第5章

树和二叉树

本章为树和二叉树部分，首先给出本章学习目标、知识点导图，使读者对本章内容有整体了解；接着，介绍树的概念及基本术语；然后，围绕二叉树给出二叉树的定义、特性、存储结构（顺序存储和链式存储）以及二叉树的先序、中序、后序和层次遍历；之后，引入线索二叉树的概念、构造和遍历；在介绍树的存储结构的基础上，给出树和森林的转换，树、森林和对应二叉树的遍历及对应关系；最后一部分为树和二叉树的应用，具体包括二叉排序树、平衡二叉树、哈夫曼树和哈夫曼编码。

本教材在每章各个需要讲解的部分配有微课视频和配套课件，读者可根据需要扫描对应部分的二维码获取；同时，考研真题部分也适当配有真题解析微课讲解，可根据学习或复习需求扫描对应的二维码获取。

树和二叉树

5.1 本章学习目标

（1）学习树的基本概念、基本术语及其特性。

（2）掌握二叉树的概念、特性、存储结构和遍历。

（3）掌握线索二叉树的概念、构造和遍历。

（4）掌握树的存储结构、森林与二叉树的转换、树和森林的遍历。

（5）掌握树和二叉树的常见应用，如哈夫曼树和哈夫曼编码。

本部分主要以选择题形式考查，考查知识点包括树的特性，二叉树的特性，树和二叉树的遍历，树、森林和二叉树的转换，满二叉树、完全二叉树、线索二叉树、二叉排序树、平衡二叉树、哈夫曼树的定义、性质和操作，哈夫曼编码等，同时可能考查树遍历等相关的算法题。

5.2 知识点导图

树和二叉树知识点导图如图 5-1 所示。

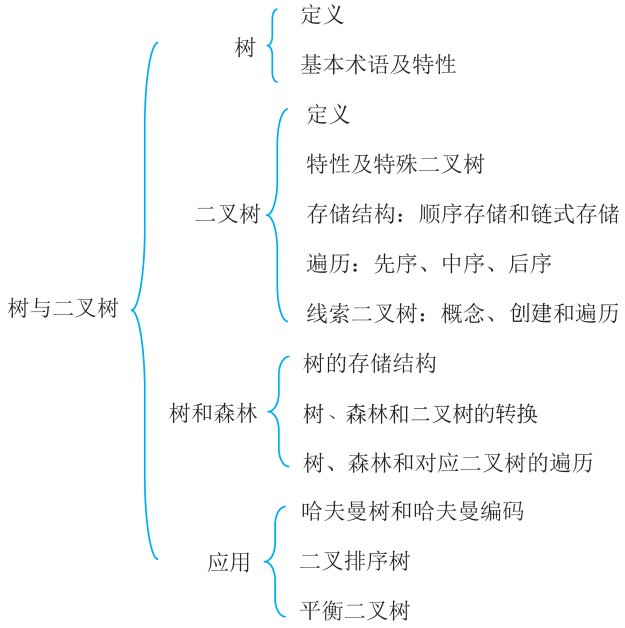


图 5-1 知识点导图

5.3 知识点归纳

5.3.1 树的基本概念和基本术语

5.3.1 树的基本概念和基本术语

1. 树的定义

树是 $n(n\geqslant 0)$个结点的有限集。当 $n=0$ 时,树为**空树**。在任意一棵非空树中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 $m(m>0)$棵互不相交的有限集 $T_1,T_2,\cdots,T_m$,每个集合本身又是一棵树,为根的**子树**(SubTree)。在一棵非空树中,数据对象是具有相同类型的数据元素的集合,数据元素之间的关系是一对多的层次关系。由于树的定义是递归的,对树的处理,原则上也可采用递归的方式。

如图 5-2 所示树的示例,左边为只有一个根结点的树;右边为具有 13 个结点的树,A 是根结点,其余结点分为 3 个不相交的子集,均为 A 的子树,分别为 T_1、T_2、T_3。其中,T_1 的根为 B,其余结点分为两个互不相交的子集 $T_{11}=\{E,K,L\}$和 $T_{12}=\{F\}$,都是 B 的子树。

T_{11}中 E 是根，{K}和{L}是 E 的两棵互不相交的子树，本身为只有一个根结点的树。可见，树的定义是递归的，树是一种递归的数据结构。

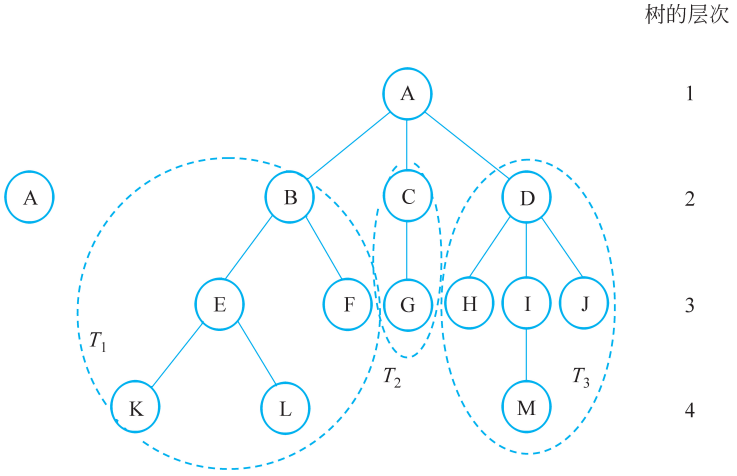


图 5-2　树的示例

2. 基本术语及树的特性

1）基本术语

（1）**结点的度、树的度**。结点拥有的子树数称为**结点的度**(Degree)；树中结点的最大度数称为**树的度**。

（2）**分支结点、叶子结点**。度不为 0 的结点称为非终端结点或**分支结点**；度为 0 的结点称为**叶子**(Leaf)或终端结点。在分支结点中，每个结点的分支数就是该结点的度。

（3）**结点的层次、深度和高度**。**结点的层次**(Level)从根结点开始定义，根结点为第一层，根结点的孩子为第二层；若某结点在第 l 层，则其子树的根就在第 $l+1$ 层；树中结点的最大层次称为树的**深度**(Depth)或**高度**。

（4）**有序树、无序树**。如果将树中结点的各子树看成从左至右是有次序的(即不能互换)，则称该树为**有序树**，否则称为**无序树**。

（5）**森林**。**森林**(Forest)是 $m(m\geqslant 0)$棵互不相交的树的集合。

（6）**路径、路径长度**。树中两个结点之间的**路径**由这两个结点之间所经过的结点序列组成；**路径长度**为路径上所经过的边的个数。

（7）**祖先、子孙、双亲、孩子、兄弟**。对于如图 5-2 所示结点 L，从根结点到结点 L 的唯一路径上的任意结点称为 L 的**祖先**；对于从根结点到结点 L 的唯一路径上，结点 L 为结点 B 的**子孙**；路径上最接近结点 L 的结点 E 为 L 的**双亲**；反过来，L 为 E 的**孩子**；有相同双亲的结点为**兄弟**，K 与 L 为兄弟。

2）树的特性

（1）树的根结点无前驱，除了根结点的其余结点均有前驱，且只有一个前驱，包括根结点在内的每个结点均有零个或者多个后继。

（2）n 个结点的树中有 $n-1$ 条边，且树中的结点数等于所有结点的度的和加 1。

（3）已知树的度为 m，则其第 i 层上最多有 m^{i-1}个结点($i\geqslant 1$)。

度为 m 的树的含义为任意结点的度小于或者等于m 并且至少有一个结点的度等于m，

假定其每个结点为最大度 m,则其第 i 层上最多有 m^{i-1} 个结点。

(4) 已知 m 叉树的高度为 h,则该树最多有 $(m^h-1)/(m-1)$ 个结点。

m 叉树的含义为任意结点的度最多只有 m 个孩子的子树,则其第 i 层上有 m^{i-1} 个结点,计算得

$$m^0+m^1+m^2+\cdots+m^h=(m^h-1)/(m-1)$$

(5) 高度为 h 的 m 叉树至少有 h 个结点。如图 5-3(a)所示,高度为 4 的三叉树至少 4 个结点;高度为 h 的度为 m 的树至少有 $h+m-1$ 个结点。

m 叉树的含义为任意结点的度最多只有 m 个孩子的子树,且允许所有结点的度都小于 m,则可得图 5-3(a)所示符合规则;高度为 h 的度为 m 的树的含义为任意结点的度小于或者等于 m 并且至少有一个结点的度等于 m,则可得图 5-3(b)所示符合规则。

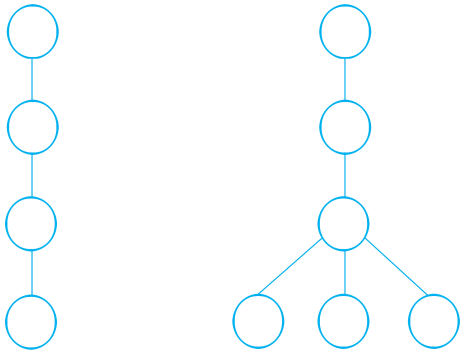


图 5-3 示意图

(6) 具有 n 个结点的 m 叉树,其最小高度为 $\lceil \log_m(n(m-1)+1) \rceil$。

其高度最小的情况为所有结点均有 m 个子树,则结点数 n 为

$$\frac{m^{h-1}-1}{m-1}<n\leqslant\frac{m^h-1}{m-1}$$

根据性质“(4) 已知 m 叉树的高度为 h,则该树最多有 $(m^h-1)/(m-1)$ 个结点”,左边为前 $h-1$ 层最多有的结点数,右边为前 h 层最多有的结点数。

求解过程为:

$$m^{h-1}<n(m-1)+1\leqslant m^h$$

$$h-1<\log_m(n(m-1)+1)\leqslant h$$

可得,h 的最小值为 $\lceil \log_m(n(m-1)+1) \rceil$。

5.3.2 二叉树的基本概念、特性及其存储结构

1. 二叉树的定义

5.3.2 二叉树的定义

二叉树(Binary Tree)是另一种树形结构,它的特点是每个结点至多只有两棵子树(即二叉树中不存在度大于 2 的结点),可以是空树($n=0$),或者由一个根结点和两个互不相交的根的左子树、右子树构成,左子树和右子树又分别是一棵二叉树。并且,二叉树的子树有左右之分,其次序不能任意颠倒。如图 5-4 所示为二叉树的 5 种基本形态。

图 5-4　二叉树基本形态示意图

图 5-4(a)为空二叉树,图 5-4(b)为仅有根结点的二叉树,图 5-4(c)为右子树为空的二叉树,图 5-4(d)为左、右子树均非空的二叉树,图 5-4(e)为左子树为空的二叉树。

满二叉树和完全二叉树是两种特殊形态的二叉树。

1) 满二叉树

一棵深度为 k 且有 2^k-1 个结点的二叉树称为**满二叉树**,如图 5-5 所示为深度为 4 的满二叉树,其特点为每层上的结点数都是最大结点数。

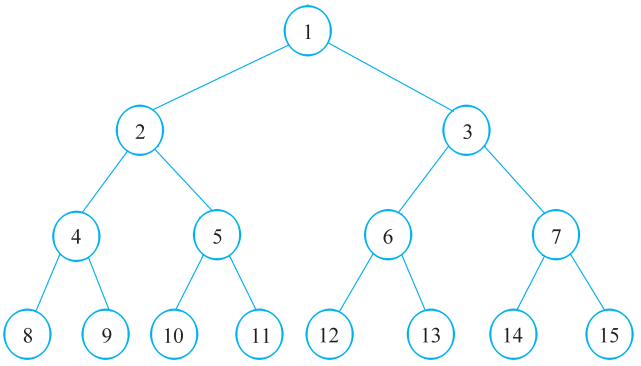


图 5-5　满二叉树

2) 完全二叉树

对满二叉树的结点进行连续编号,从根结点起,自上而下、自左而右,则深度为 k 的有 n 个结点的二叉树,当且仅当其每个结点都与深度为 k 的满二叉树中编号从 1 至 n 的结点一一对应,则称之为**完全二叉树**。如图 5-6 所示为一棵深度为 4 的完全二叉树的示意图。

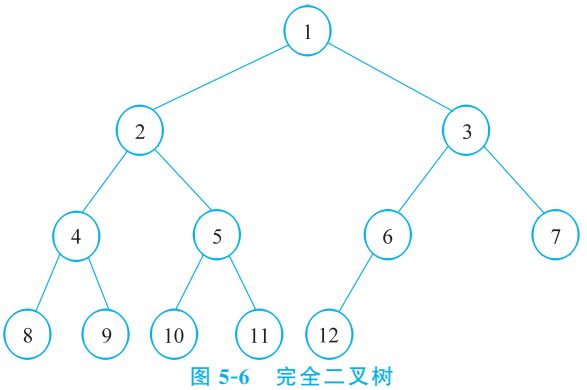


图 5-6　完全二叉树

对于结点个数为 n 的完全二叉树的特点总结如下。

(1) 完全二叉树的叶子结点只可能在层次最大的两层上出现,且最大层次中的叶子结点均依次排列在该层最左边的位置上。

(2) 对任一结点,若其右分支下的子孙的最大层次为 L,则其左分支下的子孙的最大层次必为 L 或 $L+1$。

(3) 若 $i \leqslant \lceil n/2 \rceil$,则结点 i 为分支结点。

(4) 如果有度为 1 的结点,则仅可能有一个,该结点只有左孩子没有右孩子。

(5) 如果 n 为奇数,则每个分支结点均有左、右孩子;若 n 为偶数,则编号最大的分支结点只有左孩子没有右孩子,其余的分支结点均有左孩子和右孩子。

5.3.2 二叉树的特性及其他特殊二叉树

2. 二叉树的特性及其他特殊二叉树

1) 二叉树的特性

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

证明:第一层最多有 2^{1-1} 个结点,第二层有 2^{2-1} 个结点,由此可知等比数列第 i 层上至多有 2^{i-1} 个结点。

(2) 性质 2。深度为 k 的二叉树至多有 2^k-1 个结点($k \geqslant 1$)。

证明:根据上一个性质求前 k 项的和,通过等比数列求和公式可得深度为 k 的二叉树至多有 2^k-1 个结点。

(3) 性质 3。对任何一棵二叉树 T,如果其终端结点数为 n_0,度为 2 的结点数为 n_2,则 $n_0=n_2+1$。

证明:假设度为 0、1、2 的结点数分别为 n_0、n_1、n_2,结点总数为 n,则有 $n=n_0+n_1+n_2$。

因除了根结点外其余结点均有一个分支进入,则分支总数为 $n-1$(一棵树的结点数为 n,则边数为 $n-1$);另外,从另一方面看,从度数计算分支数为 n_1+2n_2,有

$$n-1=n_1+2n_2$$

$$n_0+n_1+n_2-1=n_1+2n_2$$

可得,$n_0=n_2+1$。

(4) 性质 4。具有 n ($n>0$)个结点的完全二叉树的深度为 $\lfloor \log_2 n \rfloor+1$。

证明:假设高度为 k,根据深度为 k 的二叉树至多有 2^k-1 个结点($k \geqslant 1$)这一性质和完全二叉树的定义有 $2^{k-1} \leqslant n < 2^k$,得 $k-1 \leqslant \log_2 n < k$,因此 $k=\lfloor \log_2 n \rfloor+1$。

(5) 性质 5。如果对一棵有 n 个结点的完全二叉树,其深度为 $\lfloor \log_2 n \rfloor+1$ 的结点按层序编号,从第 1 层到第 $\lfloor \log_2 n \rfloor+1$ 层,每层从左到右,则对任一结点 i($1 \leqslant i \leqslant n$),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 PARENT(i)是结点 $\lfloor i/2 \rfloor$,也就是当 i 为偶数时,其双亲的编号为 $i/2$,i 是双亲结点左孩子;当 i 为奇数时,其双亲的编号为 $(i-1)/2$,i 是双亲结点右孩子。

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

结点 $2i$。

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 $2i+1$。

④ 结点 i 所在层次为$\lfloor \log_2 i \rfloor+1$。

2) 特殊二叉树

满二叉树和完全二叉树是两种特殊形态的二叉树，见二叉树的定义部分。此外，还有二叉排序树和平衡二叉树。

(1) 二叉排序树。**二叉排序树**(Binary Sort Tree)或者是一棵空树或者是具有如下特性的二叉树。

① 若它的左子树不为空，则左子树上所有结点的值均小于其根结点的值。

② 若它的右子树不为空，则右子树上所有结点的值均大于其根结点的值。

③ 左、右子树也都分别是二叉排序树。

(2) 平衡二叉树。平衡二叉树(Balanced Binary Tree 或 Height-Balanced Tree)又称 AVL 树。它或者是一棵空树或者是具有下列性质的二叉树：其左子树和右子树都是平衡二叉树，左子树和右子树的深度之差的绝对值不超过 1。

3. 二叉树的存储结构

5.3.2 二叉树的存储结构

二叉树有顺序存储结构和链式存储结构两种方式。

1) 顺序存储结构

二叉树的顺序存储，按照顺序存储结构的定义，用一组地址连续的存储单元自上而下、自左至右存储完全二叉树上的结点元素，也就是完全二叉树上编号为 i 的结点元素存储在上述定义的一维数组中下标为 $i-1$ 的分量中。这里，完全二叉树和满二叉树采用顺序存储，可以使树中的结点序号对应地反映出结点之间的逻辑关系，可以节省存储空间，同时能够利用数组元素的下标确定该结点在树中的关系和位置。

对于一般的二叉树，采用顺序存储结构，可对照完全二叉树的编号进行相应的存储，但在没有结点的位置补充空结点，以便能够使数组下标同样可以反映二叉树中的结点之间的逻辑关系，如图 5-7 所示为二叉树逻辑结构图及其顺序存储结构图。其中，0 表示并不存在的空结点。同时，对于二叉树的顺序存储，需要注意下标是从 0 开始还是从 1 开始。

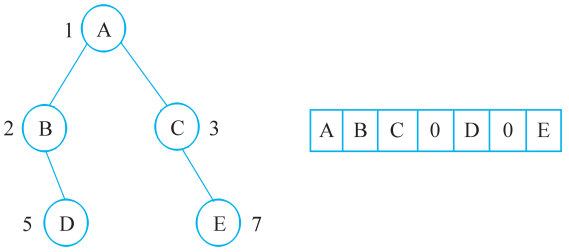


图 5-7　二叉树逻辑结构图及其顺序存储结构图

由此可见，完全二叉树和满二叉树采用顺序存储结构比较合适。

在最坏情况下,一棵深度为 k 且有 k 个结点的单支树却需要占据 2^k-1 个存储单元。

二叉树的顺序存储结构表示(二叉树的顺序存储结构的 C 语言描述)如下。

```
#define MaxTreeSize 100                    //二叉树的最大结点数
typedef char SqBiTree[MaxTreeSize];
SqBitree T;
```

二叉树的动态顺序存储结构如下。

```
typedef struct {
ElemType   * elem;                         //存储空间基址(初始化时分配空间)
int    nodenum;                            //二叉树中结点数
} SqBiTree;
SqBitree T;
```

2) 链式存储结构

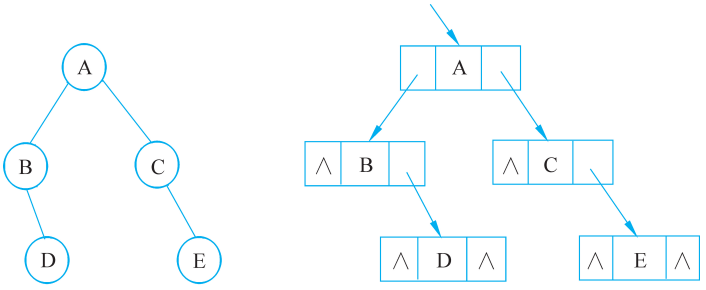
由二叉树的定义可知,二叉树的结点由一个数据元素和分别指向其左、右子树的两个分支构成,则表示二叉树的链表的结点至少包含数据域、左指针域和右指针域,即 data、lchild 和 rchild,如图 5-8 所示。

(a) 含有2个指针域的结点结构　　(b) 含有3个指针域的结点结构

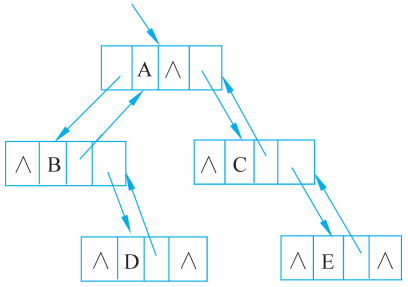
图 5-8　结点结构

顺序存储结构的存储空间利用率较低,因此,二叉树一般采取链式存储结构。

为了便于查找结点的双亲,可以在结点结构中增加一个指向其双亲结点的指针域,如图 5-9(c)所示。利用上述两种结点结构所得的二叉树的存储结构分别称为二叉链表和三叉链表,如图 5-9 所示。



(a)　　(b) 二叉链表



(c) 三叉链表

图 5-9　链式存储示意图

链表的头指针指向二叉树的根结点，并且，含有 n 个结点的二叉链表中含有 $n+1$ 个空链域，后续将通过使用这些空链域存储其他信息。另一种链式存储结构，即线索链表。

$n+1$ 个空指针的计算方法：每个叶子结点有 2 个空指针，度为 1 的结点有一个空指针，即 $2n_0+n_1$，由二叉树的性质可知 $n_0=n_2+1$，则空指针数可转化为 $n_0+n_1+n_2+1$，$n_0+n_1+n_2=n$，可得空指针数为 $n+1$。

二叉链表存储结构表示(二叉链表存储结构的 C 语言描述)如下。

```
#define MaxSize 100                          //二叉树的最大结点数
typedef char ElemType;                       //数据类型
typedef struct BiTNode
{
    ElemType data;                           //数据域
    struct BiTNode * lchild;                 //左孩子
    struct BiTNode * rchild;                 //右孩子
} BiTNode, * BiTree;                         //二叉链表结点类型定义
BiTree  T;
```

三叉链表存储结构表示(三叉链表存储结构的 C 语言描述)如下。

```
#define MaxSize 100                          //三叉树的最大结点数
typedef char ElemType;                       //数据类型
typedef struct TiTNode {
    ElemType data;
    struct TiTNode * Lchild, * Rchild;       //左、右孩子指针
    struct TiTNode * parent;                 //双亲指针
} * TriTree;
TriTree T;
```

类似于线性表的双向链表，在二叉树的三叉链表中既有指示“后继”的信息，也有指示“前驱”的信息。需要注意的是，采用不同的存储结构时，对应的二叉树的操作算法不同，换句话说，需要根据实际需求选择合适的存储结构。

5.3.3　二叉树的遍历

5.3.3 二叉树的遍历

遍历二叉树是指如何按某条搜索路径访问树中每一个结点，使得每个结点均被访问一次，而且仅被访问一次。二叉树的遍历主要包括先序遍历、中序遍历、后序遍历和层次遍历等。遍历二叉树是进行二叉树上各种操作及其应用的基础。

二叉树是一种非线性结构，每个结点都可能有两棵子树，需要寻找一种规律使得二叉树上的结点能排列在一个线性队列上以便于遍历。

二叉树由 3 个基本单元组成，即根结点 N、左子树 L 和右子树 R，依次遍历这 3 部分即可遍历整个二叉树。排列组合有 6 种遍历方案，即 NLR、LNR、LRN、NRL、RNL 和 RLN。若要求先左后右，则有 NLR、LNR、LRN 3 种遍历方案，分别对应先序(根)遍历、中序(根)

遍历和后序(根)遍历。基于二叉树的递归定义,接下来详细介绍遍历二叉树的递归算法定义。

1. 先序遍历

先序遍历的操作定义如下。

若二叉树为空树,则不做任何操作;否则

(1) 访问根结点。

(2) 先序遍历左子树。

(3) 先序遍历右子树。

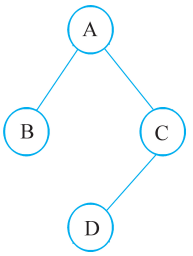
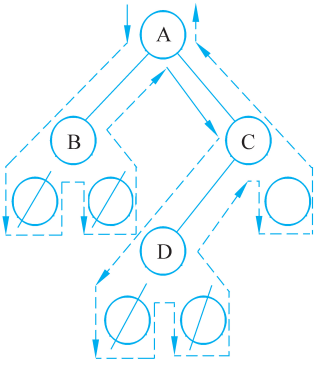
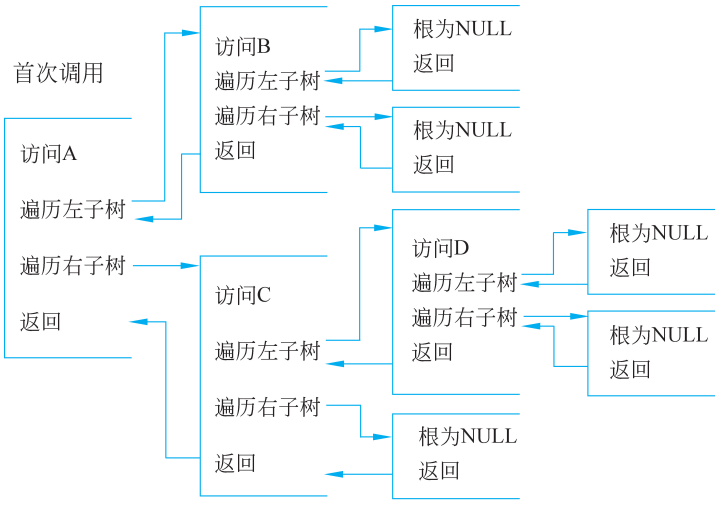


图 5-10 二叉树示意图

如图 5-10 所示,二叉树的先序遍历所得到的结点序列为 A B C D。其访问顺序如图 5-11(a)和图 5-11(b)所示。



(a) 访问顺序



(b) 详细顺序

图 5-11 先序遍历过程示意图

对应的递归算法如下。

```
void PreOrder (BiTree T)
{//先序遍历以 T 为根指针的二叉树
    if (T) {                                    //也可写作 if(T!=NULL)
```