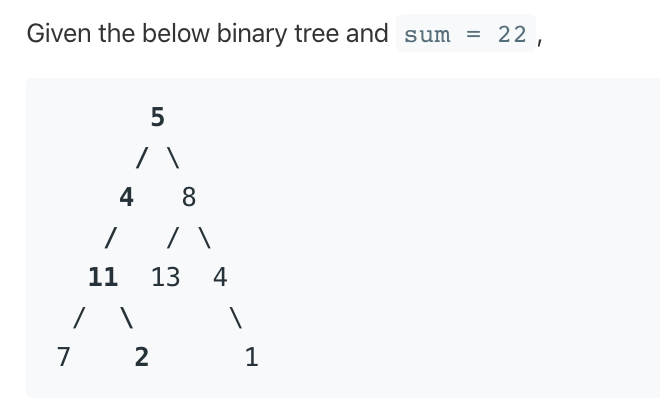
**前言：**

今天没有怎么遇到大的问题，对于二叉树至少easy的题都能驾轻就熟，然后首次使用HashSet，和HashMap差不多，只是两者数据结构不一样，一个是存储键值对，一个只是存值，但查询速度都是O(1)很快。

* 如果以后遇到偶数个数字相同数字，让你找没有重复的那一个数，一定要快速想到用异或，O(n)速度很快

#### LeetCode[112. Path Sum](https://leetcode-cn.com/problems/path-sum/)



这题很好理解，自己也做出来了，但是看到别人的代码更简洁，需要学习以下，它采用将sum-当前节点的值，如果遇到叶子节点就判断sum是否为0.

 public boolean hasPathSum(TreeNode root, int sum) {

            //DFS 回溯法

        if(root==null) return false;

        sum-=root.val;

        if(root.left==null&&root.right==null) return sum==0;

        return hasPathSum(root.left,sum)||hasPathSum(root.right,sum);

    }

#### LeetCode [141. Linked List Cycle](https://leetcode-cn.com/problems/linked-list-cycle/)

判断链表中是否有循环，hash的方式就不说用，用set存，如果遇到已经在set里就表示有循环。这里主要介绍一个快慢指针，就像操场跑步，快指针每次走两步，慢指针走一步，这样如果有环，两个指针迟早会相遇。

源代码如下：

public class Solution {

    public boolean hasCycle(ListNode head) {

        //快慢指针

        if(head==null||head.next==null)return false;

        ListNode slow=head;

        ListNode fast=head.next;

        while(slow!=fast){

            if(fast==null||fast.next==null)return false;

            slow=slow.next;

            fast=fast.next.next;

        }

        return true;

    }

}

**复杂度分析**

1. 如果没环只需要n/2次循环，复杂度为O(n)
2. 如果有环那设慢指针到环状部分的长度为K，这部分是肯定要走的；当慢指针进入环状部分以后，快指针肯定也在环状部分，两者的速度差为1，则两者相遇的次数为（快慢指针之间的距离）/速度差。两者距离最大为M，所以最大的复杂度为O(K+M)，也就是O(n)

**与hashset的比较**

因为出现循环的地方一定是在链表最后部分返回到前面，所以对哈希来说肯定是O(n),而快慢指针是最慢才O(n)，所以是快慢指针跟占优势。