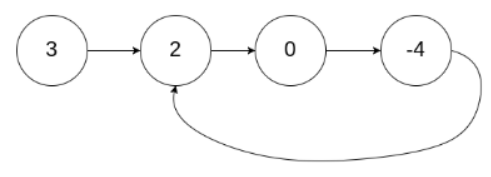
<https://leetcode.com/problems/linked-list-cycle/>

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:**

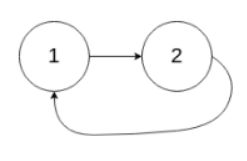


Input: head = [3,2,0,-4], pos = 1

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, 104].
* -105 <= Node.val <= 105
* pos is -1 or a **valid index** in the linked-list.

**Follow up:** Can you solve it using O(1) (i.e. constant) memory?

**Attempt 1: 2023-02-19**

**Solution 1:Fast and Slow pointer(30 min)**

/\*\*

\* 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) {

ListNode slow = head;

ListNode fast = head;

while(fast != null && fast.next != null) {

slow = slow.next;

fast = fast.next.next;

if(slow == fast) {

return true;

}

}

return false;

}

}

Time Complexity: O(n)

Space Complexity: O(1)

**Refer to**

<https://leetcode.com/problems/linked-list-cycle/solutions/1829641/c-my-first-c-code-explained/>

It's a classic algo for detecting cycles in a linked list. We use two pointers to traverse the list: The first one is moving one node at the time and the second two nodes at the time. If there is a cycle, sooner or later pointers will meet and we return true. If the fast pointer reached the end of the list, that means there is no cycle and we can return false.

*For reference: [Floyd's Cycle Detection Algorithm](https://en.wikipedia.org/wiki/Cycle_detection" \l "Floyd's_tortoise_and_hare)*

Time: O(n) - for traversing

Space: O(1) - nothing stored

class Solution {

public:

bool hasCycle(ListNode \*head) {

ListNode \*slow = head, \*fast = head;

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast) return true;

}

return false;

}

};