Previous Next

141. Linked List Cycle

March 12, 2016 | 257.9K views

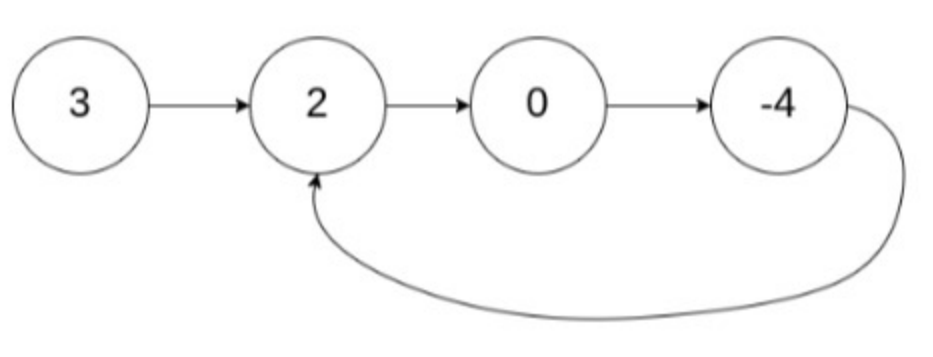
Average Rating: 4.80 (182 votes)

Given a linked list, determine if it has a cycle in it.

To represent a cycle in the given linked list, we use an integer `pos` which represents the position (0-indexed) in the linked list where tail connects to. If `pos` is `-1`, then there is no cycle in the linked list.

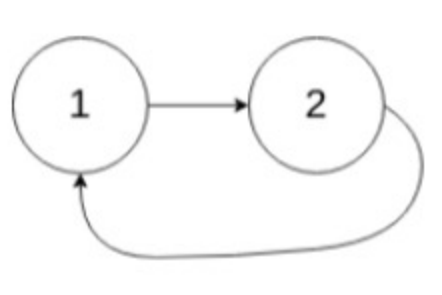
Example 1:

```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where tail connects to the second no
```



Example 2:

```
Input: head = [1,2], pos = 0
Output: true
Explanation: There is a cycle in the linked list, where tail connects to the first no
```



Example 3:

```
Input: head = [1], pos = -1
Output: false
Explanation: There is no cycle in the linked list.
```

Follow up:

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

Summary

This article is for beginners. It introduces the following ideas: Linked List, Hash Table and Two Pointers.

Solution

Approach 1: Hash Table

Intuition

To detect if a list is cyclic, we can check whether a node had been visited before. A natural way is to use a hash table.

Algorithm

We go through each node one by one and record each node's reference (or memory address) in a hash table. If the current node is `null`, we have reached the end of the list and it must not be cyclic. If current node's reference is in the hash table, then return true.

Java | Copy

```java
public boolean hasCycle(ListNode head) {
    Set<ListNode> nodesSeen = new HashSet<>();
    while (head != null) {
        if (nodesSeen.contains(head)) {
            return true;
        } else {
            nodesSeen.add(head);
        }
        head = head.next;
    }
    return false;
}
```

Complexity analysis

- Time complexity : $O(n)$. We visit each of the n elements in the list at most once. Adding a node to the hash table costs only $O(1)$ time.
- Space complexity: $O(n)$. The space depends on the number of elements added to the hash table, which contains at most n elements.

Approach 2: Two Pointers

Intuition

Imagine two runners running on a track at different speed. What happens when the track is actually a circle?

Algorithm

The space complexity can be reduced to $O(1)$ by considering two pointers at **different speed** - a slow pointer and a fast pointer. The slow pointer moves one step at a time while the fast pointer moves two steps at a time.

If there is no cycle in the list, the fast pointer will eventually reach the end and we can return false in this case.

Now consider a cyclic list and imagine the slow and fast pointers are two runners racing around a circle track. The fast runner will eventually meet the slow runner. Why? Consider this case (we name it case A) - The fast runner is just one step behind the slow runner. In the next iteration, they both increment one and two steps respectively and meet each other.

How about other cases? For example, we have not considered cases where the fast runner is two or three steps behind the slow runner yet. This is simple, because in the next or next's next iteration, this case will be reduced to case A mentioned above.

Java | Copy

```java
public boolean hasCycle(ListNode head) {
    if (head == null || head.next == null) {
        return false;
    }
    ListNode slow = head;
    ListNode fast = head.next;
    while (slow != fast) {
        if (fast == null || fast.next == null) {
            return false;
        }
        slow = slow.next;
        fast = fast.next.next;
    }
    return true;
}
```

Complexity analysis

- Time complexity : $O(n)$. Let us denote n as the total number of nodes in the linked list. To analyze its time complexity, we consider the following two cases separately.
  - ***List has no cycle:***
    The fast pointer reaches the end first and the run time depends on the list's length, which is $O(n)$.
  - ***List has a cycle:***
    We break down the movement of the slow pointer into two steps, the non-cyclic part and the cyclic part:
    1. The slow pointer takes "non-cyclic length" steps to enter the cycle. At this point, the fast pointer has already reached the cycle. $\text{Number of iterations} = \text{non-cyclic length} = N$
    2. Both pointers are now in the cycle. Consider two runners running in a cycle - the fast runner moves 2 steps while the slow runner moves 1 steps at a time. Since the speed difference is 1, it takes $\frac{\text{distance between the 2 runners}}{\text{difference of speed}}$ loops for the fast runner to catch up with the slow runner. As the distance is at most "cyclic length K" and the speed difference is 1, we conclude that $\text{Number of iterations} = \text{almost}$ "cyclic length K".

  Therefore, the worst case time complexity is $O(N+K)$, which is $O(n)$.
- Space complexity : $O(1)$. We only use two nodes (slow and fast) so the space complexity is $O(1)$.

Rate this article:

Previous Next

Comments: 76 Sort By

Type comment here... (Markdown is supported)

Preview Post

jianchao-li ★ 14342 July 14, 2018 8:28 PM

The `slow` and `fast` points can also both start at `head`.

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

Read More

176 Share Reply

SHOW 13 REPLIES

asharma78901 ★ 89 March 10, 2019 1:50 AM

The last algorithm is called 'Floyd's Tortoise and Hare'

88 Share Reply

SHOW 3 REPLIES

Argondey ★ 45 August 10, 2018 10:25 PM

I used a slightly faster destructive approach. I created a new node called "mark" and iterated through the list, setting the next value of each node to the mark node. When reaching the new node the first thing I did was check if the node is the mark node. If it ever is the mark node, we have looped.

45 Share Reply

SHOW 9 REPLIES

terrible_whiteboard ★ 627 May 19, 2020 6:28 PM

I made a video if anyone is having trouble understanding the solution (clickable link)
https://youtu.be/sMqEwkpvJvQ

Read More

19 Share Reply

lostandfoundii ★ 182 June 16, 2018 7:01 AM

Genius approach 2!!!

18 Share Reply

Dr_Sean ★ 535 November 19, 2019 4:48 AM

Python3 code using two-pointers approach:

```
        if not head or not head.next: return False
        slow, fast = head, head.next
```

Read More

11 Share Reply

SHOW 1 REPLY

leaping_tiger ★ 14 November 1, 2019 4:35 AM

What is the role of `pos` here? I'm so confused! Because if this variable is given at input (which seems like it is), then it gives away the answer. So, why do we have it and what do we do with it? Thank you.

7 Share Reply

cypherphage ★ 22 April 8, 2019 2:53 PM

Can someone please validate this answer. It passes the default test case, I don't know if its a fluke or not I am new to leetcode

```
class Solution(object):
```

Read More

10 Share Reply

SHOW 2 REPLIES

dayatar ★ 46 March 13, 2019 2:22 AM

Solution: time complexity : O(n), space 0(1)

Reverse a linked list, if you get the same head that means the linked List has a cycle otherwise it doesn't

Read More

8 Share Reply

SHOW 3 REPLIES

jrico59 ★ 13 April 3, 2020 8:03 AM

Very confused because in the Ruby version there is no "pos" input as mentioned in the description - only a head node.

4 Share Reply