

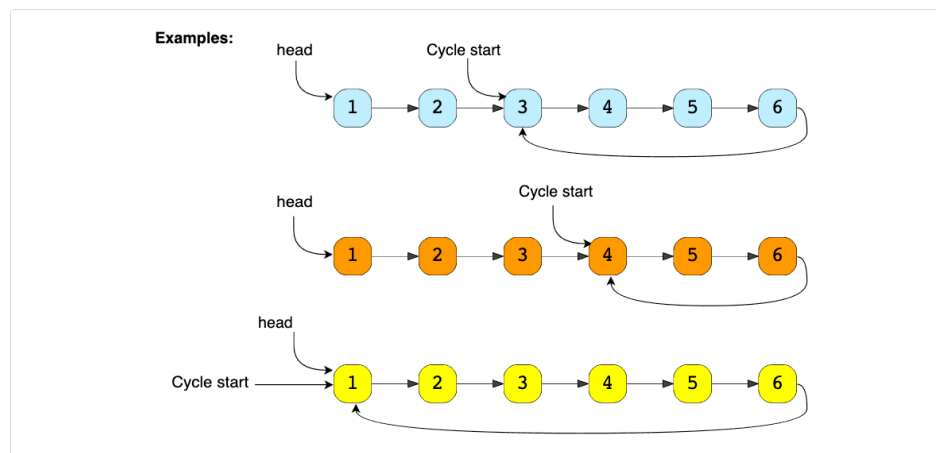
Start of LinkedList Cycle (medium)

We'll cover the following ^

- Problem Statement
- Try it yourself
- Solution
 - Code
 - Time Complexity
 - Space Complexity

Problem Statement

Given the head of a **Singly LinkedList** that contains a cycle, write a function to find the **starting node of the cycle**.



Try it yourself

Try solving this question here:

```
Java Python3 JS C++  
1 class ListNode {  
2     int value = 0;  
3     ListNode next;  
4  
5     ListNode(int value) {  
6         this.value = value;  
7     }  
8 }  
9  
10 class LinkedListCycleStart {  
11  
12     public static ListNode findCycleStart(ListNode head) {  
13         // TODO: Write your code here  
14         return head;  
15     }  
16  
17     public static void main(String[] args) {  
18         ListNode head = new ListNode(1);  
19         head.next = new ListNode(2);  
20         head.next.next = new ListNode(3);  
21         head.next.next.next = new ListNode(4);  
22         head.next.next.next.next = new ListNode(5);  
23         head.next.next.next.next.next = new ListNode(6);  
24  
25         head.next.next.next.next.next.next = head.next.next;  
26         System.out.println("LinkedList cycle start: " + LinkedListCycleStart.findCycleStart(head).value);  
27  
28         head.next.next.next.next.next.next = head.next.next.next;  
29     }  
30 }
```

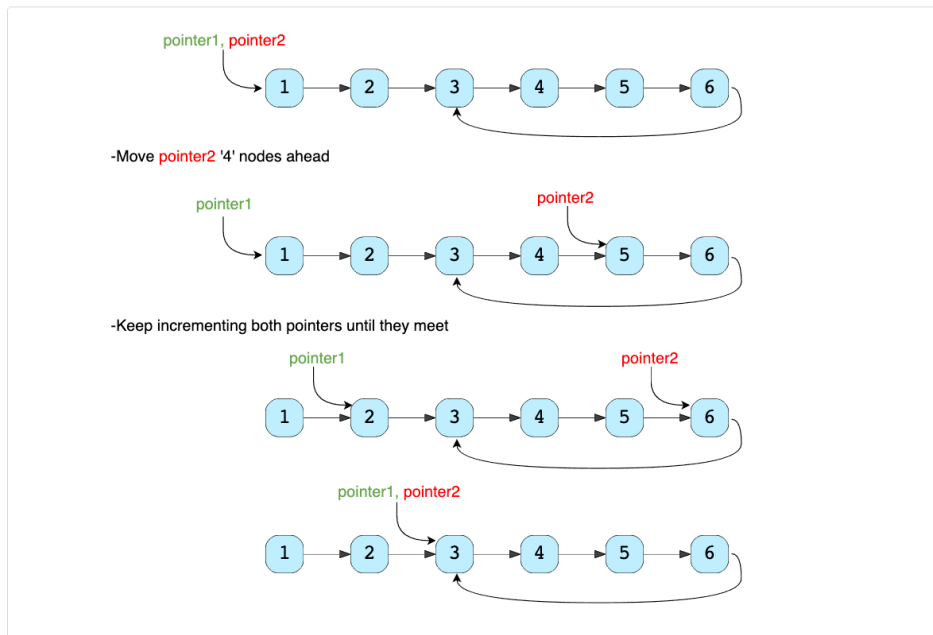
Run Save Reset

Solution

If we know the length of the **LinkedList** cycle, we can find the start of the cycle through the following steps:

1. Take two pointers. Let's call them **pointer1** and **pointer2**.
2. Initialize both pointers to point to the start of the LinkedList.
3. We can find the length of the LinkedList cycle using the approach discussed in [LinkedList Cycle](#). Let's assume that the length of the cycle is 'K' nodes.
4. Move **pointer2** ahead by 'K' nodes.
5. Now, keep incrementing **pointer1** and **pointer2** until they both meet.
6. As **pointer2** is 'K' nodes ahead of **pointer1**, which means, **pointer2** must have completed one loop in the cycle when both pointers meet. Their meeting point will be the start of the cycle.

Let's visually see this with the above-mentioned Example-1:



We can use the algorithm discussed in [LinkedList Cycle](#) to find the length of the cycle and then follow the above-mentioned steps to find the start of the cycle.

Code

Here is what our algorithm will look like:

```
Java Python3 C++ JS
1 class ListNode {
2     int value = 0;
3     ListNode next;
4
5     ListNode(int value) {
6         this.value = value;
7     }
8 }
9
10 class LinkedListCycleStart {
11
12     public static ListNode findCycleStart(ListNode head) {
13         int cycleLength = 0;
14         // find the LinkedList cycle
15         ListNode slow = head;
16         ListNode fast = head;
17         while (fast != null && fast.next != null) {
18             fast = fast.next.next;
19             slow = slow.next;
20             if (slow == fast) { // found the cycle
21                 cycleLength = calculateCycleLength(slow);
22                 break;
23             }
24         }
25
26         return findStart(head, cycleLength);
27     }
28 }
```

Run Save Reset

Time Complexity

As we know, finding the cycle in a LinkedList with 'N' nodes and also finding the length of the cycle requires $O(N)$. Also, as we saw in the above algorithm, we will need $O(N)$ to find the start of the cycle. Therefore, the overall time complexity of our algorithm will be $O(N)$.

Space Complexity

The algorithm runs in constant space $O(1)$.



[← Back](#)

LinkedList Cycle (easy)

[Next →](#)

Happy Number (medium)

✓ Completed

 Report an Issue  Ask a Question