6.1.4　树的性质

性质 1　树中的结点个数等于树中所有结点的度数之和再加 1。

【证明】假设树中的结点个数为 n，分支总数为 B，若 D_i 表示第 i 个结点的度数，则可得到所有结点的分支数等于所有结点的度数之和，即：

$$B=\sum_{i=1}^{n}D_i$$

根据树的定义，在一棵树中，除根结点外，其余的每个结点都有且仅有一个前驱结点。也就是说，每个结点均与指向它的分支一一对应，所以除树根结点之外的结点数等于所有结点的分支数，即 $B=n-1$。因此可得树中的结点个数等于所有结点的分支数之和再加 1，即 $B=n+1$，也就是树中的结点个数等于所有结点的度数之和再加 1，即：

$$n=\sum_{i=1}^{n}D_i+1$$

性质 2　度为 m 的树中第 i 层上至多有 m^{i-1} 个结点（$i\geq 1$）。

【证明】采用数学归纳法证明。

（1）对于第一层，因为树中的第一层上只有一个结点，即整个树的根结点，而由 $i=1$ 代入 m^{i-1}，得 $m^{i-1}=m^{1-1}=1$，也同样得到只有一个结点，显然结论成立。

（2）假设对于第（$i-1$）（$i>1$）层命题成立，即度为 m 的树中第（$i-1$）层上至多有 m^{i-2} 个结点，根据树的度的定义，度为 m 的树中每个结点至多有 m 个孩子结点，所以第 i 层上的结点数至多为第（$i-1$）层上结点数的 m 倍，即至多为 $m^{i-2}\times m=m^{i-1}$ 个，这与命题相同，故命题成立。

6.1.5　树的存储结构

树的存储要求既要存储结点的数据元素本身，又要存储结点之间的逻辑关系。由于树中各个结点的度可能不同，因此在存储过程中常遇到以下两类问题。

（1）根据树的度分配结点所占空间，虽然可保证结点同构，但会造成存储空间的浪费；

（2）根据各个结点的度来分配结点所占空间，虽然节省了存储空间，但造成整棵树的结构不统一，后续计算复杂度增高。

因此，虽然树的存储方式有很多种，既可以采用顺序存储结构，也可以采用链式存储结构。但无论采用何种存储方式，都要求存储结构应结合实际应用背景，根据问题的特点和所需进行的操作进行适当选用，以提高执行效率。以下将介绍树的几种存储结构。

1. 树的顺序存储结构

由树的定义可知，除根结点之外，树中的每个结点都有唯一的双亲，根据这一特点，可以用一组连续的存储空间，即一维数组来存储树中的各个结点，数组中的一个数据元素表示树中的一个结点，数据元素为结构体类型，其中包括结点本身的信息以及其双亲结点在数组中的位置信息，树的这种存储方法称为**双亲表示法（Parent Express）**。其类定义描述如下。

```
//-------------树的双亲表示法-----------
const int MaxSize = 100;   //可存储的最多结点个数
struct PNode               //树中结点的数据类型
{
    ElemType data;         //树中结点本身的数据信息
```