Question : 141. Linked List Cycle

141. Linked List Cycle

Easy 14.3K 1.2K

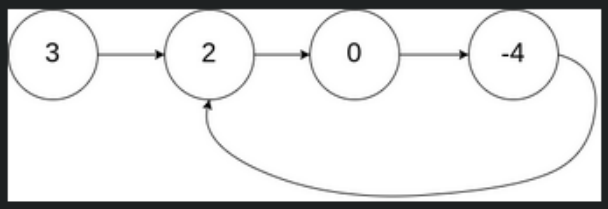
Companies

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).
```

Solution

```
public class Solution {
   public boolean hasCycle(ListNode head) {
      // Create 2 nodes slow & fast
      ListNode slow = head;
      ListNode fast = head;

      while(fast != null && fast.next != null){
         // Slow will move one step
         // Fast will move two steps
         slow = slow.next;
         fast = fast.next.next;

         // Detect if cycle or not
         if(slow == fast){
           return true;
         }
      }
      // If no cycle return false
      return false;
   }
}
```

Question : 142. Linked List Cycle II

142. Linked List Cycle II

Medium 12.8K 895

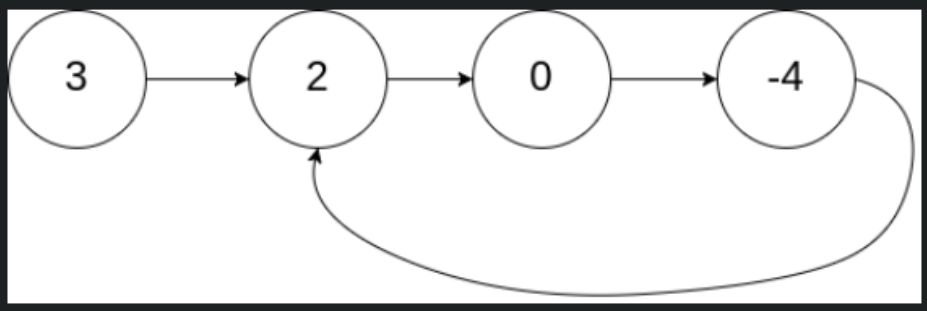
Companies

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.
```

Solution

```
public class Solution {
   public ListNode detectCycle(ListNode head) {
     ListNode fast = head;
       ListNode slow = head;
       // fast itself shouldn't be null & fast.next shouldn't be null
as well
       // or else you'll get a null pointer exception here
       while (fast != null && fast.next != null) {
          slow = slow.next;
          fast = fast.next.next;
          if (fast == slow) {
             slow = head;
             // use this while loop inside the if statement
             while (slow != fast) {
                slow = slow.next;
                fast = fast.next;
             }
             return slow;
          }
       }
       return null;
   }
}
```

Question : 202. Happy Number

202. Happy Number

Easy 9.6K 1.3K

Companies

Write an algorithm to determine if a number `n` is happy.

A **happy number** is a number defined by the following process:

- Starting with any positive integer, replace the number by the sum of the squares of its digits.
- Repeat the process until the number equals 1 (where it will stay), or it **loops endlessly in a cycle** which does not include 1.
- Those numbers for which this process **ends in 1** are happy.

Return `true` *if* `n` *is a happy number, and* `false` *if not.*

Example 1:

```
Input: n = 19
Output: true
Explanation:
1² + 9² = 82
8² + 2² = 68
6² + 8² = 100
1² + 0² + 0² = 1
```

Solution

```
class Solution {
   public boolean isHappy(int n) {
       int slow = n;
      int fast = n;
      // Now we have to move the slow & fast
      do {
         // moving slow one step ahead
         slow = findSquare(slow);
         // moving fast 2 steps forward
         fast = findSquare(findSquare(fast));
      } while (slow != fast);

      if (slow == 1) {
         return true;
      } else return false;
   }
   public int findSquare(int number) {
      int ans = 0;
      while (number > 0) {
         int rem = number % 10;
         ans = ans + rem * rem;
         number = number / 10;
      }
      return ans;
   }
}
```