 */
class Solution {
public:
    bool hasCycle(ListNode *head) {

    }
};

```

Java Solution:

```

/**
 * Problem: Linked List Cycle
 * 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 {
 *   int val;
 *   ListNode next;
 *   ListNode(int x) {
 *     val = x;
 *     next = null;
 *   }
 * }
 */

```

```

public class Solution {
    public boolean hasCycle(ListNode head) {

    }
}

```

Python3 Solution:

```

"""
Problem: Linked List Cycle
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 hasCycle(self, head: Optional[ListNode]) -> bool:
        # 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 hasCycle(self, head):
        """
        :type head: ListNode

```

```
:rtype: bool
"""
```

JavaScript Solution:

```
/**
 * Problem: Linked List Cycle
 * 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} head
 * @return {boolean}
 */
var hasCycle = function(head) {

};
```

TypeScript Solution:

```
/**
 * Problem: Linked List Cycle
 * 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 hasCycle(head: ListNode | null): boolean {

};

```

C# Solution:

```

/*
 * Problem: Linked List Cycle
 * 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;
 *     next = null;
 *   }
 * }
 */

```

```

public class Solution {
    public bool HasCycle(ListNode head) {

    }
}

```

C Solution:

```

/*
 * Problem: Linked List Cycle
 * 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;
 * };
 */
bool hasCycle(struct ListNode *head) {

}

```

Go Solution:

```

// Problem: Linked List Cycle
// 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 hasCycle(head *ListNode) bool {

}

```

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 hasCycle(head: ListNode?): Boolean {

    }
}

```

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 hasCycle(_ head: ListNode?) -> Bool {

}

}

```

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 {Boolean}
def hasCycle(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 $head
 * @return Boolean
 */
function hasCycle($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 hasCycle(head: ListNode): Boolean = {  
  
  }  
}
```