LeetCode 例题精讲 | 05 双指针×链表问题：快慢指针

原创 nettee 面向大象编程 2020-02-18

收录于话题
#LeetCode 例题精讲

1.获取倒数第k个元素
2.获取中间位置的元素
3.判断链表是否存在环
4.判断环的长度
如果存在环，如何判断环的长度呢？方法是，快慢指针相遇后继续移动，直到第二次相遇。两次相遇间的移动次数即为环的长度
5.找到链表入环的第一个节点

23个

本期例题：

- **LeetCode 876 - Middle of the Linked List**[1]（Easy）寻找链表中点
- **LeetCode 19 - Remove Nth Node From End of List**[2]（Medium）寻找链表的倒数第 k 个元素
- **LeetCode 141 - Linked List Cycle**[3]（Easy）判断链表中是否存在环

上一篇文章中，我们分析了经典的 Two Sum II 问题与双指针方法。双指针这个名字在很多题解中都能见到，那么，还有什么题目可以用双指针方法解决呢？

实际上，双指针是一个很笼统的概念。只要在解题时用到了两个指针（链表指针、数组下标皆可），都可以叫做双指针方法。根据两个指针运动方式的不同，双指针方法可以分成同向指针、对向指针、快慢指针等。

当双指针方法遇到链表问题，我们主要使用快慢指针方法。很多时候链表操作遇到疑难杂症时，可以使用快慢指针，减少算法所需的时间或者空间。

这篇文章将会包含：

- 链表问题中的快慢指针特点
- 快慢指针的案例讲解：寻找链表中点、寻找链表的倒数第 k 个元素
- 快慢指针的经典例题：判断链表中是否存在环
- 相关题目

链表问题中的双指针

我们知道，链表数据类型的特点决定了它只能单向顺序访问，而不能逆向遍历或随机访问（按下标访问）。很多时候，我们需要使用快慢指针的技巧来实现一定程序的逆向遍历，或者减少遍历的次数。这就是为什么快慢指针常用于链表问题。

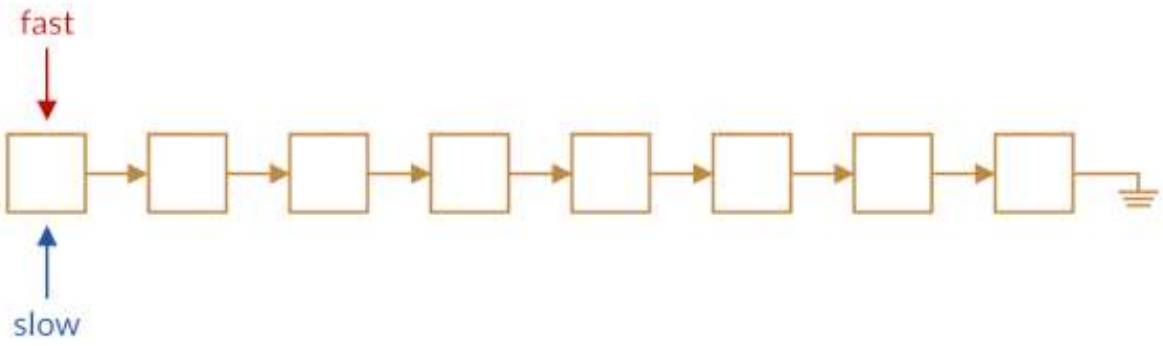
链表问题中的快慢指针又可以细分为多个不同类型。下面将依次介绍快慢指针的三个应用场景。

例题 1：寻找链表中点

LeetCode 876 - Middle of the Linked List[4]（Easy）

寻找链表中点的常规做法是，先遍历一遍链表，计算链表的长度 n。再遍历一遍链表，找到第 $n/2$ 个元素。这样需要遍历链表两遍。

更巧妙的做法是使用一快一慢两个指针。快指针一次前进两个结点，速度是慢指针的两倍。这样当快指针到达链表尾部时，慢指针正好到达的链表的中部。过程如下方动图所示。这样只需遍历链表一遍。当然，时间复杂度仍然是 $O(n)$，不过链表类题目就是需要这样减少一次遍历的技巧。



快慢指针不同速度前进（动图）

题解代码如下所示。注意循环的条件是 `fast != null && fast.next != null`，防止出现空指针异常。

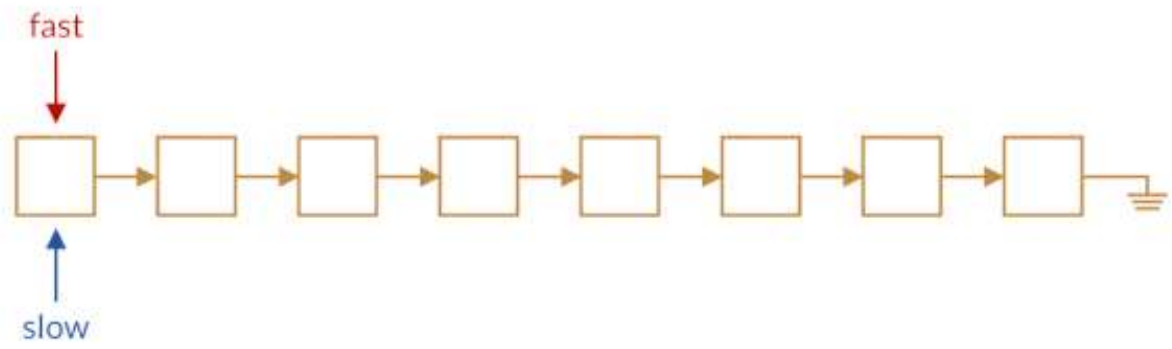
```
public ListNode middleNode(ListNode head) {
    ListNode fast = head;
    ListNode slow = head;
    while (fast != null && fast.next != null) {
        // fast 一次前进两个元素，slow 一次前进一个元素
        fast = fast.next.next;
        slow = slow.next;
    }
    // 链表元素为奇数个时，slow 指向链表的中点
    // 链表元素为偶数个时，slow 指向链表两个中点的右边一个
    return slow;
}
```

例题 2：寻找链表的倒数第 k 个元素

LeetCode 19 - Remove Nth Node From End of List[5]（Medium）

寻找链表的倒数第 k 个元素的常规做法是，先遍历一遍链表，计算链表的长度 n。这样就可以计算出要找第 $n-k$ 个元素，再遍历一遍链表即可。

这里同样可以使用快慢指针方法减少一次遍历。不过这次两个指针速度相同，只是间隔一定距离。快指针先前进 k 个元素，然后两个指针同样速度前进。这样当快指针到达链表尾部时，慢指针正好到达链表的倒数第 k 个元素。下面的动图展示了快慢指针的前进过程（以 $k=3$ 为例）。



快慢指针间隔 3 个元素前进（动图）

题解代码如下所示。注意例题中除了寻找倒数第 k 个元素，还需要删除它。为了删除元素，我们需要用到第一讲中所讲的链表遍历框架，将 `slow` 指针变成 `prev` 和 `curr` 两个指针。（关于 `prev` 和 `curr` 指针的具体用法，请自行回顾第一讲。）所以这里实际上是三个指针一起前进，注意分辨指针之间的关系。

```
public ListNode removeNthFromEnd(ListNode head, int k) {
    // 将 fast 前进 k 个元素
    ListNode fast = head;
    for (int i = 0; i < k; i++) {
        // 这里省略了检测空指针的代码
        fast = fast.next;
    }
    // fast 和 slow 指针间隔 k 个同时前进
    // 这里使用了链表遍历框架，将 slow 指针变成两个指针 curr 和 prev
    ListNode curr = head;
    ListNode prev = null;
    while (fast != null) {
        prev = curr;
```

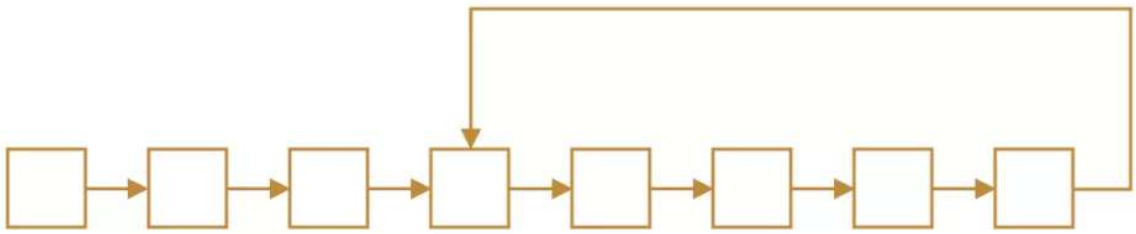
```
            prev = curr;
            curr = curr.next;
            fast = fast.next;
        }
        if (prev == null) {
            head = curr.next;
        } else {
            prev.next = curr.next;
        }
        return head;
    }
```

这里题目假设了 k 的取值一定是合法的，因此代码中省略了若干合法性检查。但是在面试中，我们应该考虑输入 k 不合法的情况，保证程序的鲁棒性。

例题 3：判断链表中是否存在环

LeetCode 141 - Linked List Cycle[6]（Easy）

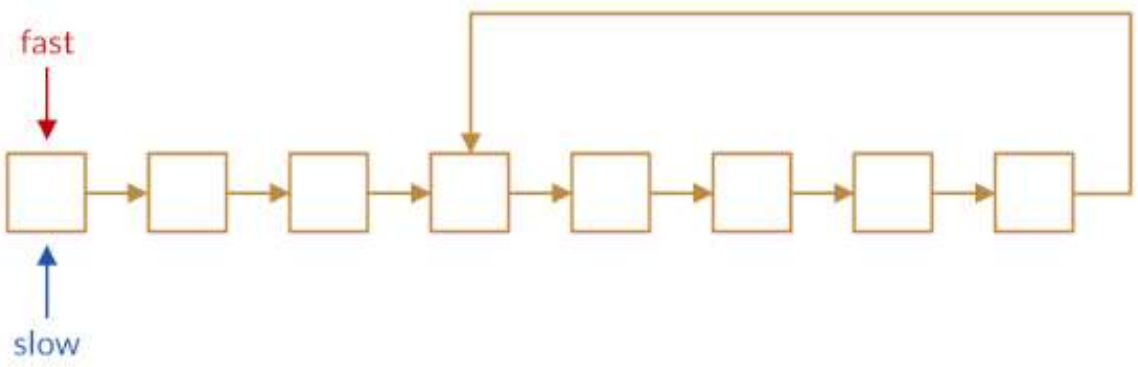
这道题的链表进行了小小的变种。链表的尾结点有可能指向链表中的某个结点，让链表成为环形，如下图所示。



链表中的环

我们需要判断链表是普通链表还是带环的链表。如果只使用一个指针遍历的话，其实是很难判断是环的存在的，只能用一个 set 保存遍历过的所有结点。而如果使用快慢指针的话，则可以不使用额外空间。

我们让快指针一次前进两个结点，慢指针一次前进一个结点。当快慢指针都进入环路时，快指针会将慢指针“套圈”，从后面追上慢指针。追及的过程如下面的动图所示。



快慢指针不同速度前进并追及（动图）

而如果链表中不存在环的话，快指针则不会和慢指针指向同一个结点，而是会走到链表结尾。依照这个思路我们可以得到以下的题解代码。

```
public boolean hasCycle(ListNode head) {
    ListNode fast = head;
    ListNode slow = head;
    while (fast != null && fast.next != null) {
        fast = fast.next.next;
        slow = slow.next;
        // fast 和 slow 指向同一个结点，说明存在“套圈”
        if (fast == slow) {
            returntrue;
        }
    }
    // fast 到达链表尾部，则不存在环
    returnfalse;
}
```

这一题还有第二问，**LeetCode 142 - Linked List Cycle II**[7]，找到链表入环的第一个结点。这也需要用到快慢指针，还涉及到一些证明。这一题本文就不展开讲了，有兴趣的同学可以自己尝试解决此题。

总结

本文讲解的是双指针技巧中的一类“快慢指针”，用于解决链表类问题。而快慢指针又存在着多个变种，文中也一一列举了。

链表类题目并不复杂，本系列第一讲的链表遍历框架和本文所讲的快慢指针就是链表类问题中最主要的两个技巧了。链表类问题只要多加练习，都非常的容易。

最后，附上一道链表问题的综合练习题：链表排序（**LeetCode 148 - Sort List**[8]）。这道题中需要使用到链表处理的各种技巧，还需要针对链表的性质选择合适的排序方法。可以说只要掌握了这道题，就算掌握了链表类问题了。链表排序也是微软面试中某个面试官的“三板斧”之一，可以看出这道题的普适性。

上一讲和这一讲我们分别了解了两种不同类型的双指针方法：对向指针和快慢指针。双指针还有一种经典的类型是滑动窗口，将在后面的章节进行讲解。敬请期待。

参考资料

[1] LeetCode 876 - Middle of the Linked List: *https://leetcode.com/problems/middle-of-the-linked-list/*

[2] LeetCode 19 - Remove Nth Node From End of List: *https://leetcode.com/problems/remove-nth-node-from-end-of-list/*

[3] LeetCode 141 - Linked List Cycle: *https://leetcode.com/problems/linked-list-cycle/*

[4] LeetCode 876 - Middle of the Linked List: *https://leetcode.com/problems/middle-of-the-linked-list/*

[5] LeetCode 19 - Remove Nth Node From End of List: *https://leetcode.com/problems/remove-nth-node-from-end-of-list/*

[6] LeetCode 141 - Linked List Cycle: *https://leetcode.com/problems/linked-list-cycle/*

[7] LeetCode 142 - Linked List Cycle II: *https://leetcode.com/problems/linked-list-cycle-ii/*

[8] LeetCode 148 - Sort List: *https://leetcode.com/problems/sort-list/*