拉勾教育
—互联网人实战大学—

链表+快慢指针：环形链表

简单/链表、快慢指针

学习目标

- 了解算法题的解题思路
- 链表/环形链表的特点
- 快慢指针的应用

题目描述

给定一个链表，判断链表中是否有环。存在环返回 true ， 否则返回 false。

- 连续跟踪 next 指针再次到达某个节点，则链表中有环。
- 你能用 O(1)（即，常量）内存解决此问题吗?
- 提示：
  - 链表中节点的数目范围是 [0, 10^4]
  - -10^5 <= Node.val <= 10^5
  - pos 为环开始节点的索引，若pos = -1，则没有环
  - pos 为 -1 或者链表中的一个 有效索引
  - pos 不作为参数进行传递，仅仅是为了标识链表的实际情况

```
public class Solution {
    public boolean hasCycle(ListNode head){
        return false;
    }
}
class ListNode {
    int val;
    ListNode next;

    ListNode(int x) {
        val = x;
        next = null;
    }
}
```

题目描述

示例1：链表中没有环

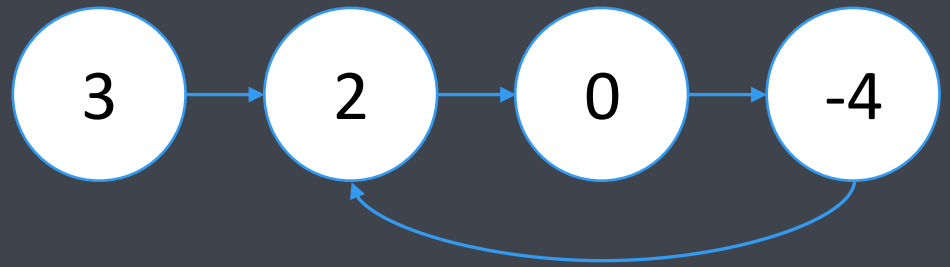
- 输入：head = [1], pos = -1
- 输出：false

示例2：有一个环，尾部连接到第1个节点

- 输入：head = [1,2], pos = 0
- 输出：true

示例3：有一个环，尾部连接到第2个节点

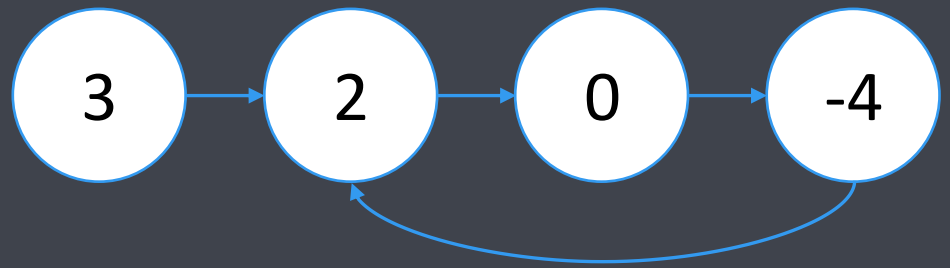
- 输入：head = [3,2,0,-4], pos = 1
- 输出：true



一. Comprehend 理解题意

该题可以理解为检测链表的某节点能否二次到达（重复访问）的问题。

- 需要一个容器记录已经访问过的节点
- 每次访问到新的节点，都与容器中的记录进行匹配，若相同则存在环
- 若匹配之后没有相同节点，则存入容器，继续访问新的节点
- 直到访问节点的next指针返回null，或者当前节点与容器的某个记录相同，操作结束

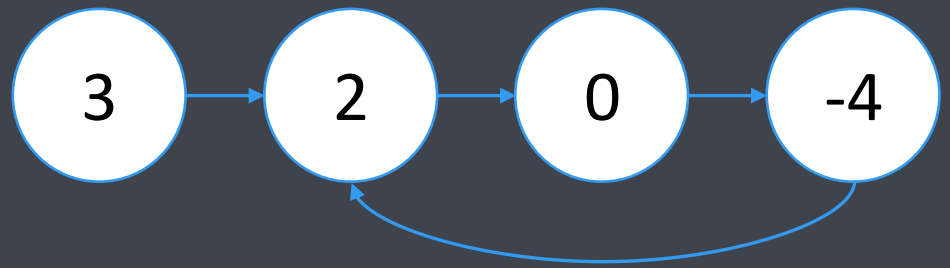


一. Comprehend 理解题意

也可以理解为“追击”问题，如果存在环，跑得快的一定能追上跑得慢的。

- 就像一快一慢两个运动员，如果在直道赛跑，不存在追击问题；如果是在环道赛跑，快的绕了一圈肯定可以追上慢的
- 定义两个快慢运动员，指向链表的第一、第二个节点：

  slow = head; fast = head.next;
- 快运动员的步长为2，慢的为1，即fast每次移动两个节点，slow每次移动一个节点
- 若最终fast==slow，说明存在环；或者fast/fast.next指向null，操作结束



二. Choose 数据结构及算法思维选择

拉勾教育
—互联网人实战大学—

解法一：二次到达解法

- 数据结构：数组

  可选的有：数组、链表、栈、队列

- 算法思维：遍历

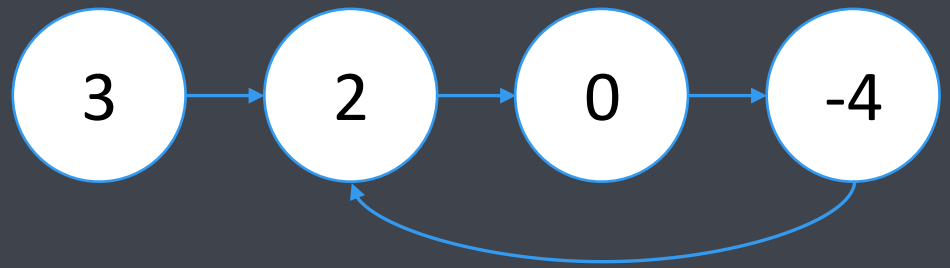
解法二：追击问题解法

- 数据结构：两个变量

  只需要额外定义slow和fast两个变量

- 算法思维：遍历、快慢指针

  fast、slow两个变量就是快慢指针



三. Code 基本解法及编码实现

解法一：二次到达解法思路分析

1. 定义数组记录已访问节点

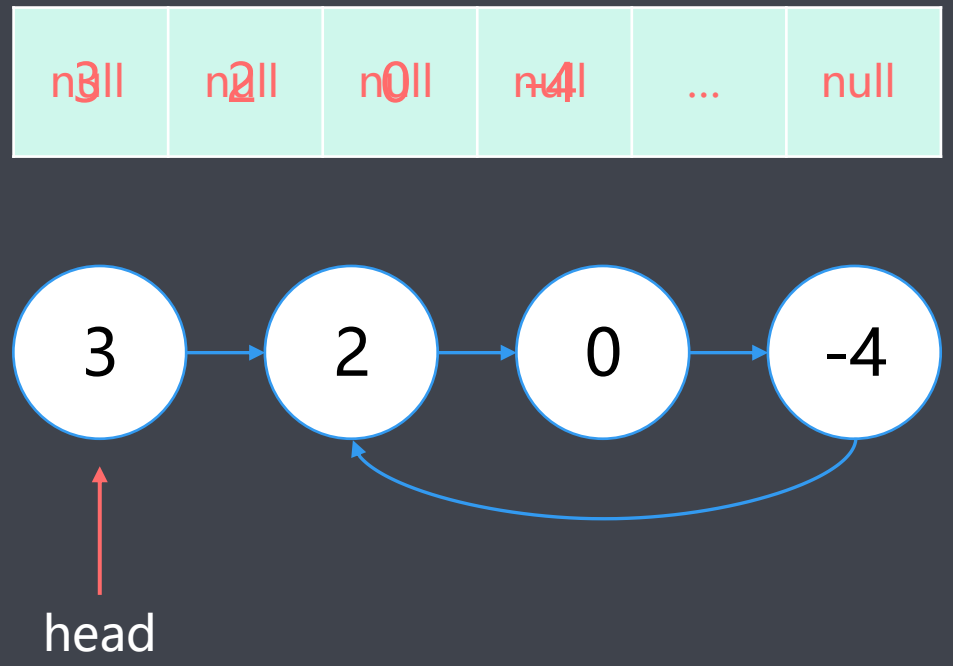
   new ListNode[10000];

2. 遍历链表的每个节点，并与容器中已存放的节点依次比较：

   相同则方法结束，返回true

   不同则存入最新位置，继续遍历下个节点

3. 若next指针为null，则方法结束，返回false



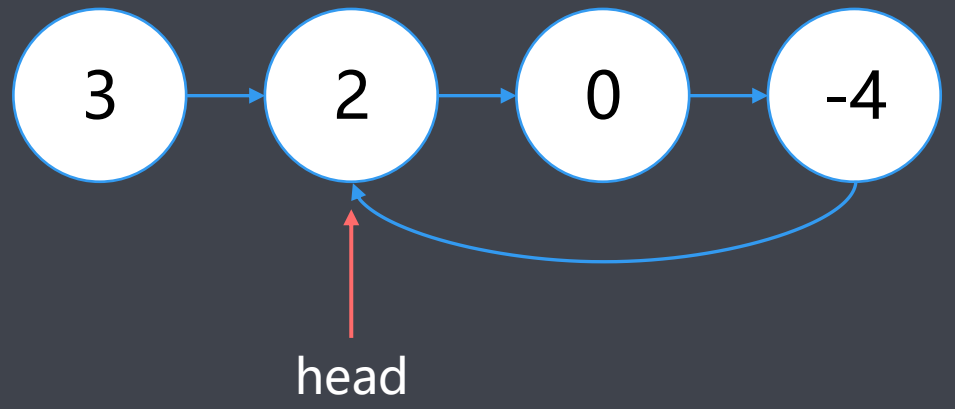
三. Code 基本解法及编码实现

解法一：边界和细节问题

1. 边界问题：

链表最多有一万个节点，容器不会越界；

与容器中节点进行比对，正向遍历容器，元素为null时终止，后续都是未使用的空间



2. 细节问题：

遍历链表，通过head=head.next进行迭代

当且仅当此节点与容器某个节点相同时返回true，其它情况都返回false

比较相等，可以用 ==

三. Code 基本解法及编码实现

拉勾教育
—互联网人实战大学—

```java
public boolean hasCycle(ListNode head) {
    // 1.定义数组记录已访问节点
    ListNode[] array = new ListNode[10000];
    // 2.遍历链表的每个节点,
    while(head != null) {
        // 并与容器中已存放的节点依次比较
        for (int i = 0; i < array.length; i++) {
            if (array[i] == head) {
                return true; // 相同则方法结束，返回true
            }
            if (array[i] == null) { // 该索引位置已存放节点
                array[i] = head; // 将当前节点存放到此位置
                break; // 结束容器遍历，因为后续都是空位置
            }
        }
        head = head.next; // 迭代，指向下一个节点
    } // 3.若next指针为null，则方法结束，返回false
    return false;
}
```

时间复杂度：O(n^2)

- 遍历整个链表：O(n)
- 每次遍历节点，再遍历数组进行匹配：O(n-1)
- 总时间复杂度：O(n^2)

空间复杂度：O(1)

- 额外固定10000长度的数组开销，空间复杂度：O(1)
- 从绝对空间消耗来说，消耗非常大

执行耗时:108 ms,击败了5.26% 的Java用户
内存消耗:38.3 MB,击败了99.89% 的Java用户

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

1. 剔除无效代码或优化空间消耗
   - 每个节点都需要遍历容器查找，比较耗时
   - 按最大测试数据量创建容器，空间消耗巨大
2. 寻找更好的算法思维
   - 要证明某个节点是否第二次到达，可否将已遍历节点进行标记?
   - 环形结构对应生活中的追击问题，可否使用“追击问题”模拟实现?
   - 借鉴其它算法

五. Code 最优解思路及编码实现

解法二：追击问题解法思路分析

1. 定义快慢两个指针：

   slow=head; fast=head.next;

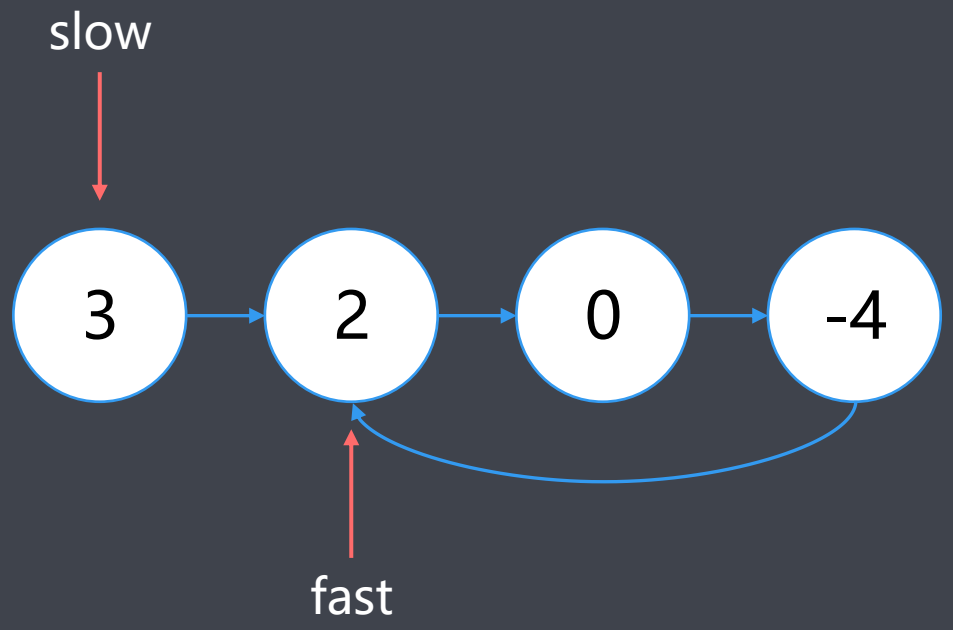
2. 遍历链表：

   快指针步长为2：fast=fast.next.next;

   慢指针步长为1：slow=slow.next;

3. 当且仅当快慢指针重合，有环，返回true

4. 快指针为null，或其next指向null，没有环，返回false，操作结束



五. Code 最优解思路及编码实现

拉勾教育
—互联网人实战大学—

```java
public boolean hasCycle(ListNode head) {
    if (head == null) // 链表中有节点[0, 10^4]个
        return false;
    // 1.定义快慢两个指针
    ListNode slow = head; // head != null，慢
    ListNode fast = head.next; // 快
    // 2.遍历链表：快指针步长为2，慢指针步长为1
    // 快指针先到终点。while任意判断条件成立，说明fast已到终点
    while (fast != null && fast.next != null) { // 4.反之没有环
        // 3.当且仅当快慢指针重合：有环，操作结束
        if (slow == fast) {
            return true;
        }
        fast = fast.next.next; // 快指针步长为2
        slow = slow.next; // 慢指针步长为1
    }
    return false;
}
```

时间复杂度：O(n)

- 遍历整个链表：O(n)

空间复杂度：O(1)

- 额外两个变量：O(1)

执行耗时:0 ms,击败了100% 的Java用户
内存消耗:38.8 MB,击败了88.78% 的Java用户

六. Change 变形延伸

拉勾教育
—互联网人实战大学—

题目变形

- （练习）分别使用head和head.next作为链表的慢/快指针
- （练习）标记值法：对节点的val属性进行标记，赋一个超出合法范围的值
- （练习）hash表法

延伸扩展

- 龟兔赛跑问题，追击问题
- 地球，火星与太阳连为一线的问题

本题来源：

- Leetcode 141 https://leetcode-cn.com/problems/linked-list-cycle/

总结

链表的基本概念，特点

快慢针的思想以及使用

复杂度的计算方法，包括时间复杂度与空间复杂度

课后练习

拉勾教育
—互联网人实战大学—

1. 环形链表 II （Leetcode 142/中等）
2. 反转链表（剑指 Offer 24/简单）
3. 最低加油次数（Leetcode 871/困难）
4. 复杂链表的复制（剑指 Offer 35/中等）

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容