**一、二叉树的遍历**

**1、递归法。**思路：只要用了递归，由于系统压栈的原因，不管是哪种遍历方式，每个节点都会被访问3次，第一次是从父节点来的，第二次是走完左孩子回去，第三次是走完右孩子。接下来要做的就是在哪一次访问的时候对其进行相应的操作（这里是打印），于是有了三种遍历方式。

**2、非递归法。**因为遍历二叉树的单向性，因此一旦到了孩子那就无法直接返回，因此需要自己压栈。比如先访问了左孩子，右孩子就没法访问，所以在当前节点就应该把孩子都压入栈中保存。

**先序遍历：**先将头节点压入栈中，然后以栈是否为空作为循环条件。每次从栈中取出栈顶进行操作，然后将该节点的左右孩子压入栈中(注意先判断存不存在，这里先压右孩子再压左孩子，由于栈是相反的)。这样当栈中为空的时候就全部访问完了。

**中序遍历：**总体思路先把左边走到底，然后走右边。首先从头节点开始，如果节点不为空就压入栈中，而且如果左子树存在就一直压左子树直到底，然后取出栈顶进行相关节点操作，并向右子树方向更新当前节点，继续转循环

**后续遍历：**后续遍历要直接实现比较困难，其顺序是左右中，但是中右左很好实现（和先序遍历一样），于是利用中右左遍历，然后放入辅助栈中，最后得到逆序输出。

**层序遍历：**思路和先序遍历一样，只不过这里用的是队列来存储，先进先出。先将头节点入，然后以队列是否为空作为循环条件。每次从队列中出队一个节点，进行相关操作，然后将该节点的左右孩子压入栈中(注意先判断存不存在，这里先压右孩子再压左孩子，由于栈是相反的)。这样当栈中为空的时候就全部访问完了。

**二、查找后继节点**

找二叉树一个节点的后继节点。（即中序遍历顺序中，每个节点的下一个节点称为其后继节点）

思路1：可以通过给定节点的父节点一直往上找到头节点，然后进行中序遍历，找到给定的节点，那么其下一个节点就是后继节点。

思路2：这个题分两种情况讨论，通过判断如果当前节点有右子树，那么为情况1，如果没有就是情况2.

情况1中，其后继节点就是其右子树的最左的那个节点。

情况2中，往上找其父节点，如果当前节点是其父节点的右子节点就继续往上，直到当节点为父节点的左子节点则父节点为其后继节点，同时还要考虑一种情况就是往上一直找到父节点为null,那么说明该节点为中序遍历的最后一个元素，没有后继节点。

**三、序列化和反序列化**

1、序列化：这个过程其实就是遍历，采用递归进行，区别就是为空的情况也要返回一个特定的值这里给出的是#\_,然后把原先打印的步骤换成字符串的增加过程。

2、反序列化，采用的思路是用队列辅助，首先将字符串解析成数组，用分隔符进行分割，然后把数组入队。接下来的问题就是给一个队列如何建树的问题了。用递归的套路思考，对于每个节点可能性只有两种，为数字和#，碰到#返回空节点，碰到数字就创建新的节点，构建好自己的左子树和右子树（为空不用，直接返回空），然后把自己的节点作为头返回，因此返回值为节点类型，希望从子代获取节点，自己也往上返回节点。

**四、判断是不是平衡二叉树**

属于二叉树遍历的套路问题，若节点存在，则思考几种可能性：左子树平衡？右子树平衡？左右都平衡的状态下当前是否平衡？，因此由于递归可以使得每个节点被访问三次，那么对于当前节点，先让其去左子树传转一圈，然后去右子树转一圈，最后回到当前节点再比较久可以得到结论。从这个角度看，去子树转一圈所要返回的就两个结果：1子树是否平衡 2子树的高度，这样当回到当前节点的是时候可以根据这个返回条件来判断自己的平衡状态，同时也能得出自己的高度，然后以相同的形式返回给上层：1自己是不是平衡2自己的高度。这样就构成了一个递归函数，返回值有两个因此额外定义一个类来包装。

**五、判断是不是二叉搜索树**

属于中序遍历的应用，对整个二叉树进行中序遍历，在原来打印的地方加入判断，如果出现小于前一个值的情况就肯定不是，返回false，如果遍历完都没出现小于情况就是二叉搜索树。问题的关键是前一个节点的保存，采取的方法是设置标志位，第一册从栈中取出的时候让pre等于取出的值，清除首次标志位。之后的循环就都是先判断，如果符合再将当前节点更新给pre.

**六、判断是不是完全二叉树**

属于层序序遍历的应用。关键在于分清楚四种状况，在树种一个节点有四种状态，1左在右在；2左不在右在；3左在右不在；4左不在右不在。层序遍历整个二叉树，如果任意结点出现2说明不是，如果出现3和4那么对于接下来出现的每一个结点都应该是叶节点。因此在程序中设置进行叶节点判断的标志位，一旦有节点出现3/4就进入叶节点判断，如果是情况1就是正常的遍历。

**七、计算一个完全二叉树的节点（时间复杂度小于n,就是不用遍历来做）**

思路：如果一个二叉树是满二叉树的话，其节点数就是一个公式可以解决的，那么对于一个完全二叉树而言，其左右子树如果存在的话必然有以便是满二叉树。

利用这个思路来做。首先找到整个树的层数，看下一共有几层，然后看当前节点的右子树有最深到几层（都是以总的头结点来看的层数），如果等于最深层数，那么左子树就是满二叉树，右子树进行递归，同样进行下一才循环。如果小于最深层数说明右子树是满二叉树，就对左子树进行递归。