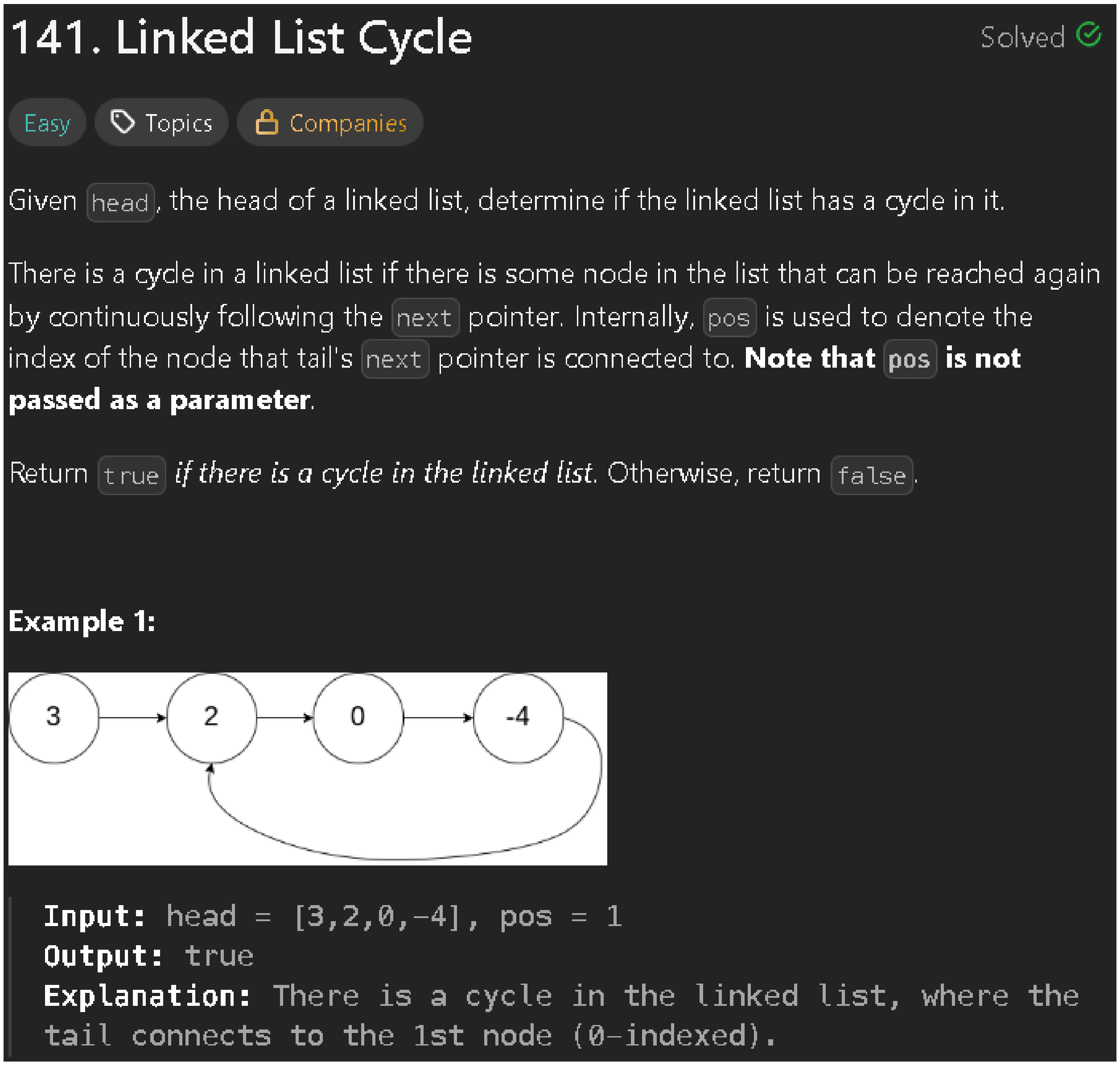


141. Linked List Cycle

Solved

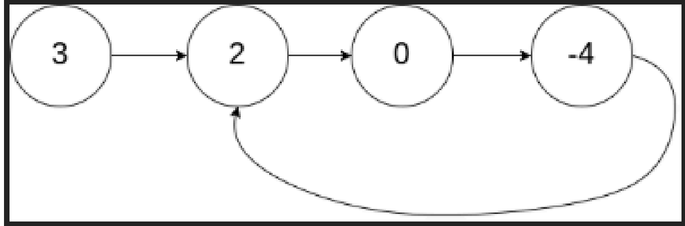
Easy Topics Companies

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

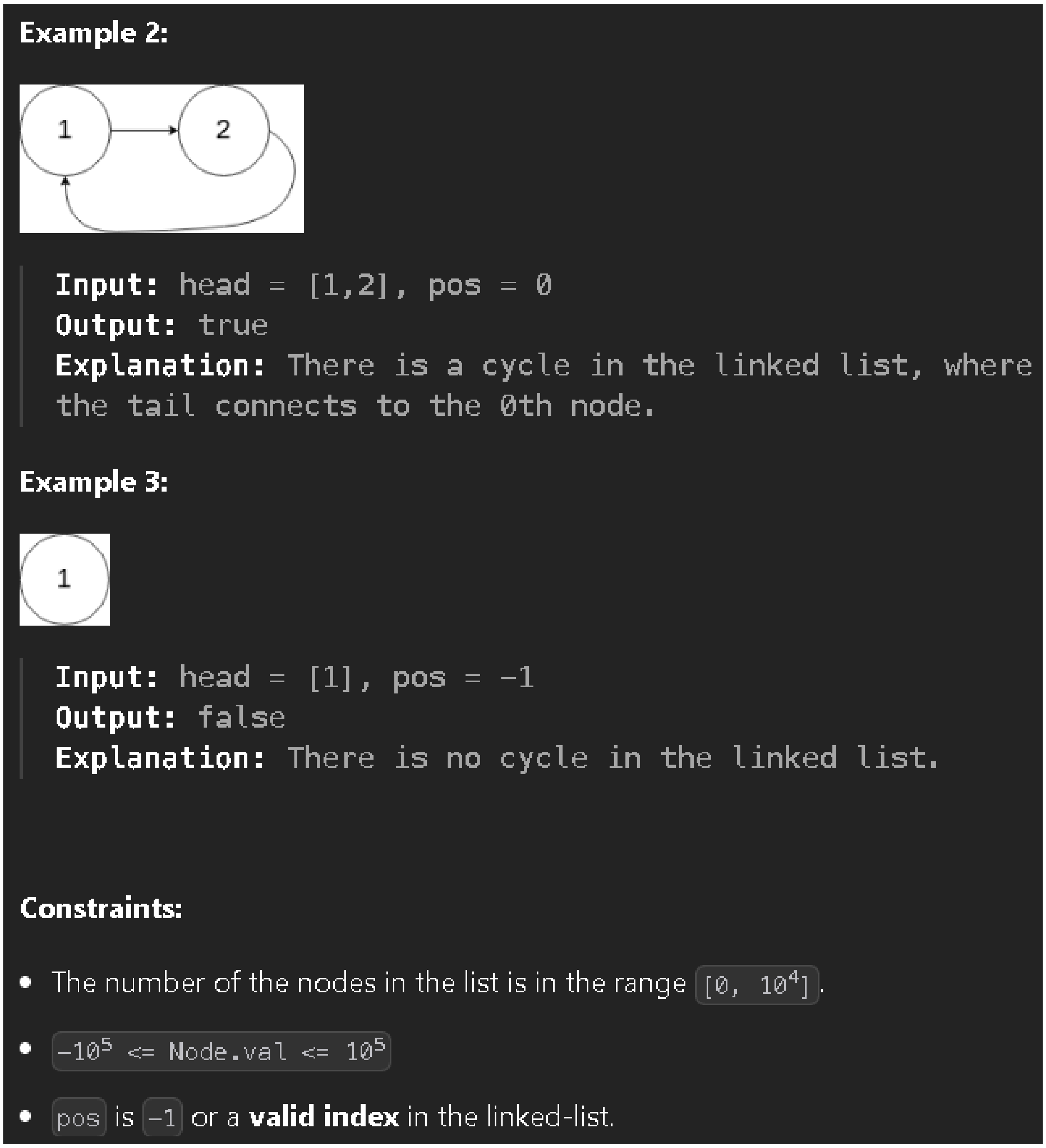
There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter**.

Return `true` *if there is a cycle in the linked list*. Otherwise, return `false`.

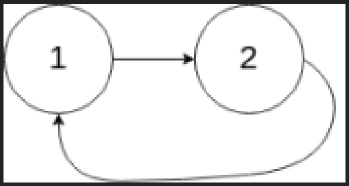
Example 1:



```
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).
```

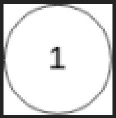


Example 2:



```
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 <= `Node.val` <= 10^5
- `pos` is `-1` or a **valid index** in the linked-list.

Python:

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

from typing import Optional
```

```
class Solution:
    def hasCycle(self, head: Optional[ListNode]) -> bool:
        if not head or not head.next:
            return False

        slow = head
        fast = head.next

        while slow != fast:
            if not fast or not fast.next:
                return False
            slow = slow.next
            fast = fast.next.next

        return True
```

JavaScript:

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

/**
 * @param {ListNode} head
 * @return {boolean}
 */
var hasCycle = function(head) {
    if (!head || !head.next) return false; // empty list or single node (no cycle)

    let slow = head;
    let fast = head;

    while (fast !== null && fast.next !== null) {
        slow = slow.next;         // move slow by 1
        fast = fast.next.next;    // move fast by 2

        if (slow === fast) {      // cycle detected
            return true;
        }
    }

    return false; // reached end → no cycle
};
```

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) {
        if (head == null || head.next == null) {
            return false; // no nodes or just one node -> no cycle
        }

        ListNode slow = head;
        ListNode fast = head;

        while (fast != null && fast.next != null) {
            slow = slow.next;        // move by 1 step
            fast = fast.next.next;   // move by 2 steps

            if (slow == fast) {
                return true; // cycle detected
            }
        }

        return false; // reached end, no cycle
    }
}
```