二叉树的属性求解

二叉树的属性求解：基于基础遍历的方式，尤其关注层序遍历，并注意什么场景下应采用递归

二叉树的值求解

这里的值，通常包括求和、平均值、最大值、找某个位置的值等，可采用**层序遍历**

二叉树的层平均值

在二叉树的基础遍历中已有介绍

在每个树行中找到最大值

在二叉树的基础遍历中已有介绍

找树左下角的值

左下角的值，即二叉树最深一层的最左边节点的值，本题采用层序遍历，每次得到每层最左边节点的值（即每次循环时队列的第一个节点值），最后返回的就是所求的值

左叶子之和

对题目的理解：左叶子之和一定要是叶节点之和，而不是任意某节点的左节点

如何判断左叶节点：以父节点为起点，考虑父节点的左节点：如果它两个子节点都没有，那么父节点的左节点就是左叶节点

方法：采用**前序遍历**（层序遍历无法反映上述判断方式）

具体步骤：在采用栈的前序遍历基础上，对每个被弹出的栈顶元素进行上述判断，符合条件就加在输出结果

题目：

637. 二叉树的层平均值（简单）

515. 在每个树行中找到最大值（中等）

513. 找树左下角的值（中等）

404. 左叶子之和（简单）

二叉树的对称性问题

对称性问题：判断两节点（的值）是否相同

方法：利用**层序遍历**的程序框架，采用“**两两打包**”的方式加入队列，判断打包两节点的值是否相同（或同时为空节点）。**由于存在空节点的判断，队列需要声明成LinkedList类型**（ArrayDeque不支持队列元素为空）

！！！下面的题目都是二叉树的对称性问题，区别在于“两两打包”的方式（即把节点加入队列的顺序不同）

对称二叉树

按“打包的第一个节点的左节点”→“打包的第二个节点的右节点”→“打包的第一个节点的右节点”→“打包的第二个节点的左节点”的顺序加入队列，每次循环完成两个打包

相同的树

按“打包的第一个节点的左节点”→“打包的第二个节点的左节点”→“打包的第一个节点的右节点”→“打包的第二个节点的右节点”的顺序加入队列，每次循环完成两个打包

另一棵树的子树

主二叉树进行层序遍历，以每次遍历到的节点为起点，完成“相同的树”的操作（即调用“相同的树”函数，函数返回布尔值，为1就可以直接返回true）

题目：

101. 对称二叉树（简单）

100. 相同的树（简单）

572. 另一棵树的子树（简单）

二叉树的深度问题

思路：用迭代法进行层序遍历

二叉树的最大深度

在二叉树的基础遍历中已有介绍

二叉树的最小深度

在二叉树的基础遍历中已有介绍

题目：

104. 二叉树的最大深度（简单）

111. 二叉树的最小深度（简单）

二叉树的节点个数问题

完全二叉树的节点个数

和二叉树的**层序遍历**完全一致，只是添加了res在每次大while循环中统计个数（每次循环队列中节点个数进行累加）

题目：

72. 完全二叉树的节点个数（简单）

二叉树的高度问题

平衡二叉树

更正：leetcode上对平衡二叉树的定义有误，应该是一个二叉树每个节点的左右两个子树的**高度差绝对值不超过1**

深度与高度的区别：

深度：从上到下；高度：从下到上

二叉树高度问题的方法：递归（不适合用迭代）

本题采用**自底向上的递归法**（类似**后序遍历**），因为自底向上才符合高度“从下到上”的特点

本题递归三部曲：

（1）确定递归的参数和返回值：

递归参数：自然是二叉树的节点

返回值：如果高度差绝对值不超过1，则返回左/右节点当中**更大高度的值**；如果高度差超过1，则说明不是平衡二叉树，用-1做标记

（2）确定递归的终止条件：

如果遍历到节点为空，则返回0

（3）确定递归的逻辑：

按照类似后序遍历左右（**递归操作**）中（**判断操作**）的顺序，如果递归过程中遇到返回值为-1，则上层的递归值全部返回-1；最后主函数中只需判断调用函数的值是

否为-1即可

题目：

110. 平衡二叉树（简单）

二叉树的路径问题

二叉树的所有路径

方法：采用**层序遍历**，维护节点队列和路径队列，仅用一个循环，弹出首节点和首路径，判断首节点是否有左/右节点，有左/右节点（作为else），就将节点加入队列并对应更新首路径；如果左右节点都没有（作为if），就把队列首路径加入结果数组中

*注意：更新路径时要使用StringBuilder类型StringBuffer进行append操作

路径总和

二叉树的路径问题：递归法

递归三要素的应用：

（1）确定递归的参数和返回值

（2）确定递归的终止条件

（3）确定递归的逻辑

路径总和：

（1）递归参数：节点和targetSum（因此**可以直接递归调用主函数**），每遍历一个节点就把targetSum减去该节点的值，作为下一次递归的targetSum；返回值应当为布尔，即判断是否符合条件

***总结：递归函数何时需要返回值？**

case 1：如果需要搜索整棵二叉树且不用处理递归返回值，递归函数就不要返回值（如路径总和II）

case 2：如果需要搜索整棵二叉树但需要处理递归返回值，递归函数就需要返回值（如平衡二叉树）

case 3：如果要搜索其中一条符合条件的路径，那么递归一定需要返回值，因为遇到符合条件的路径了就要及时返回（本题）

（2）终止条件：如果根节点是空节点则返回false，如果访问到叶节点则需判断targetSum是否为0，为0则返回true，不为0则返回false（说明没找到符合条件的路径）

（3）递归逻辑：左节点不为空则对左节点调用递归函数，如果递归返回的布尔值为1，则直接返回true；右节点同理

路径总和II：

（1）递归参数：节点、targetSum、结果数组res和res数组的元素path，其中path用于每次调用递归函数时把访问节点加入。和路径总和一样，每遍历一个节点就把targetSum减去该节点的值，作为下一次递归的targetSum；**递归函数不需要返回值**

（2）终止条件：判断条件和路径总和相同，targetSum为0则把path加入res中，否则返回空

*注意：res加入path的方法是用new ArrayList<>(path），如果直接res.add(path），后续path变化时也会使res的答案有变化，因此需要做类似**拷贝的操作**

（3）递归逻辑：左节点不为空则对左节点调用递归函数，要**注意使用回溯把path的最后一个元素去掉**，否则path是在接续几次递归后的结果基础上添加元素，不是从当前节点开始的，会出问题；右节点同理

题目：

257. 二叉树的所有路径（简单）

112. 路径总和（简单）

113. 路径总和II（中等）