第5章 树和二叉树

CHAPTER

5.1 内容概述

本章首先介绍树的定义和基本术语，然后介绍二叉树的定义、性质、存储结构、遍历和线索二叉树，之后介绍树的存储结构、遍历、树（森林）与二叉树的转换，最后介绍哈夫曼树及其应用。本章的知识结构如图 5.1 所示。

图 5.1 第 5 章知识结构

考核要求：掌握树的定义和基本术语，掌握二叉树的定义和性质，掌握

满二叉树和完全二叉树的特点，掌握二叉树的遍历方法及算法实现，了解二叉树的线索化过程及其算法实现，掌握树的存储、遍历以及树（森林）与二叉树的转换，掌握哈夫曼树的构造方法。

重点难点：本章的重点是二叉树的遍历算法及其相关应用，难点是如何利用本章的相关知识设计出实现某种功能的有效算法。

核心考点：二叉树的定义和性质，满二叉树和完全二叉树的特点，二叉树的存储结构，二叉树的遍历过程，二叉树的线索化过程，哈夫曼树的构造方法，二叉树上递归算法的设计。

5.2 典型题解析

5.2.1 二叉树的定义及其性质

二叉树的特点是每个结点至多有两棵子树，即任何结点的度不大于2，而且二叉树的子树有左右之分，其次序不能任意颠倒。二叉树的性质是计算或证明二叉树中某类结点的个数或高度的基础。

【例5.1】 下列关于二叉树度的说法中正确的是（　　）。

A. 二叉树的度为2　　B. 一棵二叉树的度可以小于2

C. 二叉树中至少有一个结点的度为2　　D. 二叉树中任何一个结点的度都为2

解析：二叉树中每个结点至多有两棵子树，即二叉树中任何一个结点的度都不大于2。由此可知，二叉树中可以没有度为2的结点，此时二叉树的度或者为1（结点数≥2），或者为0（结点数≤1）。

答案：B

【例5.2】 下列叙述中正确的是（　　）。

A. 二叉树是度为2的有序树

B. 完全二叉树一定存在度为1的结点

C. 对于有n个结点的二叉树，其高度为$\lfloor \log_2 n \rfloor + 1$

D. 深度为k的二叉树中的结点总数≤$2^k - 1$

解析：二叉树是度不大于2的有序树，选项A错误。当结点个数n为偶数时，完全二叉树中有且仅有一个度为1的结点；当结点个数n为奇数时，完全二叉树中没有度为1的结点，选项B错误。具有n个结点的完全二叉树的深度为$\lfloor \log_2 n \rfloor + 1$，选项C错误。深度为k（k≥1）的二叉树上至多有$2^k - 1$个结点，选项D正确。

答案：D

【例5.3】 若一棵二叉树有10个度为2的结点，5个度为1的结点，则度为0的结点的个数是（　　）。

A. 9　　B. 11　　C. 15　　D. 不确定

解析：任意一棵二叉树中，叶子结点的数目（用n_0表示）总比度为2的结点的数目（用n_2表示）多一个，即$n_0 = n_2 + 1$。

答案：B

【例 5.4】 一个具有 1025 个结点的二叉树，其高度最小为(　　)。

A. 10　　B. 11　　C. 12　　D. 不确定

解析：在结点数相同的所有形态的二叉树中，完全二叉树的深度最小，且具有 n 个结点的完全二叉树的深度为$\lfloor \log_2 n \rfloor + 1$。由此可知，具有 1025 个结点的二叉树的最小深度为$\lfloor \log_2 1025 \rfloor + 1 = 10 + 1 = 11$。

答案：B

【例 5.5】 高度为 K 的二叉树最多有(　　)个结点。

A. 2^K　　B. 2^{K-1}　　C. $2^K - 1$　　D. $2^{K-1} - 1$

解析：在所有高度为 K 的二叉树中，满二叉树的结点数最多，结点数为 $2^K - 1$。

答案：C

【例 5.6】 有 n 个结点且高度为 n 的二叉树的数目是多少？

解析：有 n 个结点且高度为 n 的二叉树是从根结点到叶子结点的单枝树，单枝树的数目由单枝树的形态所决定，单枝树的形态由叶子结点的位置所决定。深度为 n 的单枝树的叶子结点共有 2^{n-1} 种不同的位置，所以二叉树的数目是 2^{n-1}。由此可知，本题等价于高度为 n 的满二叉树有多少个叶子结点。

【例 5.7】 已知完全二叉树的第 7 层有 10 个叶子结点，则整个二叉树的结点数最多是多少？

解析：第 7 层共有 $2^{7-1} = 64$ 个结点，已知有 10 个叶子，其余 54 个结点均为分支结点。由于本题求二叉树的结点数最多是多少，所以第 8 层上有 108 个叶子结点。该二叉树的结点数最多为 $2^7 - 1 + 108 = 235$。

【例 5.8】 已知一棵满二叉树的结点数在 20～40 之间，此二叉树的叶子结点有多少个？

解析：因为深度为 k 的满二叉树的结点数为 $2^k - 1$，所以结点数在 20～40 之间的满二叉树的深度为 5。由此可知，其叶子结点数为 $2^{5-1} = 16$。

【例 5.9】 假设高度为 h 的二叉树上只有度为 0 和 2 的结点，问此类二叉树中的结点数可能达到的最大值和最小值各为多少？

解析：当二叉树是满二叉树时，结点数达到最大值，结点数为 $2^h - 1$；当二叉树除第一层外其余每层均有两个结点时，结点数达到最小值，结点数为 2h－1。

【例 5.10】 已知一棵完全二叉树顺序存储在 A[1..N]中，如何求出 A[i]和 A[j]的最近的公共祖先？

解析：根据顺序存储的完全二叉树的性质，编号为 i 的结点的双亲的编号是$\lfloor i/2 \rfloor$，故 A[i]和 A[j]的最近公共祖先可如下求出：

```
while(i/2!=j/2)
  if(i>j) i=i/2; else j=j/2;
```

退出 while 循环后，若 i/2＝＝0，则最近的公共祖先为根结点，否则最近的公共祖先是 i/2(或 j/2)。

5.2.2 二叉树的存储及其遍历

二叉树有顺序存储和链式存储（二叉链表）两种。二叉树遍历有先序遍历、中序遍历、后序遍历和层次遍历四种。考查的重点是二叉树的递归遍历和非递归遍历的思想及算法实现。考查方式主要有两种：一是对给定的二叉树，写出其先序、中序和后序遍历序列，或根据给定的遍历序列画出二叉树；二是算法设计，如建立给定遍历序列的存储结构、计算满足条件的结点数、计算二叉树的高度（深度）、交换二叉树的左右子树、将叶子结点连接成链表等。

【例 5.11】 对二叉树的结点从1开始进行连续编号，要求每个结点的编号大于其左右孩子的编号，同一结点的左右孩子中，其左孩子的编号小于其右孩子的编号，可采用（　　）遍历实现编号。

A. 先序　　B. 中序

C. 后序　　D. 从根开始按层次

解析：由于每个结点的编号大于其左右孩子的编号，所以先遍历该结点的孩子，再遍历该结点。在同一结点的左右孩子中，其左孩子的编号小于其右孩子的编号，所以先遍历左孩子，再遍历右孩子。由此可知，遍历的顺序为左孩子→右孩子→根结点，故可采用后序遍历实现编号。

答案：C

【例 5.12】 若一个二叉树的先序序列和后序序列正好相反，则该二叉树一定是（　　）的二叉树。

A. 空或只有一个结点　　B. 任一结点无左子树

C. 高度等于其结点数　　D. 任一结点无右子树

解析：若二叉树的高度等于其结点数，则每层只有一个结点，其先序序列和后序序列正好相反。选项A、B和D都是二叉树的高度等于其结点数的特殊情况，都不够全面。

答案：C

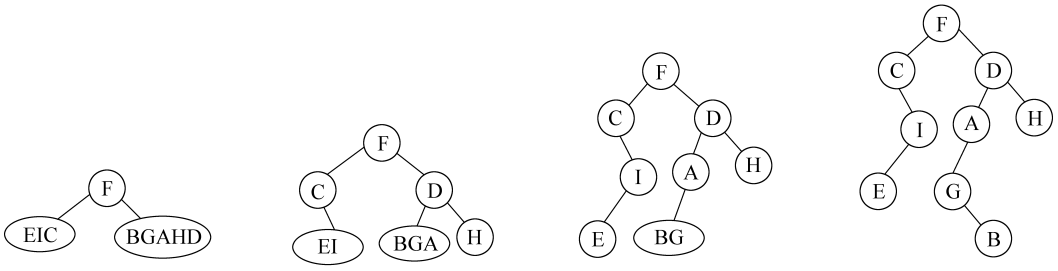
【例 5.13】 已知一棵二叉树的中序序列和后序序列，编写一个建立该二叉树的二叉链表的存储结构的算法。已知一棵二叉树的中序遍历序列为CEIFGBADH，后序遍历序列为EICBGAHDF，试画出该二叉树。

解析：首先建立根结点，根结点数据是后序遍历序列的最后一个元素，然后在中序序列中查找根结点数据，根结点之前的元素在根结点的左子树上，根结点之后的元素在根结点的右子树上，递归建立根结点的左右子树。

```
typedef char ElemType;                  /* 数据元素类型 */
typedef struct Node
{ ElemType data;                        /* 数据域 */
  struct Node *lchild,*rchild;          /* 左右指针域,分别存储左右孩子的存储位置 */
}BitTree;
BitTree *Creat(ElemType in[],ElemType post[],int l1,int h1,int l2,int h2)
/* in 和 post 存放中序序列和后序序列,l1,h1,l2,h2 分别是两个序列首尾结点的下标 */
```

```
{ BitTree *t;
  int i;
  t=(BitTree *)malloc(sizeof(BitTree));     /*申请结点*/
  t->data=post[h2];                 /*后序遍历序列的最后一个元素是根结点数据*/
  for(i=l1;i<=h1;i++)
     if(in[i]==post[h2]) break;     /*在中序序列中查找根结点*/
  if(i==l1) t->lchild=NULL;         /*处理左子树*/
  else t->lchild=Creat(in,post,l1,i-1,l2,l2+i-l1-1);
  if(i==h1) t->rchild=NULL;         /*处理右子树*/
  else t->rchild=Creat(in,post,i+1,h1,l2+i-l1,h2-1);
  return(t);
}
```

中序遍历序列为 CEIFGBADH、后序遍历序列为 EICBGAHDF 的二叉树的生成过程如图 5.2 所示。



(a) 确定第1层　(b) 确定第2层　(c) 确定第3层　(d) 确定第4层和第5层

图 5.2　二叉树的生成过程

【例 5.14】 有 n 个结点的完全二叉树存放在一维数组 A[1..n]中，试据此建立该二叉树的二叉链表。

解析：完全二叉树按层次存储在一维数组 A 中，编号从 1 到 n。在数组 A 中从下标为 1 的单元开始依次取元素值，同时建立新结点，若结点的孩子存在，则递归建立该结点的子树，否则子树为空。

```
BitTree *Creat(ElemType A[],int n,int i)
/*n是结点数,i是数组A的下标,调用时 i=1*/
{ BitTree *T;                              /*BitTree的定义见例 5.13*/
  if(i<=n)
  { T=(BitTree *)malloc(sizeof(BitTree)); /*建立新结点*/
    T->data=A[i];                         /*新结点的数据域值*/
    if(2*i>n) T->lchild=NULL;             /*左子树为空*/
    else T->lchild=Creat(A,n,2*i);        /*左子树不为空,递归建立左子树*/
    if(2*i+1>n) T->rchild=NULL;           /*右子树为空*/
    else T->rchild=Creat(A,n,2*i+1);      /*右子树不为空,递归建立右子树*/
  }
```

```
  return T;
}
```

【例5.15】 已知二叉树按二叉链表形式存储，编写算法，判断给定的二叉树是否为完全二叉树。

解析：深度为k、具有n个结点的完全二叉树的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。从根结点开始，按层次扫描二叉树，若当前结点的左子树不为空且前面已扫描结点的左右子树都不为空，或当前结点及前面已扫描的结点的左右子树都不为空，则继续扫描下一个结点；否则不是完全二叉树，结束扫描。

```
#define MAXSIZE 20                              /*最多结点数*/
int Judge(BitTree *bt)                          /*BitTree的定义见例5.13*/
{ int tag=0,front=0,rear=0;
  BitTree *p=bt, *Q[MAXSIZE];                   /*Q是队列,元素是二叉树结点指针*/
  if(p==NULL) return 1;
  Q[rear++]=p;                                  /*根结点指针入队*/
  while(front!=rear)
  { p=Q[front++];                               /*出队*/
    if(p->lchild&&! tag) Q[rear++]=p->lchild;       /*左孩子入队*/
    else if(p->lchild) return 0;                /*前边已有结点为空,本结点不为空*/
         else tag=1;                            /*首次出现结点为空*/
    if(p->rchild&&! tag) Q[rear++]=p->rchild;       /*右孩子入队*/
    else if(p->rchild) return 0;
         else tag=1;
  }
  return 1;
}
```

【例5.16】 设T是一棵满二叉树，按顺序存储方式存储，编写将T的先序遍历序列转换为后序遍历序列的递归算法。

解析：先序序列的第一个元素是后序序列的最后一个元素，剩余元素的前一半在左子树上，后一半在右子树上。用递归的方法将左子树的先序序列转换为后序序列，将右子树的先序序列转换为后序序列。对一般二叉树，仅根据一个先序、中序或后序遍历序列不能确定另一个遍历序列。但对于满二叉树，任一结点的左右子树均含有数量相等的结点，根据此性质，可将任一遍历序列转换为另一遍历序列，即任一遍历序列均可确定一棵二叉树。

```
typedef char ElemType;                          /*数据元素类型*/
void PreToPost(ElemType pre[],ElemType post[],int l1,int h1,int l2,int h2)
/*l1,h1,l2,h2分别是序列初始结点和最后结点的下标*/
{ int half;
  if(h1>=l1)
  { post[h2]=pre[l1];                           /*根结点*/
    half=(h1-l1)/2;                             /*左子树或右子树的结点数*/
```

```
    PreToPost(pre,post,l1+1,l1+half,l2,l2+half-1);
    /*将左子树先序序列转换为后序序列*/
    PreToPost(pre,post,l1+half+1,h1,l2+half,h2-1);
    /*将右子树先序序列转换为后序序列*/
  }
}
```

【例 5.17】 已知二叉树按二叉链表方式存储，设计一个算法，把二叉树的叶子结点按从左到右的顺序连接成一个单链表，表头指针为 head。连接时用叶子结点的右指针域存放单链表指针。

解析：采用中序递归遍历的方法查找叶子结点，设置前驱结点指针 pre，初始为空。第一个叶子结点由指针 head 指向，当遍历到叶子结点时，其前驱的 rchild 指针指向它，最后叶子结点的 rchild 为空。

```
BitTree *head,*pre=NULL;                          /*head为头指针,pre为尾指针*/
LinkLeaf(BitTree *bt)                             /*BitTree的定义见例5.13*/
{ if(bt)
  { LinkLeaf(bt->lchild);                         /*中序遍历左子树*/
    if(bt->lchild==NULL&&bt->rchild==NULL)        /*叶子结点*/
      if(pre==NULL) { head=bt; pre=bt;}           /*处理第一个叶子结点*/
      else { pre->rchild=bt; pre=bt; }            /*将叶子结点链入链表*/
    LinkLeaf(bt->rchild);                         /*中序遍历右子树*/
    pre->rchild=NULL;                             /*设置链表尾*/
  }
}
```

【例 5.18】 已知二叉树采用二叉链表存储，编写非递归算法，交换二叉树的左右子树。

解析：设置一个栈 stack 存放还没有交换过的结点，它的栈顶指针为 top。交换左右子树的算法如下。

(1) 把根结点放入栈。

(2) 当栈不为空时，取出栈顶元素，交换它的左右子树，并把它的左右子树的根结点分别入栈。

(3) 重复操作(2)，直到栈为空为止。

```
#define MAXSIZE 20                                /*最多结点数*/
void Exchange(BitTree *t)                         /*BitTree的定义见例5.13*/
{ BitTree *r,*p,*stack[MAXSIZE];
  int top=0;
  stack[top++]=t;                                 /*根结点入栈*/
  while(top>0)                                    /*栈不为空*/
  { p=stack[--top];                               /*出栈*/
    if(p)
    { r=p->lchild;                                /*交换*/
```

```
      p->lchild=p->rchild;
      p->rchild=r;
      stack[top++]=p->lchild;                    /*左子树根结点入栈*/
      stack[top++]=p->rchild;                    /*右子树根结点入栈*/
    }
  }
}
```

【例5.19】 假设一棵完全二叉树使用顺序存储结构存储在数组bt[1..n]中，写出进行非递归先序遍历的算法。

解析：完全二叉树按顺序存储结构存储时，双亲与孩子结点的下标之间有确定关系。对顺序存储结构的完全二叉树进行遍历与二叉链表类似。在顺序存储结构下，通过结点下标是否大于n判断完全二叉树是否为空。

```
typedef char ElemType;                           /*数据元素类型*/
#define MAXSIZE 20                               /*最多结点数*/
void PreOrder(ElemType bt[],int n)
{ int top=0,s[MAXSIZE];                          /*top是栈s的栈顶指针*/
  int i=1;
  while(i<=n||top>0)
  { while(i<=n)
    { printf("%3c",bt[i]);                       /*访问根结点*/
      if(2*i+1<=n) s[top++]=2*i+1;               /*右孩子的下标进栈*/
      i=2*i;                                     /*沿左孩子向下*/
    }
   if(top>0) i=s[--top];
  }
}
```

【例5.20】 已知二叉树T采用二叉链表存储，设计算法，返回二叉树T的先序序列的最后一个结点的指针，要求采用非递归形式，且不允许使用栈。

解析：若二叉树有右子树，则二叉树先序序列的最后一个结点是右子树中最右下的叶子结点；若二叉树无右子树，仅有左子树，则二叉树先序序列的最后一个结点是左子树最右下的叶子结点；若二叉树无左右子树，则根结点是二叉树先序序列的最后一个结点。

```
BitTree *LastNode(BitTree *bt)     /*BitTree的定义见例5.13*/
{ BitTree *p=bt;
  if(bt==NULL) return NULL;
  else
  while(p)
    if(p->rchild) p=p->rchild;     /*若右子树不为空，则沿右子树向下*/
    else if(p->lchild) p=p->lchild;/*若右子树为空，左子树不为空，则沿左子树向下*/
         else return p;            /*p即为所求*/
}
```

5.2.3　线索二叉树

线索二叉树分为先序线索二叉树、中序线索二叉树和后序线索二叉树三种。考查的重点是建立和遍历线索二叉树的算法。考查的方式主要有两种：一是画出给定二叉树的线索树，二是算法设计，如建立线索二叉树、在线索二叉树中查找给定条件的结点和在线索二叉树中插入结点等。

【例 5.21】 线索二叉树是一种(　　)结构。

A. 逻辑　　B. 逻辑和存储　　C. 物理　　D. 线性

解析：当用二叉链表作为二叉树的存储结构时，因为每个结点中只有指向左右孩子结点的指针域，所以从任一结点出发只能直接找到该结点的左右孩子，一般情况下无法直接找到该结点在某种遍历序列中的前驱和后继结点。但是，在 n 个结点的二叉链表中含有 n+1 个空指针域，因此可以利用这些空指针域存放指向结点在某种遍历次序下的前驱或后继结点的指针。这种附加的指针称为线索，加上线索的二叉链表称为线索链表，相应的二叉树称为线索二叉树。由此可知，线索二叉树是一种物理结构。

答案：C

【例 5.22】 一棵左子树为空的二叉树在先序线索化后，其空链域的个数为(　　)。

A. 不确定　　B. 0　　C. 1　　D. 2

解析：一棵左子树为空的二叉树在先序线索化后，因为第一个遍历的结点没有左孩子且没有前驱，最后一个遍历的结点没有右孩子且没有后继，所以空链域的个数为 2。

答案：D

【例 5.23】 若 X 是二叉中序线索树中的一个有左孩子的结点，且 X 不为根结点，则 X 的前驱为(　　)。

A. X 的双亲　　B. X 的右子树中最左的结点

C. X 的左子树中最右的结点　　D. X 的左子树中最右的叶子结点

解析：由中序遍历的过程可知，访问 X 的左子树的最右结点后访问 X，X 的左子树的最右结点的右链域是线索，指向其后继 X。由此可知，X 的前驱为 X 的左子树中最右的结点。

答案：C

【例 5.24】 设 t 是一棵按后序遍历方式构成的线索二叉树的根结点指针，试设计一个非递归的算法，把一个地址为 x 的新结点插入 t 树，已知地址为 y 的结点右孩子作为结点 x 的右孩子，并使插入后的二叉树仍为后序线索二叉树。

解析：在线索二叉树上插入结点，破坏了与被插入结点的线索，因此在插入结点时必须修复线索。因为是后序线索树，所以在结点 y 的右侧插入结点 x 时要区分结点 y 有无左子树的情况。

```
typedef char ElemType;                    /* 数据元素类型 */
typedef struct node
{ ElemType data;                          /* 数据域 */
  struct node *lchild;                    /* 左链域 */
```

```
  struct node * rchild;                          /* 右链域 */
  int ltag;                                      /* 左链域信息标志 */
  int rtag;                                      /* 右链域信息标志 */
}BiThrTree;
void Insert(BiThrTree * t,BiThrTree * y,BiThrTree * x)
{ BiThrTree * p;
  if(y->ltag==0)                                 /* y 有左孩子 */
  { p=y->lchild;
    if(p->rtag==1) p->rchild=x;                  /* x 是 y 的左孩子的后序后继 */
    x->ltag=1; x->lchild=p;                      /* x 的左线索是 y 的左孩子 */
  }
  else                                           /* y 无左孩子 */
  { x->ltag=1; x->lchild=y->lchild;              /* y 的左线索成为 x 的左线索 */
    if(y->lchild->rtag==1)                       /* 若 y 的后序前驱的右标记为 1 */
      y->lchild->rchild=x;                       /* 则将 y 的后序前驱的后继改为 x */
  }
  x->rtag=1;
  x->rchild=y;
  y->rtag=0;
  y->rchild=x;                                   /* x 作为 y 的右子树 */
}
```

【例 5.25】 编写算法，在中序线索二叉树中查找值为 x 的结点的后继结点，并返回该后继结点的指针。

解析：先在带头结点的中序线索二叉树 T 中查找值为 x 的结点，然后在中序线索二叉树 T 中查找值为 x 的结点的后继结点。

```
BiThrTree * Search(BiThrTree * T,ElemType x)   /* BiThrTree 的定义见例 5.24 */
/* 在带头结点的中序线索二叉树 T 中查找值为 x 的结点 */
{ BiThrTree * p;
  p=T->lchild;                                   /* p 指向二叉树的根结点 */
  while(p!=T)
  { while(p->ltag==0&&p->data!=x) p=p->lchild;
    if(p->data==x) return p;
    while(p->rtag==1&&p->rchild!=T)
    { p=p->rchild; if(p->data==x) return p;}
    p=p->rchild;
  }
}
BiThrTree * AfterNode(BiThrTree * T,ElemType x)
/* 在中序线索二叉树 T 中查找值为 x 的结点的后继结点 */
{ BiThrTree * p, * q;
  p=Search(T,x);          /* 在 T 中查找给定值为 x 的结点，由 p 指向 */
  if(p->rtag==1)
```