**树**

# 树

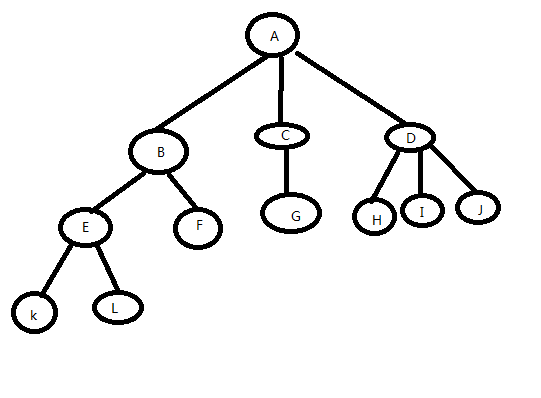
与线性表表示的一一对应的线性关系不同，

树表示的是更为复杂的数据元素之间的**非线性关系**。直观来看，树是以分支关系定义的层次结构。树在客观世界中广泛存在，如人类社会的族谱和各种社会组织机构都可以用树的形象来表示。

简单来说，树表示的是1对多的关系。

树的定义：树(Tree)是n( n>=0 )个节点的有限集合。没有节点的树称为空树，在任意一颗非空树中：

* 有且仅有一个特定的称为根(root)的节点
* 当n>1的时，其余节点可分文m( m>0 )个互不相交的有限集T1，T2，..., Tm，其中每一个集合本身又是一棵树，并且称之为根的子树。



比如，这棵树，其中A是根，其余节点分成3个互不相交的子集

* T1={ B, E, F, K, L}
* T2={ C, G}
* T3={ D, H, I, J}

T1, T2, T3都是根A的子树，且其本身也是一颗树

再比如T1，其根为B，其余结点分别为两个互不相交的子集：

* T11={E, K, L}
* T12={F}

T11和T12都是T1的子树。

注意，树的定义本身是一个递归定义，即在树的定义中又用到树的概念。

在树中，还有一些基本术语：

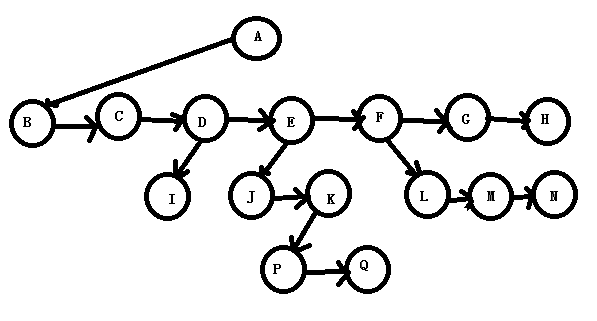
* 一个节点的子树的根，称为该节点的**儿子**（child），相应的该节点称为子树的根的**父亲**（parent）
* 没有儿子的节点称为**树叶**(leaf)
* 具有相同父亲的节点互为**兄弟**（sibling）
* 用类似的方法可以定义**祖父** (grandparent) 和**孙子**（grandchild）的关系。
* 节点的**层级**（level）从根开始定义，根为第一层，根的孩子为第二层。若某节点在第i层，则其孩子就在i+1层。
* 从节点n1 到 nk 的**路径**（path）定义为节点 n1 n2 … nk 的一个序列，使得对于 1 <= i < k，节点 ni是 ni+1 的父亲。这条**路径**的**长**是为该路径上边的条数，即 k-1。从每一个节点到它自己有一条长为 0 的路径。注意，在一棵树中从根到每个节点恰好存在一条路径。
* 对任意节点ni，ni的**深度**为从根到ni的唯一路径的长。因此，根的深度为0。ni 的**高**是从ni 到一片树叶的最长路径的长。因此，所有树叶的高都为0。一颗树的高等于它根的高。一颗树的深度等于它最深的树叶的深度; 该深度总是等于这棵树的高。
* 如果存在从n1到n2的一条路径，那么n1是n2的一位**祖先** (ancestor)，而n2是n1的一个**后裔**（descendant）。如果n1 != n2，那么n1是n2的**真祖先**（proper ancestor）而n2是n1的**真后裔**（proper descendant）

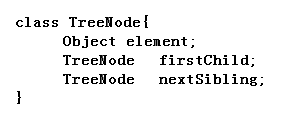
性质：如果一棵树有n个节点，那么它有n-1边。

# 树的实现

实现树的一种方法可以是在每一个节点除数据外还要有一些链，使得该节点的每一个儿子都有一个链指向它。然而，由于每个节点的儿子数可以变化很大，并且事先不知道，因此，在数据结构中建立到各子节点的直接链接是不可行的，因为这样会产生太多浪费的空间。实际上解决办法很简单：将每个节点的所有儿子都放在树节点的链表中。

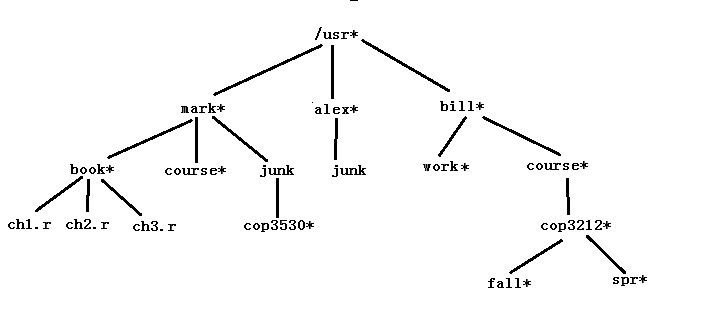
图中向下的箭头指向firstChild的链，而水平箭头是指向nextSibling的链。因为null链太多了，所以没有把他们画出来。





# 树的应用

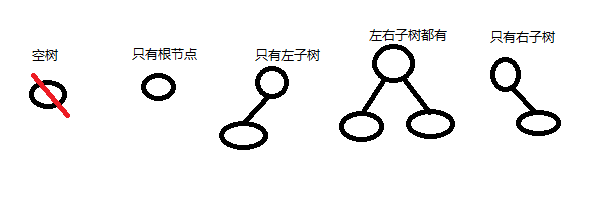
树的应用很多。流行的用法之一是包括UNIX和DOS在内的许多常用操作系统中的目录结构。下图是UNIX文件系统中一个典型的目录。



# 二叉树

二叉树是一棵树，它的特点是每个节点至多有两颗子树，并且，二叉树有左右之分，其次序不能颠倒（有序）。

二叉树有以下5种基本形态：



二叉树具有以下重要特性：

* 二叉树，在第i层至多有2i-1个节点 （主要是在理论上使用的比较多）
* 层次为k的二叉树至多有2k-1个节点
* 对任何一颗二叉树T，如果其叶子节点数为n0 , 度为2的节点数为n2，则n0 = n2 + 1
* 具有n个节点的完全二叉树，根节点的深度为log2(n)
* 如果对一颗有n个节点的完全二叉树的节点按层序编号，则对任意一结点，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parent(i)的节点， i/2
2. 如果2i > n，则结点i无左孩子（结点i为叶子结点）；否则其左孩子lchild(i)是节点 2i。
3. 如果2i+1 > n，则结点i无右孩子，否则，其右孩子rchild(i)是结点2i+1。

## 二叉树的存储结构

同线性表，二叉树的存储结构也分为顺序映像和非顺序映像。

### 二叉树的顺序映像（完全二叉树）

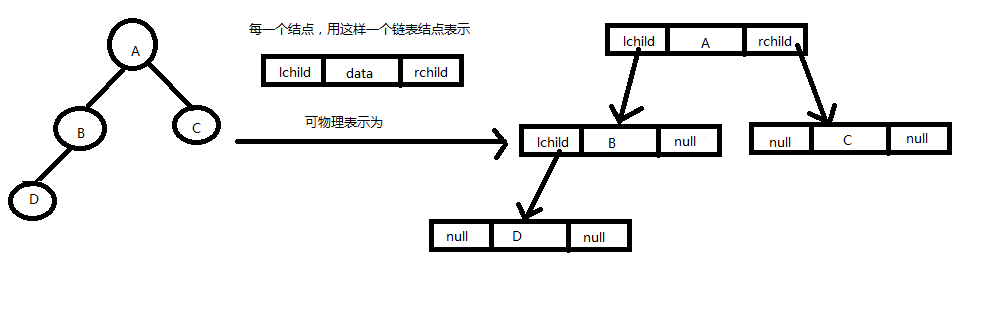
二叉树的顺序映像，还是用数组来存储数据元素，那么如何表示数据元素之间的关系呢？

约定在数组中，自上而下，自左而右的存储完全二叉树上的节点元素，即将完全二叉树上编号为i的结点元素，存储在该数组的下标为i的索引中。

从存储的角度来看，这种顺序存储结构，仅适用于完全二叉树。因为在最坏的情况下，一个深度为k且只有k个结点的单支树( 树中不存在度为2的结点 )，却需要长度为2k-1的一维数组。

### 二叉树的非顺序映像

二叉树的非顺序映像，是以链表的形式，存储数据元素以及数据元素之间的关系。但是由于数据元素的关系变得更加复杂，因此，链表结点的结构，与线性表中链表的结点有有所不同



### 二叉树的前中后遍历

在二叉树的很多应用场景中，常常要求在树中查找具有某种特征的结点，或者对树中结点逐一进行某种处理，这就提出了遍历二叉树的问题：

即如何按照某条搜索路径巡访树中的每个节点，使得每个节点均被访问一次，而且仅被访问一次。

遍历对线性结构来说，是一个容易解决的问题，但是对于二叉树则不然，由于二叉树是一种非线性结构，每个结点都有可能有两颗子树，因而需要寻找一种规律，以便使二叉树上的节点能排列在一个线性队列上，从而便于遍历。

我们再来回顾一下二叉树的递归定义，可知，二叉树由3部分组成：根，左子树和右子树，因此如果能遍历这三个部分，便是遍历了整个二叉树。

假如以L, D, R分别表示遍历左子树，访问根结点和遍历右子树，则可能有DLR, LDR, LRD, DRL, RDL, RLD这六种二叉树的遍历方案。

若限定，先左后右，则只有前3种情况，分别称之为先（根）序遍历，中（根）序遍历，和后（根）序遍历。三者的区别主要在于访问根结点的先后顺序

### 二叉树的广度优先遍历

这个没啥好额外说的，主要是让大家了解BFS的模板 。

### 二叉树的建树

如果对于某二叉树，已知按某种遍历方式的遍历该树得到的序列。那么，我们可以根据此序列，建立一颗与之对应的二叉树的物理表示。

但这种遍历，必须把为空的节点也包含在内，才可以完成。

还可以以另外一种方式建树，就是分别根据对一颗二叉树的中序遍历序列和先序遍历序列，或者根据中序遍历序列和后序遍历序列，来创建这颗二叉树。

