**Summary - Two-Pointer in Linked List**

Here we provide a template for you to solve the two-pointer problem in the linked list. (java)

// Initialize slow & fast pointers

ListNode\* slow = head;

ListNode\* fast = head;

/\*\*

\* Change this condition to fit specific problem.

\* Attention: remember to avoid null-pointer error

\*\*/

while (slow && fast && fast->next) {

slow = slow->next; // move slow pointer one step each time

fast = fast->next->next; // move fast pointer two steps each time

if (slow == fast) { // change this condition to fit specific problem

return true;

}

}

return false; // change return value to fit specific problem

*Tips*

It is similar to what we have learned in an array. But it can be trickier and error-prone. There are several things you should pay attention:

**1. Always examine if the node is null before you call the next field.**

Getting the next node of a null node will cause the null-pointer error. For example, before we run fast = fast.next.next, we need to examine both fast and fast.next is not null.

**2. Carefully define the end conditions of your loop.**

Run several examples to make sure your end conditions will not result in an endless loop. And you have to take our first tip into consideration when you define your end conditions.

*Complexity Analysis*

It is easy to analyze the space complexity. If you only use pointers without any other extra space, the space complexity will be O(1). However, it is more difficult to analyze the time complexity. In order to get the answer, we need to analyze how many times we will run our loop .

In our previous finding cycle example, let's assume that we move the faster pointer 2 steps each time and move the slower pointer 1 step each time.

1. If there is no cycle, the fast pointer takes N/2 times to reach the end of the linked list, where N is the length of the linked list.
2. If there is a cycle, the fast pointer needs M times to catch up the slower pointer, where M is the length of the cycle in the list.

Obviously, M <= N. So we will run the loop up to N times. And for each loop, we only need constant time. So, the time complexity of this algorithm is O(N) in total.

Analyze other problems by yourself to improve your analysis skill. Don't forget to take different conditions into consideration. If it is hard to analyze for all situations, consider the worst one.