**树**

树是由一个**集合以及在该集合上定义的一种关系**构成的

集合中的元素称为**树的结点**，所定义的关系称为**父子关系**

父子关系在树的结点之间建立了一个**层次结构**

树的结点包含**一个数据元素**及**若干指向其子树的若干分支**

在这种层次结构中有一个结点具有特殊的地位，这个结点称为该**树的根结点，**或简称为**树根**

我们可以形式地给出树的递归定义如下∶

树（tree）是n（n>=0）个结点的有限集，它:

1）或者是一棵空树（n=0），空树中不包含任何结点

2）或者是—棵非空树（n>0），此时有且仅有一个特定的称为根（root）的结点;当n>1时，其余结点可分为m（m>0）个互不相交的有限集T1，T2…，Tm，其中每一个本身又是—棵树，并且称为根的子树（sub tree））

**结点的度与树的度**

结点拥有的子树的数目称为结点的 度（Degree）

度为0的结点称为叶子（leaf）或终端结点

度不为0的结点称为非终端结点或 分支结点

除根之外的分支结点也称为内部结点

树内各结点的度的最大值称为树的度

**结点的层次和树的深度**

结点的层次（level））从根开始定义，层次数为1的结点是根结点，其子树的根的层次数为2

树中结点的最大层次数称为树的深度（Depth）或高度

**父亲、儿子、兄弟**

父亲（parent）∶一个结点的直接前驱结点

儿子（child）∶一个结点的直接后继结点

兄弟（sibling）∶同一个父亲结点的其他结点

例：

结点A是结点B、C、D的父亲，结点B、C、D是结点A的孩子

由于结点H、I、J有同一个父结点D，因此它们互为兄弟

**祖先、子孙、堂兄弟**

将父子关系进行扩展，就可以得到祖先、子孙、堂兄弟等关系

结点的祖先是从根到该结点路径上的所有结点

以某结点为根的树中的任一结点都称为该结点的子孙

父亲在同一层次的结点互为堂兄弟

**有序树、m叉树、森林**

如果将树中结点的各子树看成是从左至右是有次序的，则称该树为有序树

若不考虑子树的顺序则称为 无序树

对于有序树，我们可以明确的定义每个结点的第一个孩子、第二个孩子等

直到最后一个孩子

若不特别指明，一般讨论的树都是有序树

树中所有结点最大度数为 m的有序树称为 m 叉树

森林（forest）是m（mk）棵互不相交的树的集合（可以是0棵）

对树中每个结点而言，其子树的集合即为森林

树和森林的概念相近：删去一棵树的根，就得到一个森林;反之，加上一个结点作树根，森林就变为一棵树

**二叉树**

每个结点的度均不超过2的有序树，称为二叉树（binary tree）

与树的递归定义类似，二叉树的递归定义如下∶

二叉树或者是一棵空树，或者是一棵由一个根结点和两棵互不相交的分别称为根的左子树和右子树的子树所组成的非空树

二叉树里每个结点的孩子结点值只能是：0、1、2

**满二叉树**

高度为k并且有2k+1-1个结点的二叉树

在满二叉树中，每层结点都达到最大数，即每层结点都是满的，因此称为满二叉树

**完全二叉树**

若在一棵满二叉树中，在最下层从最右侧起去掉相邻的若干叶子结点，得到的二叉树即为完全二叉树

**二叉树的性质**

性质1∶在二叉树的第i层上最多有2i-1个结点（根是第1层）

性质2∶高度为h的二叉树至多有2h-1个结点

性质3∶对任何一棵二叉树T，如果其终端结点数为n0，度为2的结点数为n2，则n0=n2+1

性质4∶有n个结点的完全二叉树的高度为【log2n】+1，其中【log2n】是向下取整

性质5∶含有n>=1个结点的二叉树的高度至多为n-1;高度至少为【log2n】+1，其中log2n是向下取整

性质6∶如果对一棵有n个结点的完全二叉树的结点进行编号，则对任一结点i（1≤i≤n），有

1. 如果i=1，则结点i是二叉树的根，无双亲;如果 i>1，则其双亲结点PARENT（i）是结点 [i/2]
2. 如果2i>n，则结点i无左孩子;否则其左孩子是结点2i
3. 如果 2i+1>n，则结点i无右孩子;否则其右孩子是结点2i+1

**二叉树的存储结构**

顺序存储结构和链式存储结构

**顺序存储结构**

对于满二叉树和完全二叉树来说，可以将其数据元素逐层存放到一组连续的存储单元中

用一维数组来实现顺序存储结构时，将二叉树中编号为i的结点存放到数组中的第i个分量中

如此根据二叉树性质，可以得到结点i的父结点、左右孩子结点分别存放在2i以及 2i+1分量中

这种存储方式对于满二叉树和完全二叉树是非常合适也是高效方便的

因为满二叉树和完全二叉树采用顺序存储结构既不浪费空间，也可以根据公式很快的确定结点之间的关系n

但是对于一般的二叉树而言，必须用"虚结点"将一棵二叉树补成—棵完全二叉树来存储，否则无法确定结点之间的前驱后续关系，但是这样一来就会造成空间的浪费

二叉树结点以顺序结构存储会造成空间浪费，所以一般不用

**链式存储结构**

设计不同的结点结构可构成不同的链式存储结构

在二叉树中每个结点都有两个孩子，则可以设计每个结点至少包括3个域∶数据域、左孩子域和右孩子域

数据域存放数据元素，左孩子域存放指向左孩子结点的指针，右孩子域存放指向右孩子结点的指针

利用此结点结构得到的二叉树存储结构称为二叉链表

为了方便找到父结点，可以在上述结点结构中增加一个指针域，指向结点的父结点

采用此结点结构得到的二叉树存储结构称为三叉链表

二叉树链式存储结构中的二叉链表

**遍历（Traverse）**

**就是按照某种次序访问树中的所有结点，且每个结点恰好访问一次**

也就是说，按照被访问的次序，可以得到由树中所有结点排成的一个序列

树的遍历也可以看成是人为的将非线性结构线性化

这里的"访问"是广义的，可以是对结点作各种处理，例如输出结点信息、更新结点信息等。在我们的实现中，并不真正的"访问"这些结点，而是得到一个结点的线性序列，以线性表的形式输出

将整个二叉树看做三部分∶根、左子树、右子树

如果规定先遍历左子树、再遍历右子树那么根据根的遍历顺序就有三种遍历方式：左子树 右子树 根

先序/根遍历DLR∶**根** 左子树 右子树

中序/根遍历LDR∶左子树 **根** 右子树

后序/根遍历LRD∶左子树 右子树 **根**

面试题∶已知一棵二叉树的后序遍历的序列为5 4 3 7 6 2 1，中序遍历的序列为4 5 1 3 2 6 7，则其先序遍历的序列是什么?1452367

