第 7 章 树和二叉树

教材中练习题及参考答案

1. 有一棵树的括号表示为 $A(B，C(E，F(G))，D)$，回答下面的问题：
（1）指出树的根结点。
（2）指出棵树的所有叶子结点。
（3）结点 C 的度是多少？
（4）这棵树的度为多少？
（5）这棵树的高度是多少？
（6）结点 C 的孩子结点是哪些？
（7）结点 C 的双亲结点是谁？
答：该树对应的树形表示如图 7.2 所示。
（1）这棵树的根结点是 A。
（2）这棵树的叶子结点是 B、E、G、D。
（3）结点 C 的度是 2。
（4）这棵树的度为 3。
（5）这棵树的高度是 4。
（6）结点 C 的孩子结点是 E、F。
（7）结点 C 的双亲结点是 A。

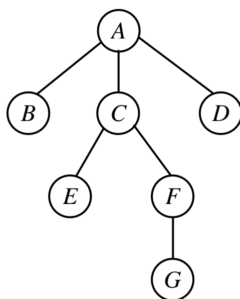

图 7.2 一棵树

2. 若一棵度为 4 的树中度为 2、3、4 的结点个数分别为 3、2、2，则该树的叶子结点的个数是多少？

答：结点总数 $n=n_0+n_1+n_2+n_3+n_4$，又由于除根结点外，每个结点都对应一个分支，所以总的分支数等于 $n-1$。而一个度为 i（$0 \leqslant i \leqslant 4$）的结点的分支数为 i，所以有：总分支数

$=n-1=1\times n_1+2\times n_2+3\times n_3+4\times n_4$。综合两式得：$n_0=n_2+2n_3+3n_4+1=3+2\times2+3\times2=14$。

3. 为了实现以下各种功能，其中 *x* 结点表示该结点的位置，给出树的最适合的存储结构：

（1）求 *x* 和 *y* 结点的最近祖先结点。

（2）求 *x* 结点的所有子孙。

（3）求根结点到 *x* 结点的路径。

（4）求 *x* 结点的所有右边兄弟结点。

（5）判断 *x* 结点是否是叶子结点。

（6）求 *x* 结点的所有孩子。

答：（1）双亲存储结构。

（2）孩子链存储结构。

（3）双亲存储结构。

（4）孩子兄弟链存储结构。

（5）孩子链存储结构。

（6）孩子链存储结构。

4. 设二叉树 *bt* 的一种存储结构如表 7.1 所示。其中，*bt* 为树根结点指针，*lchild*、*rchild* 分别为结点的左、右孩子指针域，在这里使用结点编号作为指针域值，0 表示指针域值为空；*data* 为结点的数据域。请完成下列各题：

（1）画出二叉树 *bt* 的树形表示。

（2）写出按先序、中序和后序遍历二叉树 *bt* 所得到的结点序列。

（3）画出二叉树 *bt* 的后序线索树（不带头结点）。

表7.1 二叉树bt的一种存储结构

	1	2	3	4	5	6	7	8	9	10
lchild	0	0	2	3	7	5	8	0	10	1
data	*j*	*h*	*f*	*d*	*b*	*a*	*c*	*e*	*g*	*i*
rchild	0	0	0	9	4	0	0	0	0	0

答：（1）二叉树*bt*的树形表示如图7.3所示。

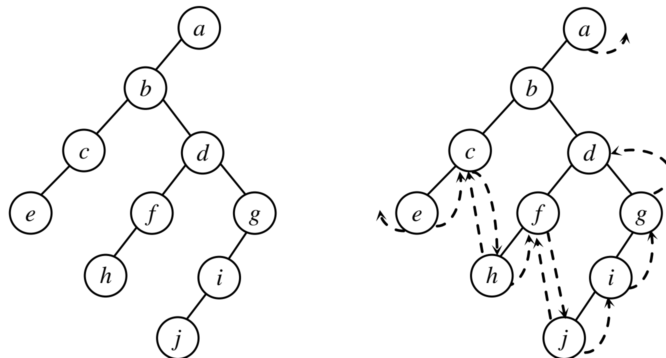

图 7.3 二叉树 *bt* 的逻辑结构　　　图 7.4 二叉树 *bt* 的后序线索化树

（2）先序序列：*abcedfhgij*

　　中序序列：*ecbhfdjiga*

　　后序序列：*echfjigdba*

（3）二叉树 *bt* 的后序序列为 *echfjigdba*，则后序线索树如图 7.4 所示。

5. 含有 60 个叶子结点的二叉树的最小高度是多少？

答：在该二叉树中，$n_0=60$，$n_2=n_0-1=59$，$n=n_0+n_1+n_2=119+n_1$，当 $n_1=0$ 且为完全二叉树时高度最小，此时高度 $h=\lceil\log_2(n+1)\rceil=\lceil\log_2 120\rceil=7$。所以含有 60 个叶子结点的二叉树的最小高度是 7。

6. 已知一棵完全二叉树的第 6 层（设根结点为第 1 层）有 8 个叶子结点，则该完全二叉树的结点个数最多是多少？最少是多少？

答：完全二叉树的叶子结点只能在最下面两层，所以结点最多的情况是第 6 层为倒数第 2 层，即 1～6 层构成一棵满二叉树，其结点总数为 $2^6-1=63$。其中第 6 层有 $2^5=32$ 个结点，含 8 个叶子结点，则另外有 32−8=24 个非叶子结点，它们中每个结点有两个孩子结点（均为第 7 层的叶子结点），计为 48 个叶子结点。这样最多的结点个数=63+48=111。

结点最少的情况是第 6 层为最下层，即 1～5 层构成一棵满二叉树，其结点总数为 $2^5-1=31$，再加上第 6 层的结点，总计 31+8=39。这样最少的结点个数为 39。

7. 已知一棵满二叉树的结点个数为 20～40 之间，此二叉树的叶子结点有多少个？

答：一棵高度为 h 的满二叉树的结点个数为 2^h-1，有：$20\leqslant 2^h-1\leqslant 40$。

则 $h=5$，满二叉树中叶子结点均集中在最底层，所以叶子结点个数$=2^{5-1}=16$ 个。

8. 已知一棵二叉树的中序序列为 *cbedahgijf*，后序序列为 *cedbhjigfa*，给出该二叉树树形表示。

答：该二叉树的构造过程和二叉树如图 7.5 所示。

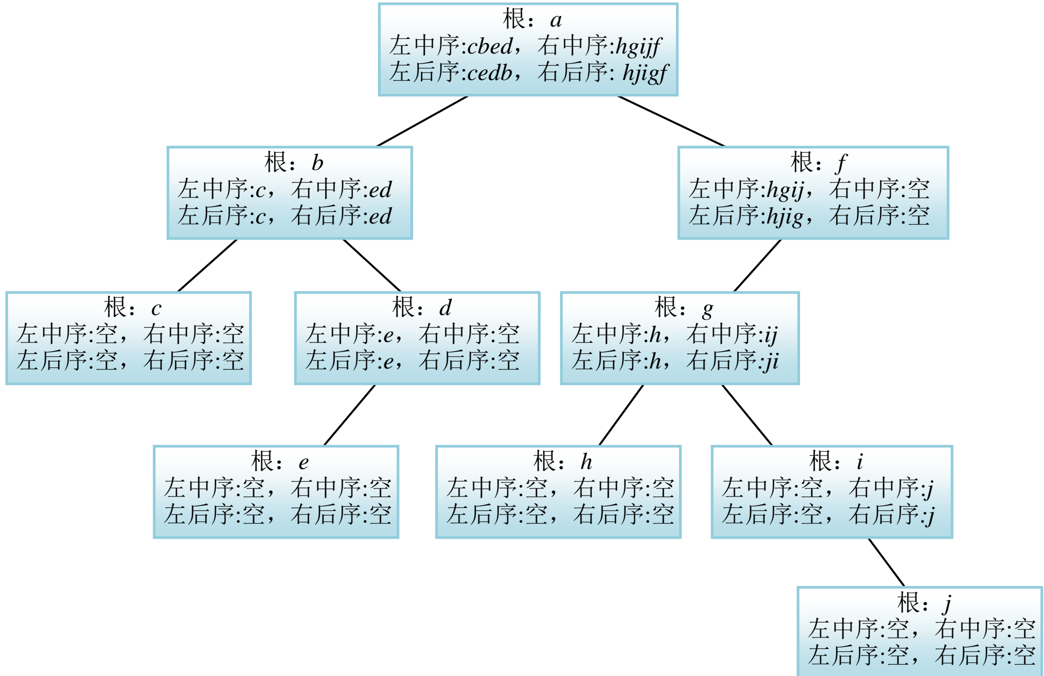

图 7.5 二叉树的构造过程

9. 给定 5 个字符 *a*～*f*，它们的权值集合 *W*={2，3，4，7，8，9}，试构造关于 *W* 的一棵哈夫曼树，求其带权路径长度 *WPL* 和各个字符的哈夫曼树编码。

答： 由权值集合 *W* 构建的哈夫曼树如图 7.6 所示。其带权路径长度 $WPL=(9+7+8)\times2+4\times3+(2+3)\times4=80$。

各个字符的哈夫曼树编码：*a*：0000，*b*：0001，*c*：001，*d*：10，*e*：11，*f*：01。

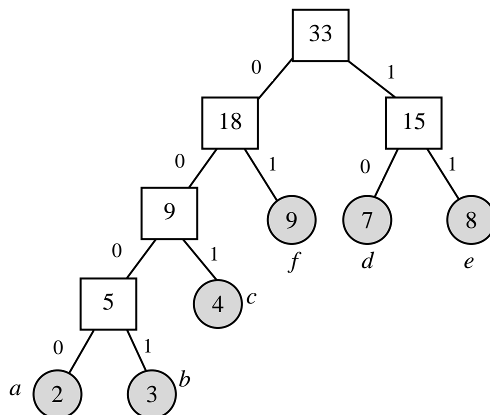

图7.6 一棵哈夫曼树

10. 假设二叉树中每个结点的值为单个字符，设计一个算法将一棵以二叉链方式存储的二叉树 *b* 转换成对应的顺序存储结构 *a*。

解：设二叉树的顺序存储结构类型为 SqBTree，先将顺序存储结构 *a* 中所有元素置为'#'（表示空结点）。将 *b* 转换成 *a* 的递归模型如下：

f(*b*，*a*，*i*) ≡ *a*[*i*]='#';　　　　　　　　　当 *b*=NULL

f(*b*，*a*，*i*) ≡ 由 *b* 结点 data 域值建立 *a*[*i*]元素;　　其他情况
　　f(b->lchild，*a*，2**i*);
　　f(b->rchild，*a*，2**i*+1)

调用方式为：*f*(*b*，*a*，1)（*a* 的下标从 1 开始）。对应的算法如下：

```
void Ctree(BTNode *b,SqBTree a,int i)
{   if (b!=NULL)
    {   a[i]=b->data;
        Ctree(b->lchild,a,2*i);
        Ctree(b->rchild,a,2*i+1);
    }
    else a[i]='#';
}
```

11. 假设二叉树中每个结点值为单个字符，采用顺序存储结构存储。设计一个算法，求二叉树 *t* 中的叶子结点个数。

解：用 *i* 遍历所有的结点，当 *i* 大于等于 MaxSize 时，返回 0。当 *t*[*i*]是空结点时返回 0；当 *t*[*i*]是非空结点时，若它为叶子结点，num 增 1；否则递归调用 num1=LeftNode(*t*，2**i*)求出左子树的叶子结点个数 num1，再递归调用 num2=LeftNode(*t*，2**i*+1)求出右子树的叶子结点个数 num2，置 num+=num1+num2。最后返回 num。对应的算法如下：

```
int LeftNode(SqBTree t,int i)
{   //i 的初值为 1
    int num1,num2,num=0;
    if (i<MaxSize)
    {   if (t[i]!='#')
        {   if (t[2*i]=='#' && t[2*i+1]=='#')
                num++;              //叶子结点个数增 1
            else
            {   num1=LeftNode(t,2*i);
                num2=LeftNode(t,2*i+1);
                num+=num1+num2;
            }
            return num;
        }
        else return 0;
    }
    else return 0;
}
```

12. 假设二叉树中每个结点值为单个字符，采用二叉链存储结构存储。设计一个算法计算一棵给定二叉树 *b* 中的所有单分支结点个数。

解：计算一棵二叉树的所有单分支结点个数的递归模型 *f*(*b*)如下：

f(*b*)=0　　　　　　　　　　　　　　若 *b*=NULL

$f(b)=f(b\text{->}lchild)+f(b\text{->}rchild)+1$　　若 b 结点为单分支

$f(b)=f(b\text{->}lchild)+f(b\text{->}rchild)$　　其他情况

对应的算法如下：

```
int SSonNodes(BTNode *b)
{   int num1,num2,n;
    if (b==NULL)
        return 0;
    else if ((b->lchild==NULL && b->rchild!=NULL) ||
        (b->lchild!=NULL && b->rchild==NULL))
        n=1;                    //为单分支结点
    else
        n=0;                    //其他结点
    num1=SSonNodes(b->lchild);  //递归求左子树中单分支结点数
    num2=SSonNodes(b->rchild);  //递归求右子树中单分支结点数
    return (num1+num2+n);
}
```

上述算法采用的是先序遍历的思路。

13. 假设二叉树中每个结点值为单个字符，采用二叉链存储结构存储。设计一个算法求二叉树 b 中最小值的结点值。

解：设 $f(b, min)$是在二叉树 b 中寻找最小结点值 min，其递归模型如下：

$f(b, min) \equiv$ 不做任何事件　　若 b=NULL

$f(b, min) \equiv$ 当 $b\text{->}data<min$ 时置 $min=b\text{->}data$;　　其他情况

$f(b\text{->}lchild, min); f(b\text{->}rchild, min);$

对应的算法如下：

```
void FindMinNode(BTNode *b,char &min)
{   if (b->data<min)
        min=b->data;
    FindMinNode(b->lchild,min);     //在左子树中找最小结点值
    FindMinNode(b->rchild,min);     //在右子树中找最小结点值
}
void MinNode(BTNode *b)             //输出最小结点值
{   if (b!=NULL)
    {   char min=b->data;
        FindMinNode(b,min);
        printf("Min=%c\n",min);
    }
}
```

14. 假设二叉树中每个结点值为单个字符，采用二叉链存储结构存储。设计一个算法将二叉链 $b1$ 复制到二叉链 $b2$ 中。

解：当 $b1$ 为空时，置 $b2$ 为空树。当 $b1$ 不为空时，建立 $b2$ 结点（$b2$ 为根结点），置 $b2\text{->}data=b1\text{->}data$；递归调用 Copy($b1\text{->}lchild$，$b2\text{->}lchild$)，由 $b1$ 的左子树建立 $b2$ 的左子树；递归调用 Copy($b1\text{->}rchild$，$b2\text{->}rchild$)，由 $b1$ 的右子树建立 $b2$ 的右子树。对应的算法如下：

```
void Copy(BTNode *b1,BTNode *&b2)
{   if (b1==NULL)
        b2=NULL;
    else
    {   b2=(BTNode *)malloc(sizeof(BTNode));
        b2->data=b1->data;
        Copy(b1->lchild,b2->lchild);
        Copy(b1->rchild,b2->rchild);
    }
}
```

15. 假设二叉树中每个结点值为单个字符，采用二叉链存储结构存储。设计一个算法，求二叉树 *b* 中第 *k* 层上叶子结点个数。

解：采用先序遍历方法，当 *b* 为空时返回 0。置 *num* 为 0。若 *b* 不为空，当前结点的层次为 *k*，并且 *b* 为叶子结点，则 *num* 增 1，递归调用 *num*1=LevelkCount(*b*->*lchild*，*k*，*h*+1)求出左子树中第 *k* 层的结点个数 *num*1，递归调用 *num*2=LevelkCount(*b*->*rchild*，*k*，*h*+1)求出右子树中第 *k* 层的结点个数 *num*2，置 *num*+=*num*1+*num*2，最后返回 *num*。对应的算法如下：

```
int LevelkCount(BTNode *b,int k,int h)
{   //h 的初值为 1
    int num1,num2,num=0;
    if (b!=NULL)
    {   if (h==k && b->lchild==NULL && b->rchild==NULL)
            num++;
        num1=LevelkCount(b->lchild,k,h+1);
        num2=LevelkCount(b->rchild,k,h+1);
        num+=num1+num2;
        return num;
    }
    return 0;
}
int Levelkleft(BTNode *b,int k   //返回二叉树 b 中第 k 层上叶子结点个数
{
    return LevelkCount(b,k,1);
}
```

16. 假设二叉树中每个结点值为单个字符，采用二叉链存储结构存储。设计一个算法，判断值为 *x* 的结点与值为 *y* 的结点是否互为兄弟，假设这样的结点值是唯一的。

解：采用先序遍历方法，当 *b* 为空时直接返回 false；否则，若当前结点 *b* 是双分支结点，且有两个互为兄弟的结点 *x*、*y*，则返回 true；否则，递归调用 *flag*=Brother(*b*->*lchild*，*x*，*y*)，求出 *x*、*y* 在左子树中是否互为兄弟，若 *flag* 为 true，则返回 true；否则递归调用 Brother(*b*->*rchild*，*x*，*y*)，求出 *x*、*y* 在右子树中是否互为兄弟，并返回其结果。对应的算法如下：

```
bool Brother(BTNode *b,char x,char y)
{   bool flag;
    if (b==NULL)
        return false;
```

```
        else
        {   if (b->lchild!=NULL && b->rchild!=NULL)
            {   if ((b->lchild->data==x && b->rchild->data==y) ||
                    (b->lchild->data==y && b->rchild->data==x))
                    return true;
            }
            flag=Brother(b->lchild,x,y);
            if (flag==true)
                return true;
            else
                return Brother(b->rchild,x,y);
        }
    }
```

17. 假设二叉树中每个结点值为单个字符，采用二叉链存储结构存储。设计一个算法，采用先序遍历方法求二叉树 b 中值为 x 的结点的子孙，假设值为 x 的结点是唯一的。

解：设计 Output(p)算法输出以 p 为根结点的所有结点。首先在二叉树 b 中查找值为 x 的结点，当前 b 结点是这样的结点，调用 Output(*b*->*lchild*)输出其左子树中所有结点，调用 Output(*b*->*rchild*)输出其右子树中所有结点，并返回；否则，递归调用 Child(*b*->*lchild*，x)在左子树中查找值为 x 的结点，递归调用 Child(*b*->*rchild*，x)在右子树中查找值为 x 的结点。对应的算法如下：

```
void Output(BTNode *p)          //输出以p为根结点的子树
{   if (p!=NULL)
    {   printf("%c ",p->data);
        Output(p->lchild);
        Output(p->rchild);
    }
}
void Child(BTNode *b,char x)    //输出x结点的子孙
{   if (b!=NULL)
    {   if (b->data==x)
        {   if (b->lchild!=NULL)
                Output(b->lchild);
            if (b->rchild!=NULL)
                Output(b->rchild);
            return;
        }
        Child(b->lchild,x);
        Child(b->rchild,x);
    }
}
```

18. 假设二叉树采用二叉链存储结构，设计一个算法把二叉树 b 的左、右子树进行交换。要求不破坏原二叉树。并用相关数据进行测试。

解：交换二叉树的左、右子树的递归模型如下：

$f(b，t) \equiv t$=NULL　　　　若 b=NULL

$f(b，t) \equiv$ 复制根结点 b 产生结点 t;　　　　其他情况

$f(b\text{->}lchild，t1)$; $f(b\text{->}rchild，t2)$;

$t\text{->}lchild=t2$; $t\text{->}rchild=t1$

对应的算法如下（算法返回左、右子树交换后的二叉树）：

```
#include "btree.cpp"              //二叉树基本运算算法
BTNode *Swap(BTNode *b)
{    BTNode *t,*t1,*t2;
     if (b==NULL)
          t=NULL;
     else
     {    t=(BTNode *)malloc(sizeof(BTNode));
          t->data=b->data;        //复制产生根结点 t
          t1=Swap(b->lchild);
          t2=Swap(b->rchild);
          t->lchild=t2;
          t->rchild=t1;
     }
     return t;
}
```

或者设计成如下算法（算法产生左、右子树交换后的二叉树 $b1$）：

```
void Swap1(BTNode *b,BTNode *&b1)
{    if (b==NULL)
          b1=NULL;
     else
     {    b1=(BTNode *)malloc(sizeof(BTNode));
          b1->data=b->data;                //复制产生根结点 b1
          Swap1(b->lchild,b1->rchild);
          Swap1(b->rchild,b1->lchild);
     }
}
```

设计如下主函数：

```
int main()
{    BTNode *b,*b1;
     CreateBTree(b,"A(B(D(,G)),C(E,F))");
     printf("交换前的二叉树:");DispBTree(b);printf("\n");
     b1=Swap(b);
     printf("交换后的二叉树:");DispBTree(b1);printf("\n");
     DestroyBTree(b);
     DestroyBTree(b1);
     return 1;
}
```

程序执行结果如下：

```
交换前的二叉树:A(B(D(,G)),C(E,F))
交换后的二叉树:A(C(F,E),B(,D(G)))
```

19. 假设二叉树采用二叉链存储结构，设计一个算法判断一棵二叉树 b 的左、右子树是否同构。

解：判断二叉树 $b1$、$b2$ 是否同构的递归模型如下：

$f(b1，b2)$=true　　　　$b1=b2$=NULL

$f(b1，b2)$=false　　　　若 $b1$、$b2$ 中有一个为空，另一个不为空

$f(b1，b2)=f(b1\text{->}lchild，b2\text{->}lchild)$ & $f(b1\text{->}rchild，b2\text{->}rchild)$　　　　其他情况

对应的算法如下：

```
bool Symm(BTNode *b1,BTNode *b2) //判断二叉树 b1 和 b2 是否同构
{   if (b1==NULL && b2==NULL)
        return true;
    else if (b1==NULL || b2==NULL)
        return false;
    else
        return (Symm(b1->lchild,b2->lchild) &Symm(b1->rchild,b2->rchild));
}
bool Symmtree(BTNode *b)          //判断二叉树的左、右子树是否同构
{   if (b==NULL)
        return true;
    else
        return Symm(b->lchild,b->rchild);
}
```

20. 假设二叉树以二叉链存储，设计一个算法，判断一棵二叉树 b 是否为完全二叉树。

解：根据完全二叉树的定义，对完全二叉树按照从上到下、从左到右的次序遍历（层次遍历）应该满足：

（1）某结点没有左孩子，则一定无右孩子。

（2）若某结点缺左或右孩子（一旦出现这种情况，置 bj=false），则其所有后继一定无孩子。

若不满足上述任何一条，均不为完全二叉树（cm=true 表示是完全二叉树，cm=false 表示不是完全二叉树）。对应的算法如下：

```
bool CompBTree(BTNode *b)
{   BTNode *Qu[MaxSize],*p;            //定义一个队列，用于层次遍历
    int front=0,rear=0;                //环形队列的队头队尾指针
    bool cm=true;                      //cm 为真表示二叉树为完全二叉树
    bool bj=true;                      //bj 为真表示到目前为止所有结点均有左右孩子
    if (b==NULL) return true;          //空树当成特殊的完全二叉树
    rear++;
    Qu[rear]=b;                        //根结点进队
    while (front!=rear)                //队列不空
    {   front=(front+1)%MaxSize;
        p=Qu[front];                   //出队结点 p
        if (p->lchild==NULL)           //p 结点没有左孩子
        {   bj=false;                  //出现结点 p 缺左孩子的情况
            if (p->rchild!=NULL)       //没有左孩子但有右孩子,违反(1),
                cm=false;
        }
```

```
        else                            //p 结点有左孩子
        {   if (!bj) cm=false;          //bj 为假而结点 p 还有左孩子，违反(2)
            rear=(rear+1)%MaxSize;
            Qu[rear]=p->lchild;         //左孩子进队
            if (p->rchild==NULL)
                bj=false;               //出现结点 p 缺右孩子的情况
            else                        //p 有左右孩子,则继续判断
            {   rear=(rear+1)%MaxSize;
                Qu[rear]=p->rchild;     //将 p 结点的右孩子进队
            }
        }
    }
    return cm;
}
```