第 5 章　树与二叉树

到目前为止，已经学习了线性结构、简单的多维数组和表结构。这些数据结构一般不适合于描述具有分支结构的数据。在这种数据之间可能有祖先-后代、一般-特殊、整体-部分等分支的关系。本章讨论的树结构则是以分支关系定义的层次结构，是一类重要的非线性数据结构，在计算机领域有着广泛应用。例如，在文件系统和数据库系统中，树是组织信息的重要形式之一；在编译系统中，树用来表示源程序的语法结构；在算法设计与分析中，树还是刻画程序动态性质的工具。

5.1　树的基本概念

树结构广泛存在于现实世界，例如，家族的家谱、公司的组织机构、书的章节等。在计算机应用中，最为人们熟悉的就是磁盘中的文件夹，即文件目录。它包含文件和文件夹。

5.1.1　树的定义和术语

1. 树的定义

为了完整地建立有关树的基本概念，以下给出两种树的定义，即自由树和有根树。

自由树(free tree)　一棵自由树 T_f 可定义为一个二元组 $T_f=(V,E)$，其中 $V=\{v_1,\cdots,v_n\}$ 是由 $n(n>0)$ 个元素组成的有限非空集合，称为顶点(vertex)集合，$v_i(1\leqslant i\leqslant n)$ 称为顶点。$E=\{(v_i,v_j)\mid v_i,v_j\in V,1\leqslant i,j\leqslant n\}$ 是由 $n-1$ 个元素间关系组成的序对集合，称为边集合，属于 E 的序对 (v_i,v_j) 称为边(edge)或分支(branch)。E 使得 T_f 成为一个连通图，参看图 5-1。

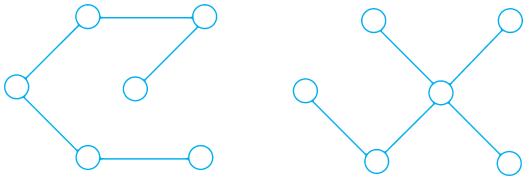


图 5-1　自由树

有关自由树的研究是图论讨论的主要内容之一，本书不进行讨论。本章讨论的树是有根树，它们的顶点统一称为结点(node)。

有根树(rooted tree)　一棵有根树 T 简称为树，定义为 $n(n>0)$ 个结点的有限集合。记作：

$$T=\{r,T_1,T_2,\cdots,T_n\}$$

其中，r 是 T 的根结点(root)。$T_1,T_2,\cdots,T_m$ 是除 r 之外其他结点构成的互不相交的 $m(m\geqslant 0)$ 个子集合，每个子集合也是一棵树，称为根的子树(subtree)。

每棵子树的根结点有且仅有一个直接前驱(即它的上层结点),但可以有 0 个或多个直接后继(即它的下层结点)。m 称为 r 的分支数。

图 5-2 给出树的逻辑表示,它形如一棵倒长的树。图 5-2(a)是空树,一个结点也没有。图 5-2(b)是只有一个根结点的树,它的子树为空。图 5-2(c)是有 13 个结点的树,其中 A 是根结点,它一般都画在树的顶部。其余结点分成 3 个互不相交的子集:$T_1=\{B,E,F,K,L\}$,$T_2=\{C,G\}$,$T_3=\{D,H,I,J,M\}$。它们都是根结点 A 的子树。再看 T_1,它的根是 B,其余结点又分成 2 个互不相交的子集 $T_{11}=\{E,K,L\}$,$T_{12}=\{F\}$,它们是 T_1 的子树。

由此可知,树的定义是一个递归的定义,即树的定义中又用到了树的概念。

除了图 5-2 所描述的逻辑表示外,在计算机系统和其他领域还有几种表示方法,参看图 5-3。图 5-3(a)是树的目录结构表示;图 5-3(b)是树的集合文氏图表示;图 5-3(c)是树的凹入表表示,图 5-3(d) 是树的广义表表示。

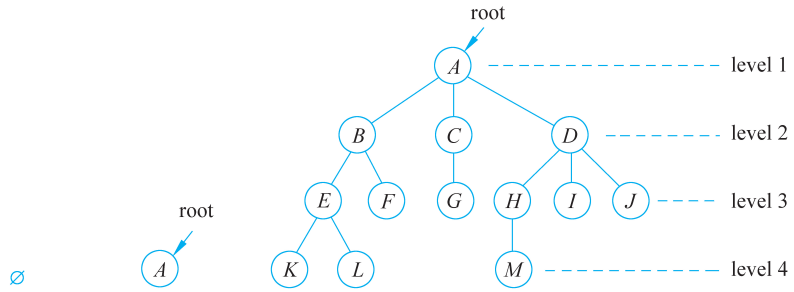


图 5-2 树的示意图

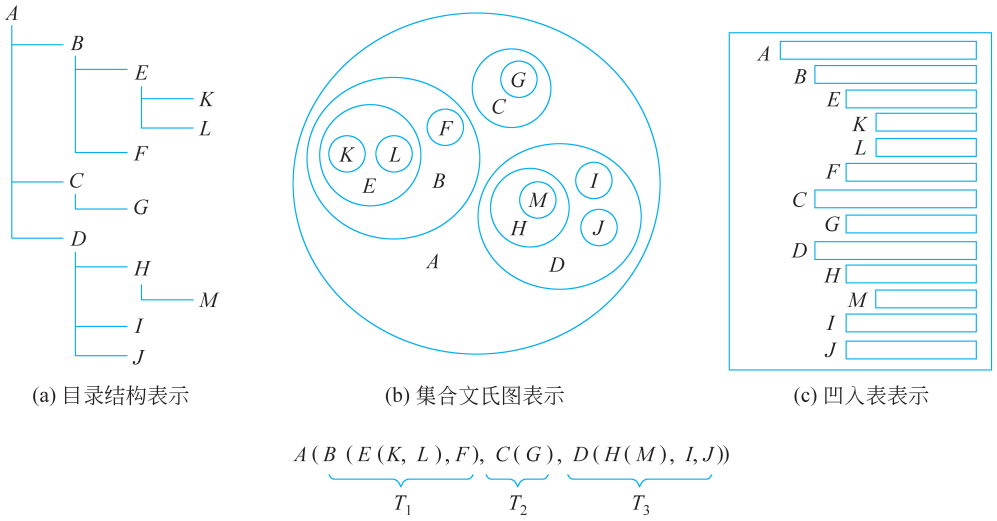


图 5-3 树的其他表示方法

2. 树的基本术语

结点(node):它包含数据元素的值及指向其他结点的分支指针。例如在图 5-2(c) 中的

树总共 13 个结点。为方便起见，每个数据元素用单个字母表示。

结点的度（degree）：是结点所拥有的子树棵数。例如在图 5-2(c)所示的树中，根 A 的度为 3，结点 E 的度为 2，结点 K，L，F，G，M，I，J 的度为 0。

叶结点（leaf）：即度为 0 的结点，又称为终端结点。例如在图 5-2(c)所示的树中，{K，L，F，G，M，I，J} 构成树的叶结点的集合。

分支结点（branch）：除叶结点外的其他结点，又称为非终端结点。例如在图 5-2(c)所示的树中，A，B，C，D，E，H 就是分支结点。

子女结点（child）：若结点 x 有子树，则子树的根结点即为结点 x 的子女。例如在图 5-2(c)所示的树中，结点 A 有 3 个子女，结点 B 有 2 个子女，结点 L 没有子女。

双亲结点（parent）：又称为父结点。若结点 x 有子女，它即为子女的双亲结点。例如在图 5-2(c)所示的树中，结点 B，C，D，E 有一个双亲结点，根结点 A 没有双亲结点。

兄弟结点（sibling）：同一双亲结点的子女结点互称为兄弟结点。例如在图 5-2(c)所示的树中，结点 B，C，D 为兄弟结点，E，F 也为兄弟结点，但 F，G，H 不是兄弟结点。

祖先结点（ancestor）：从根结点到该结点所经分支上的所有结点。例如在图 5-2(c)所示的树中，结点 L 的祖先结点为 A，B，E，L。不包括 L 的祖先结点称为 L 的真祖先结点。

子孙结点（descendant）：某一结点的子女，以及这些子女的子女都是该结点的子孙结点。例如在图 5-2(c)所示的树中，结点 B 的子孙结点为 B，E，F，K，L。不包括 B 的子孙结点称为 B 的真子孙结点。本书下文中如果不加特别说明，所谓“子孙”都是指真子孙。

结点间的路径（path）：树中任一结点 v_i 经过一系列结点 v_1，v_2，…，v_k 到 v_j，其中(v_i，v_1)，(v_1，v_2)，…，(v_k，v_j)是树中的分支，则称 v_i，v_1，v_2，…，v_k，v_j 是 v_i 与 v_j 间的路径。

结点的深度（depth）：即结点所处层次(level)，简称结点的层次，即从根到该结点的路径上的分支数加 1。例如在图 5-2(c)所示的树中，根结点在第 1 层，它的子女在第 2 层。树中任一结点的层次为它的父结点的层次加 1。①

结点的高度（height）：空树的高度为 0，叶结点的高度为 1，非叶结点的高度等于它的两个子女结点高度相比，取其大值加 1。

树的深度（depth）：树中距离根结点最远的结点所处层次即为树的深度。空树的深度为 0，只有一个根结点的树的深度为 1，图 5-2(c)所示的树的深度为 4。

树的高度（height）：树的高度等于根结点的高度。需要注意的是：树的高度与树的深度的值相等，但高度与深度计算的方向不同。

树的宽度（width）：统计树中每一层结点个数，取其最大者作为树的宽度。

树的度（degree）：树中结点的度的最大值。例如，图 5-2(c)所示的树的度为 3。

有序树（ordered tree）：树中结点的各棵子树 T_0，T_1，…是有次序的，即为有序树。其中，T_1 称为根的第 1 棵子树，T_2 称为根的第 2 棵子树，……。

无序树（unordered tree）：树中结点的各棵子树之间的次序是不重要的，可以互相交换位置。

森林（forest）：是 $m(m\geqslant 0)$棵树的集合。在自然界，树与森林是两个不同的概念，但在

① 某些教材设定根结点在第 0 层，实用中确实需要把根结点定为第 0 层。本书基于学习，还是按照传统，把根结点定为第 1 层。

数据结构中，它们之间的差别很小。删去一棵非空树的根结点，树就变成森林（不排除空的森林）；反之，若增加一个根结点，让森林中每一棵树的根结点都变成它的子女，森林就成为一棵树。

想想看 根据树的定义，请验证以下一些结论：

- 树中的分支条数等于结点个数减 1。
- 树中任意两个结点之间存在唯一的一条路径。
- 结点间路径的长度等于路径上的分支条数。
- 树中每对结点至少存在一个共同祖先结点。
- 在一对结点的所有共同祖先结点中，深度最大的共同祖先结点称为最低共同祖先结点。每一对结点间的最低共同祖先结点必存在且唯一。

5.1.2 树的基本操作

```
(1) void InitTree( Tree& T );                          //创建一棵树并初始化
```

先决条件：无。

操作结果：建立一棵根指针为 T 的空树。

```
(2) void ClearTree ( TreeNode * & T ) :                //删除一棵树
```

先决条件：树非空。

操作结果：将树中所有结点释放，将树清空。

```
(3) position FirstChild ( Tree& T, position p );       //求结点 p 第一个子女地址
```

先决条件：结点 p 是树的结点。

操作结果：返回结点 p 的第一个子女结点地址，若无子女结点则函数返回 0。

```
(4) position NextSibling ( Tree& T, position p );      //求结点 p 下一兄弟结点地址
```

先决条件：结点 p 是树的结点。

操作结果：返回结点 p 的下一个兄弟结点地址，若无下一兄弟结点则返回 0。

```
(5) position Parent( Tree& T, position p );            //求结点 p 的双亲结点地址
```

先决条件：结点 p 是树的结点。

操作结果：返回结点 p 的双亲结点地址，若结点 p 为根则返回 0。

```
(6) int InsertChild(Tree& T, position p, TElemType x);  //在结点 p 下插入新子女结点
```

先决条件：树非空且结点 p 为树的结点。

操作结果：在结点 p 下插入值为 x 的新子女结点，若插入失败，函数返回 0，否则函数返回 1。

```
(7) int DeleteChild ( Tree& T, position p, int i );    //删除结点 p 的第 i 个子结点
```

先决条件：树非空且结点 p 是树的结点。

操作结果：删除结点 p 的第 i 个结点，若删除失败，则函数返回 0，否则函数返回 1。

```
(8) void DeleteSubTree ( Tree& T, position t );     //删除以 t 为根结点的子树
```

先决条件：t 是树的结点。

操作结果：删除以 t 为根的子树的全部结点且 t 置为空。

```
(9) void Traversal ( Tree& T, position p );         //遍历以 p 为根的子树
```

先决条件：树非空且结点 p 是树的结点。

操作结果：遍历以结点 p 为根的子树。

5.2 二 叉 树

树的子女-兄弟链表表示实际上就是把树转化为二叉树表示来实现的。其实，二叉树(Binary Tree)在很多应用中都得到实用，是一类重要的数据类型。

5.2.1 二叉树的概念

一棵二叉树是结点的一个有限集合，该集合或者为空，或者是由一个根结点加上两棵分别称为左子树和右子树的、互不相交的二叉树组成。

$$T=\begin{cases}\varnothing, & n=0\\ \{r,T_L,T_R\}, & n>0\end{cases}$$

二叉树的特点如下：

(1) 每个结点最多有两个子女结点，分别称为该结点的左子女结点和右子女结点。就是说，在二叉树中不存在度大于 2 的结点，且二叉树的子树有左、右之分，其子树的次序不能颠倒。

(2) 二叉树的定义是递归的。根结点的子树 T_L、T_R 仍是二叉树，到达空子树时递归的定义结束。许多基于二叉树的算法都利用了这个递归的特性。

(3) 二叉树可能有 5 种不同的形态，参看图 5-4。图 5-4(a)表示一棵空二叉树。图 5-4(b)是只有根结点的二叉树，根的左子树和右子树都是空的。图 5-4(c)是根的右子树为空的二叉树，图 5-4(d)是根的左子树为空的二叉树，图 5-4(e)是根的两棵子树都不为空的二叉树。一棵二叉树的根的子树还可以是这样 5 种形态中的一种。

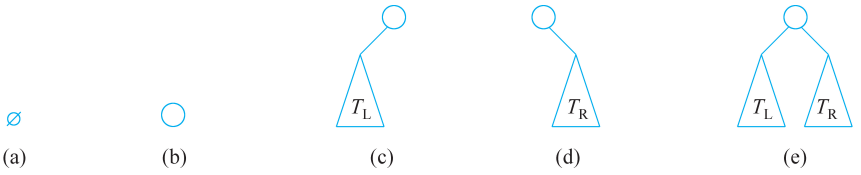


图 5-4 二叉树的 5 种不同形态

(4) 关于树的术语对于二叉树都适用，但二叉树不是树。理由是：

- 树在图论中被视为用 $n-1$ 条边连接 n 个顶点的特殊图。图的顶点集合非空，故树

的顶点非空。图论中另外定义了 N 叉树，它可以是空树，二叉树属于 N 叉树。

- 非空二叉树有根，根结点的子树有左、右之分，次序不能颠倒；若其中某一棵子树为空，另一棵子树也需保持左、右之分。树可以没有根（自由树）；即使有根，其子树也没有这种区分。

想想看　二叉树的叶结点无子女结点，是否可称它无子树？

5.2.2　二叉树的性质

二叉树具有如下性质：

性质 1　在二叉树的第 $i(i\geqslant 1)$ 层最多有 2^{i-1} 个结点。

【证明】　用归纳法：当 $i=1$ 时，非空二叉树在第 1 层只有一个根结点，$2^{1-1}=2^0=1$，结论成立；现假定对于所有的 j，$1\leqslant j<i$，结论成立，即第 j 层上至多有 2^{j-1} 个结点，由于二叉树每个结点最多有 2 个子女，因而第 $j+1$ 层上的最大结点数最多为第 j 层上最大结点数的 2 倍，即 $2\times 2^{j-1}=2^j$ 个结点，性质成立。

性质 2　深度为 $k(k\geqslant 0)$ 的二叉树最少有 k 个结点，最多有 2^k-1 个结点。

【证明】　因为每一层最少要有 1 个结点，因此，最少结点数为 k。$k=0$ 是空二叉树的情形，此时一个结点也没有，$2^0-1=0$，结论成立。$k\geqslant 1$ 是非空二叉树情形，具有层次 $i=1,2,\cdots,k$。根据性质 1，第 i 层最多有 2^{i-1} 个结点，则整个二叉树中所具有的最大结点数为：

$$\sum_{i=1}^{k}(\text{第 } i \text{ 层的最大结点数})=\sum_{i=1}^{k}2^{i-1}=2^k-1$$

性质 3　对于任一棵非空二叉树，若其叶结点数为 n_0，度为 2 的非叶结点数为 n_2，则：

$$n_0=n_2+1。$$

【证明】　设二叉树中度为 1 的结点数为 n_1，因为二叉树只有度为 0、度为 1 和度为 2 的结点，所以树中结点总数为 $n=n_0+n_1+n_2$。再看二叉树中边（分支）条数 e。因为二叉树中除根结点没有双亲结点，进入它的边数为 0 之外，其他每一结点都有一个且仅有一个双亲结点，进入它们的边数均为 1，故二叉树中总的边数为 $e=n-1=n_0+n_1+n_2-1$。又由于每个度为 2 的结点发出 2 条边，每个度为 1 的结点发出 1 条边，度为 0 的结点发出 0 条边，因此总的边数为 $e=2n_2+n_1$。将两个关于边的等式等同起来，有 $n_0+n_1+n_2-1=2n_2+n_1$，消去 n_1 和一个 n_2，得 $n_0-1=n_2$，即 $n_0=n_2+1$。结论成立。

其他一些性质是有关某些特殊二叉树的。为此，先定义两种特殊的二叉树。

满二叉树（full binary tree）　深度为 k 的满二叉树是有 2^k-1 个结点的二叉树。在满二叉树中，每一层结点都达到了最大个数。除最底层结点的度为 0 外，其他各层结点的度都为 2。图 5-5(a)给出的就是高度为 4 的满二叉树。

完全二叉树（complete binary tree）　如果一棵具有 n 个结点的深度为 k 的二叉树，它的每一个结点都与高度为 k 的满二叉树中编号为 $1\sim n$ 的结点一一对应，则称这棵二叉树为完全二叉树。图 5-5(b)给出的就是深度为 4 的完全二叉树。其特点是：上面从第 1 层到第 $k-1$ 层的所有各层的结点数都是满的，仅最下面第 k 层或是满的，或从右向左连续缺若干结点。

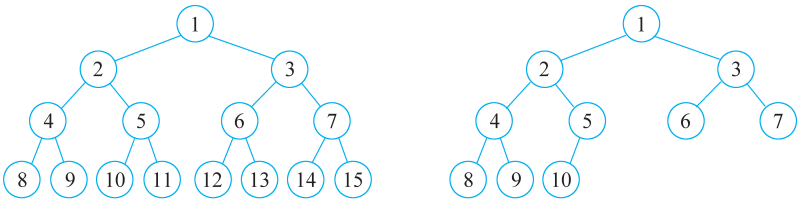


图 5-5 两种特殊的二叉树

想想看 若完全二叉树有 n 个结点，当 n 为奇数时，$n_1=0$，$n_2=\lfloor n/2 \rfloor$，$n_0=n_2+1$；当 n 为偶数时，$n_1=1$，$n_0=n/2$，$n_2=n_0-1$。其中的道理是什么？

性质 4 具有 n 个结点的完全二叉树的深度为 $\lceil \log_2(n+1) \rceil$。

【证明】 根据二叉树的性质 2，深度为 k 的完全二叉树最多结点个数 $n \leqslant 2^k-1$，最少结点个数 $n>2^{k-1}-1$，因此：

$$2^{k-1}-1<n \leqslant 2^k-1$$

移项得 $2^{k-1}<n+1 \leqslant 2^k$

取对数 $k-1<\log_2(n+1) \leqslant k$

因为 $\log_2(n+1)$ 介于 $k-1$ 和 k 之间且不等于 $k-1$，深度又只能是整数，因此有：

$$k=\lceil \log_2(n+1) \rceil$$

结论成立。

注意，此性质针对完全二叉树给出。但对如图 5-6 所示的二叉树也适合。这种二叉树称为丰满树或理想平衡树（perfect balance tree），其第 1 层到第 $k-1$ 层也是满的，第 k 层的结点不一定集中在最左边，可能分布在该层的各处。

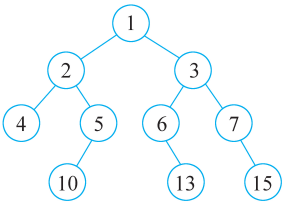


图 5-6 丰满树

想想看 有的教科书定义 $k=\lfloor \log_2 n \rfloor+1$，这与上述定义在 $n>0$ 时等效，但 $n=0$ 时是否同样有效？

性质 5 如果将一棵有 n 个结点的完全二叉树自顶向下，同一层自左向右连续给结点编号 1，2，3，…，n，然后按此结点编号将树中各结点顺序地存放于一个一维数组中，并简称编号为 i 的结点为结点 i（$1 \leqslant i \leqslant n$），参看图 5-5(b)。则有以下关系：

(1) 若 $i=1$，则结点 i 为根，无双亲结点；若 $i>1$，则结点 i 的双亲结点为结点 $\lfloor i/2 \rfloor$。

(2) 若 $2i<=n$，则结点 i 的左子女为结点 $2i$。

(3) 若 $2i+1<=n$，则结点 i 的右子女为结点 $2i+1$。

(4) 若结点编号 i 为奇数，且 $i!=1$，它处于右兄弟位置，则它的左兄弟为结点 $i-1$。

(5) 若结点编号 i 为偶数，且 $i!=n$，它处于左兄弟位置，则它的右兄弟为结点 $i+1$。

(6) 结点 i 所在层次为 $\lfloor \log_2 i \rfloor+1$。

想想看 如果结点编号从 0 开始，则：

- 结点 i（$1 \leqslant i \leqslant n-1$）的双亲结点为结点 $\lfloor (i-1)/2 \rfloor$，结点 0 无双亲结点；
- 分支结点中编号最大的是结点 $\lfloor (n-2)/2 \rfloor$ 或结点 $\lfloor n/2 \rfloor-1$；

- 若 $i \leqslant \lfloor (n-2)/2 \rfloor$，则结点 i 的左子女为 $2i+1$；若 $i \leqslant \lfloor (n-3)/2 \rfloor$，则结点 i 的右子女为 $2i+2$；
- 若 i 为偶数且大于 0，则结点 i 有左兄弟结点 $i-1$；若 i 为奇数且 $i \leqslant n-2$，则结点 i 有右兄弟结点 $i+1$；
- 结点 $i(0 \leqslant i \leqslant n-1)$ 在第 $\lfloor \log_2(i+1) \rfloor + 1$ 层。

想想看 对于只有度为 0 和度为 2 的严格二叉树，其最大高度是多少？最小高度是多少？如何称为严格二叉树？

5.2.3 二叉树的主要操作

```
(1) void InitBTree ( BiTNode * & T );                    //初始化一棵根为 T 二叉树
```

先决条件：无。
操作结果：建立一棵根指针为 T 的空二叉树。

```
(2) void ClearBTree ( BiTNode * & T ) :                  //删除一棵树
```

先决条件：二叉树非空。
操作结果：将根为 T 的二叉树中所有结点释放，将二叉树清空。

```
(3) BiTNode * LeftChild ( BiTNode * p );                 //求结点 p 左子女结点地址
```

先决条件：结点 p 是二叉树的结点。
操作结果：返回结点 p 的左子女结点地址，若无左子女则函数返回 NULL。

```
(4) BiTNode * RightChild ( BiTNode * p );                //求结点 p 右子女结点地址
```

先决条件：结点 p 是二叉树的结点。
操作结果：返回结点 p 的右子女结点地址，若无右子女则函数返回 NULL。

```
(5) BiTNode * Parent ( BiTNode * p );                    //求结点 p 的父结点地址
```

先决条件：结点 p 是二叉树的结点。
操作结果：返回结点 p 的双亲结点地址，若结点 p 为根则返回 NULL。

```
(6) int GetData ( BiTNode * p, TElemType& x );           //返回结点 p 中存放的值
```

先决条件：二叉树非空且结点 p 是二叉树的结点。
操作结果：结点 p 的数据通过 x 返回且函数返回 1，否则返回 0。

```
(7) void CreateBinTree ( BiTNode * & T );                //建立以 T 为根的二叉树
```

先决条件：二叉树为空且输入序列按前序排列。
操作结果：建立以结点 T 为根的二叉树。

```
(8) void PrintBinTree ( BiTNode * & T );                 //输出以结点 T 为根的子树
```

先决条件：二叉树非空。

操作结果：按照广义表的形式输出以结点 T 为根的子树。

```
(9) int Height ( BiTNode * & T );                    //求以结点 T 为根的子树高度
```

先决条件：无。

操作结果：计算以 T 为根的子树的高度，空树返回 0。

```
(10) void PreOrder ( BiTNode * T );                  //前序遍历
```

先决条件：无。

操作结果：按照前序顺序访问二叉树 T 的所有结点且每个结点仅访问一次。

```
(11) void InOrder ( BiTNode * T );                   //中序遍历
```

先决条件：无。

操作结果：按照中序顺序访问二叉树 T 的所有结点且每个结点仅访问一次。

```
(12) void PostOrder ( BiTNode * T );                 //后序遍历
```

先决条件：无。

操作结果：按照后序顺序访问二叉树 T 的所有结点且每个结点仅访问一次。

```
(13) void LevelOrder ( BiTNode * T );                //层次序遍历
```

先决条件：无。

操作结果：按照层次序顺序访问二叉树 T 的所有结点且每个结点仅访问一次。

5.3 二叉树的存储表示

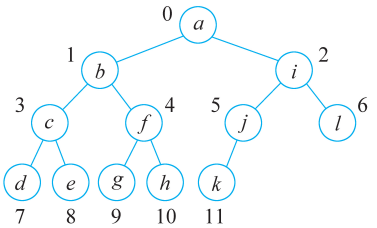
二叉树的存储表示有两种：顺序存储表示和链接存储表示。

5.3.1 二叉树的顺序存储表示

在数据处理过程中二叉树的大小和形态不发生剧烈动态变化的场合，适宜采用顺序存储方式来表示二叉树。用顺序存储方式存储二叉树结构，就是将二叉树的数据元素存储在一组连续的存储单元之内。这种方式可以用 C 的一维数组来描述，用数组元素的下标为索引，随机存取二叉树的结点。

1. 完全二叉树的顺序存储表示

设有一棵完全二叉树，如图 5-7(a)所示，对它所有的结点按照层次次序自顶向下，同一层自左向右顺序编号 1,2,…,n，就得到一个结点的顺序(线性)序列。按这个线性序列，把这棵完全二叉树放在一个一维数组中，如图 5-7(b)所示。



(a) 树状表示

0	1	2	3	4	5	6	7	8	9	10	11
a	b	i	c	f	j	l	d	e	g	h	k

(b) 数组存储

图 5-7 完全二叉树的数组表示

在数组 data[i] 中存放编号为 i($i \geqslant 0$)的完全二叉

树的结点。采用这种方式，可以利用完全二叉树的性质5，从一个结点的编号推算出它的双亲结点及子女、兄弟等结点的编号，从而在数组中找到这些结点。

2. 一般二叉树的顺序存储表示

设有一棵一般的二叉树，如图5-8(a)所示，需要将它存放在一个一维数组中。为了能够简单地找到某一个结点的上下左右的关系，也必须仿照完全二叉树那样，对二叉树的结点进行编号。然后按其编号将它放到数组中去，如图5-8(b)所示。在编号时，如遇到空子树，应在编号时假定有此子树进行编号，而在顺序存储时当作有此子树那样把位置留出来。这样才能反映二叉树结点之间的相互关系，由其存储位置找到它的双亲结点及子女、兄弟结点的位置。但这样做有可能会消耗大量的存储空间。例如，图5-9给出的单支二叉树，要求一个可存放31个结点的一维数组，但只在第0,2,6,14,30这几个位置存放有结点数据，其他大多数结点空间都空着，又不能压缩到一起，造成很大空间浪费。

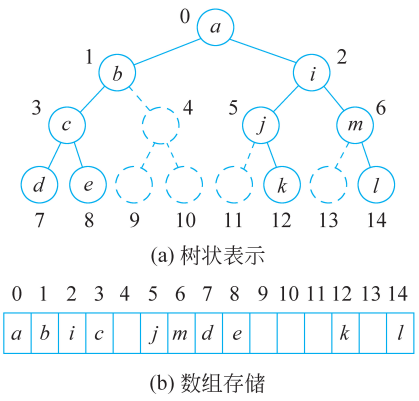


图5-8 一般二叉树的数组表示

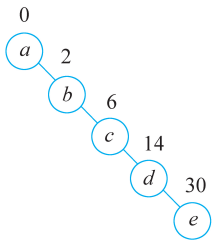


图5-9 单支二叉树

3. 二叉树顺序存储结构的类型定义

程序5-1 二叉树的顺序存储结构定义(存放于SqBTree.h中)

```
#include<stdio.h>
#define maxSize 127                          //默认存储大小,按满二叉树定义
typedef char DataType;                       //假设元素数据类型为 char
typedef struct {
    DataType data[maxSize];                  //存储数组
    int n;                                   //当前结点个数
} SqBTree;
```

顺序存储结构对于一般二叉树不太适用。它是存储完全二叉树的最简单、最省存储的存储方式。但要注意，在数组中结点下标是从0开始的，计算双亲、子、兄弟结点时与性质5所述有差异。而对于一般的二叉树，最好采用链式存储结构。

4. 顺序存储下二叉树的几个操作的实现

(1) 输入二叉树的广义表表示建立二叉树的顺序存储表示。

从二叉树的广义表表示建立二叉树的规则如下：

① 广义表的表名放在表前，表示二叉树的根结点，括号中是根的左、右子树。

② 表名后面没有括号，表明它是二叉树的叶结点。