本日学习内容

1. 复习之前写过的算法题，准备考核
2. 开始学习数据结构

本日分享内容

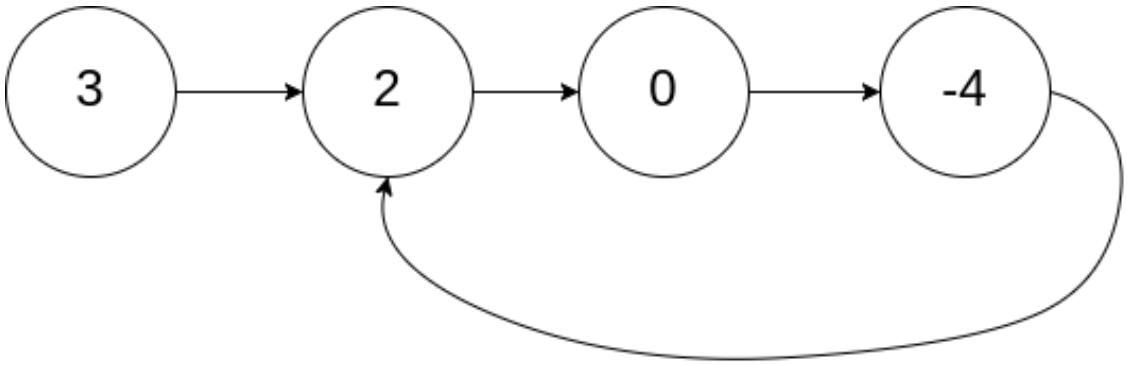
题目一：142. 环形链表 II

给定一个链表的头节点 `head` ，返回链表开始入环的第一个节点。 *如果链表无环，则返回* `null`。

如果链表中有某个节点，可以通过连续跟踪 `next` 指针再次到达，则链表中存在环。 为了表示给定链表中的环，评测系统内部使用整数 `pos` 来表示链表尾连接到链表中的位置（**索引从 0 开始**）。如果 `pos` 是 `-1`，则在该链表中没有环。**注意：`pos` 不作为参数进行传递**，仅仅是为了标识链表的实际情况。

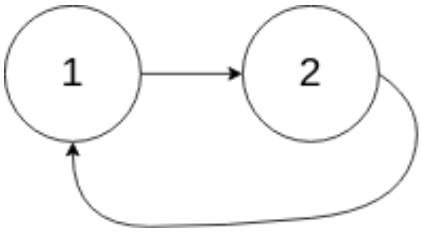
不允许修改 链表。

示例 1：




```
输入：head = [3,2,0,-4], pos = 1
输出：返回索引为 1 的链表节点
解释：链表中有一个环，其尾部连接到第二个节点。
```

示例 2：




```
输入：head = [1,2], pos = 0
输出：返回索引为 0 的链表节点
解释：链表中有一个环，其尾部连接到第一个节点。
```

示例 3：


```
输入：head = [1], pos = -1
输出：返回 null
解释：链表中没有环。
```

提示：

- 链表中节点的数目范围在范围 `[0, 104]` 内
- `-105 <= Node.val <= 105`
- `pos` 的值为 `-1` 或者链表中的一个有效索引

进阶：你是否可以使用 `O(1)` 空间解决此题?

思路

1. 快指针（`fast`）每次走 2 步，慢指针（`slow`）每次走 1 步

   如果 `fast` 遇到 `nullptr`，说明无环。

   如果 `fast == slow`，说明有环。

2. 找到环的入口

   将 `fast` 重新指向 `head`，然后 `fast` 和 `slow` 每次各走 1 步。

   再次相遇的节点就是环的入口。

代码

```
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* detectCycle(ListNode* head) {
        ListNode* fast = head;
        ListNode* slow = head;
        while (fast && fast->next) {
            fast = fast->next->next;
            slow = slow->next;
            if (fast == slow) {
                ListNode* temp1 = head;
                ListNode* temp2 = slow;
                while (temp1 != temp2) {
                    temp1 = temp1->next;
                    temp2 = temp2->next;
                }
                return temp1;
            }
        }
        return NULL;
    }
};
```

题目二：15. 三数之和

给你一个整数数组 `nums` ，判断是否存在三元组 `[nums[i], nums[j], nums[k]]` 满足 `i != j`、`i != k` 且 `j != k` ，同时还满足 `nums[i] + nums[j] + nums[k] == 0` 。请你返回所有和为 `0` 且不重复的三元组。

注意：答案中不可以包含重复的三元组。

示例 1：

```
输入：nums = [-1,0,1,2,-1,-4]
输出：[[-1,-1,2],[-1,0,1]]
解释：
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0 。
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0 。
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0 。
不同的三元组是 [-1,0,1] 和 [-1,-1,2] 。
注意，输出的顺序和三元组的顺序并不重要。
```

示例 2：

```
输入：nums = [0,1,1]
输出：[]
解释：唯一可能的三元组和不为 0 。
```

示例 3：

```
输入：nums = [0,0,0]
输出：[[0,0,0]]
解释：唯一可能的三元组和为 0 。
```

提示：

- `3 <= nums.length <= 3000`
- `-105 <= nums[i] <= 105`

思路：

1. **排序**：先对数组排序，方便后续处理重复解和双指针遍历。
2. 固定一个数，用双指针找另外两个

   遍历数组，固定 `nums[i]`，然后在 `i+1` 到 `n-1` 的区间内用双指针 `left` 和 `right` 寻找 `nums[left] + nums[right] == -nums[i]`。

   如果 `sum < 0`，`left++`（需要更大的数）。

   如果 `sum > 0`，`right--`（需要更小的数）。

   如果 `sum == 0`，记录解，并跳过重复值。
3. 去重

   跳过 `nums[i]` 的重复值。

   在找到解后，跳过 `nums[left]` 和 `nums[right]` 的重复值。

代码：

```cpp
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        int n = nums.size();
        vector<vector<int>> ans;
        for (int i = 0; i < n - 2; i++) {
            if (nums[i] > 0)
                continue;
            if (i > 0 && nums[i] == nums[i - 1])
                continue;
            int l = i + 1, r = n - 1;
            while (l < r) {
                if (nums[i] + nums[l] + nums[r] > 0) {
                    r--;
                } else if (nums[i] + nums[l] + nums[r] < 0) {
                    l++;
                } else {
                    ans.push_back(vector<int>{nums[i], nums[l], nums[r]});
                    while (l < r && nums[r] == nums[r - 1]) {
                        r--;
                    }
                    while (l < r && nums[l] == nums[l + 1]) {
                        l++;
                    }
                    r--;
                    l++;
                }
            }
        }
        return ans;
    }
};
```

本日遇到的问题

1. 之前写过的题不能很快反应出来

明日学习内容

1. 继续写写动态规划部分题目
2. 开始学习二叉树，看大话数据结构