第 4 章 树结构

1. 选择题

（1）C （2）C （3）B （4）B （5）B （6）C （7）C （8）D （9）A （10）D
（11）D （12）B （13）B （14）D （15）B

2. 判断题

（1）√（2）√ （3）X （4）X（5）√ （6）X（7）X （8）X（9）√（10）X
（11）X（12）X（13）√（14）X（15）X（16）X（17）√（18）X（19）X（20）√

3. 简答题

（1）一棵度为 2 的树与一棵二叉树有何区别？树与二叉树之间有何区别？

【解答】

①二叉树是有序树，度为 2 的树是无序树，二叉树的度不一定是 2。

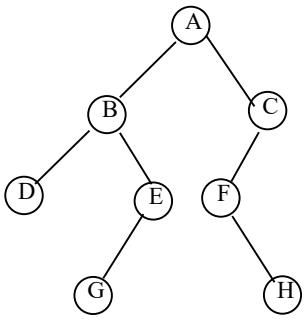
②二叉树是有序树，每个结点最多有两棵子树，树是无序树，且每个结点可以有多棵子树。

（2）对于图 4-37 所示二叉树，试给出：

1）它的顺序存储结构示意图；

2）它的二叉链表存储结构示意图；

3）它的三叉链表存储结构示意图。



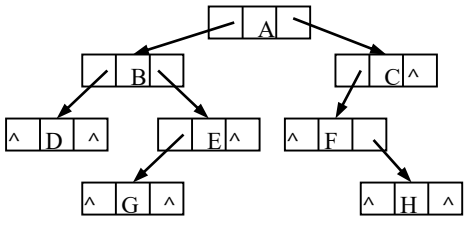
(图 4-37)

【解答】

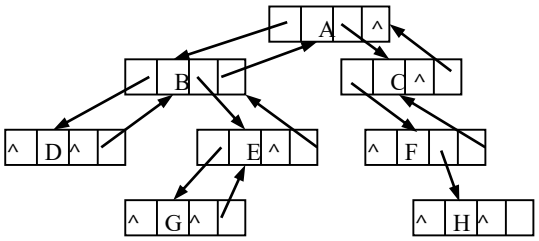
1）顺序存储结构示意图：

A	B	C	D	E	F	^	^	^	G	^	^	H

2）二叉链表存储结构示意图：



3）三叉链表存储结构示意图：

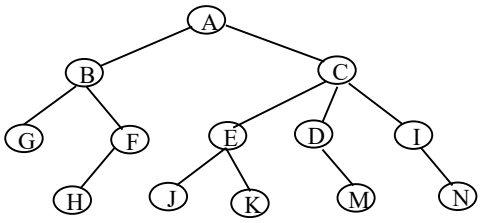


（3）对于图 4-38 所示的树，试给出：

1）双亲数组表示法示意图；

2）孩子链表表示法示意图；

3）孩子兄弟链表表示法示意图。

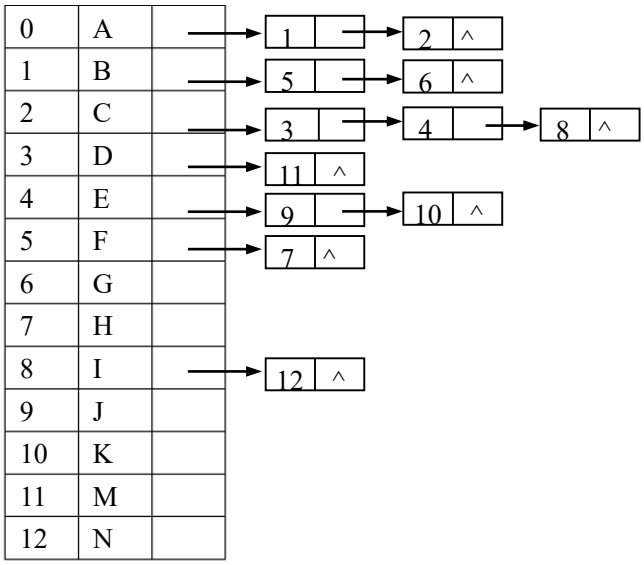


(图 4-38)

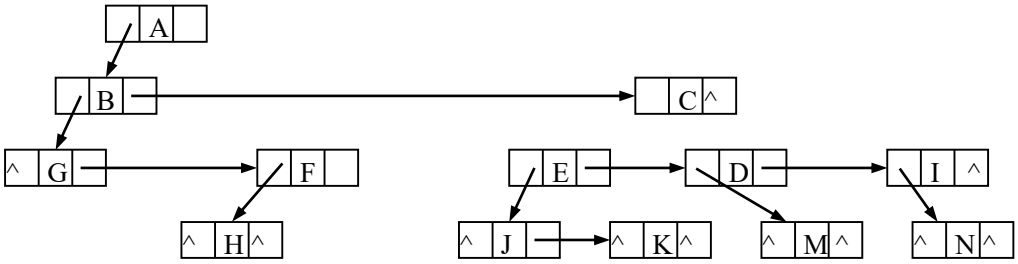
【解答】

1）双亲数组表示法示意图：　　2）孩子链表表示法示意图：

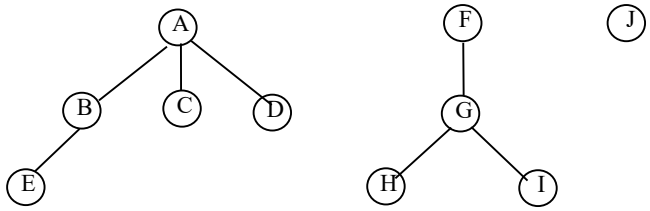
0	A	-1
1	B	0
2	C	0
3	D	2
4	E	2
5	F	1
6	G	1
7	H	5
8	I	2
9	J	4
10	K	4
11	M	3
12	N	8



3）孩子兄弟链表表示法示意图：

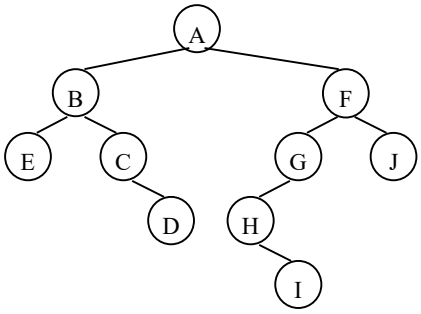


（4）画出图 4-39 所示的森林经转换后所对应的二叉树，并指出森林中满足什么条件的结点在二叉树中是叶子。



(图 4-39)

【解答】



在二叉树中某结点所对应的森林中结点为叶子结点的条件是该结点在森林中既没有孩子也没有右兄弟结点。

（5）将题 4-40 图所示的二叉树转换成相应的森林。

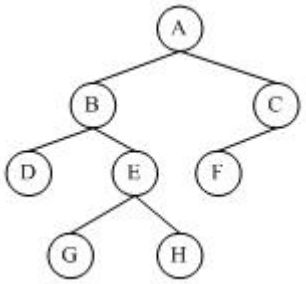
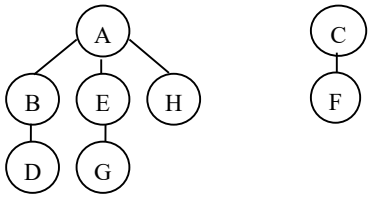


图 4-40 简答题 5 用图

【解答】森林：



（6）证明：在结点数多于 1 的哈夫曼树中不存在度为 1 的结点。

证明：由哈夫曼树的构造过程可知，哈夫曼树的每一分支结点都是由两棵子树合并产生的新结点，其度必为 2，所以哈夫曼树中不存在度为 1 的结点。

（7）证明：若哈夫曼树中有 n 个叶结点，则树中共有 2n－1 个结点。

证明：n 个叶结点，需经 n-1 次合并形成哈夫曼树，而每次合并产生一个分支结点，所以树中共有 2n-1 个结点。

（8）证明：由二叉树的前序序列和中序序列可以唯一地确定一棵二叉树。

证明：给定二叉树结点的前序序列和对称序（中序）序列，可以唯一确定该二叉树。因为前序序列的第一个元素是根结点，该元素将二叉树中序序列分成两部分，左边（设 l 个元素）表示左子树，若左边无元素，则说明左子树为空；右边（设 r 个元素）是右子树，若为空，则右子树为空。根据前序遍历中“根—左子树—右子树”的顺序，则由从第二元素开始的 l 个结点序列和中序序列根左边的 l 个结点序列构造左子树，由前序序列最后 r 个元素序列与中序序列根右边的 r 个元素序列构造右子树。

（9）已知一棵度为 m 的树中有 n_1 个度为 1 的结点，n_2 个度为 2 的结点，……，n_m 个度为 m 的结点，问该树中共有多少个叶子结点？有多少个非终端结点？

解：设树中共有 n 个结点，n_0 个叶结点，那么

$$n=n_0+n_1+\ldots+n_m \qquad (1)$$

树中除根结点外，每个结点对应着一个分支，而度为 k 的结点发出 k 个分支，所以：

$$n=n_1+2*n_2+\ldots+m*n_m+1 \qquad (2)$$

由(1)、(2)可知 $n_0= n_2+2*n_3+3*n_4+\ldots+(m-1)*n_m+1$

（10）在具有 n（n>1）个结点的树中，深度最小的那棵树其深度是多少？它共有多少叶子和非叶子结点？深度最大的那棵树其深度是多少？它共有多少叶子和非叶子结点？

2; n-1; 1; n; 1, n-1

（11）设高度为 h 的二叉树上只有度为 0 和度为 2 的结点，问该二叉树的结点数可能达到的最大值和最小值。

最大值：2^h-1； 最小值：2h-1

（12）求表达式： a＋b*(c－d)－e/f 的波兰式（前缀式）和逆波兰式（后缀式）。

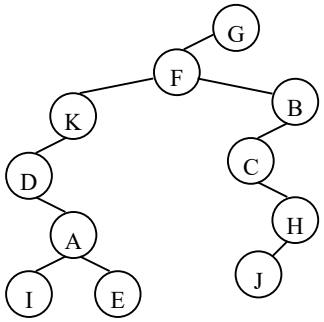
波兰式： - + a * b – c d / e f　　逆波兰式：a b c d - * + e f / -

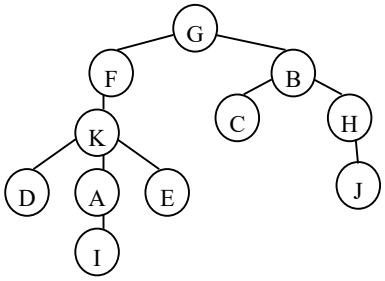
（13）画出和下列已知序列对应的树 T：

树的先根次序访问序列为：GFKDAIEBCHJ；

树的后根访问次序为：DIAEKFCJHBG。

【解答】对应的二叉树和树分别如下左、右图所示：

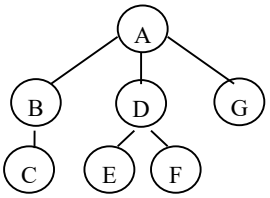


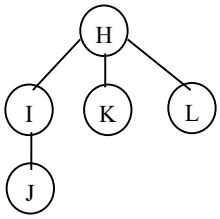


（14）画出和下列已知序列对应的森林 F：

森林的先根次序访问序列为：ABCDEFGHIJKL；

森林的后根访问次序为：CBEFDGAJIKLH。



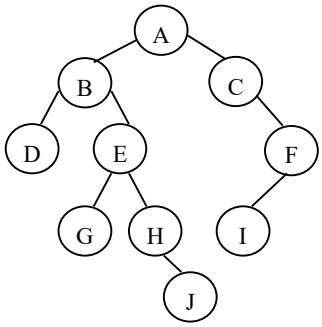


（15）画出和下列已知序列对应的树 T：

二叉树的层次访问序列为：ABCDEFGHIJ；

二叉树的中序访问次序为：DBGEHJACIF。

【解答】



按层次遍历，第一个结点（若树不空）为根，该结点在中序序列中把序列分成左右两部分—左子树和右子树。若左子树不空，层次序列中第二个结点左子树的根；若左子树为空，则层次序列中第二个结点右子树的根。对右子树也作类似的分析。层次序列的特点是：从左到右每个结点或是当前情况下子树的根或是叶子。

16．假设用于通信的电文由字符集{a,b,c,d,e,f,g}中的字母构成。它们在电文中出现的频度分别为{0.31,0.16,0.10,0.08,0.11,0.20,0.04}，

（1）为这 7 个字母设计哈夫曼编码。

（2）对这 7 个字母进行等长编码，至少需要几位二进制数？哈夫曼编码比等长编码使电文总长压缩多少？

（1）哈夫曼树：

a：10

b：110

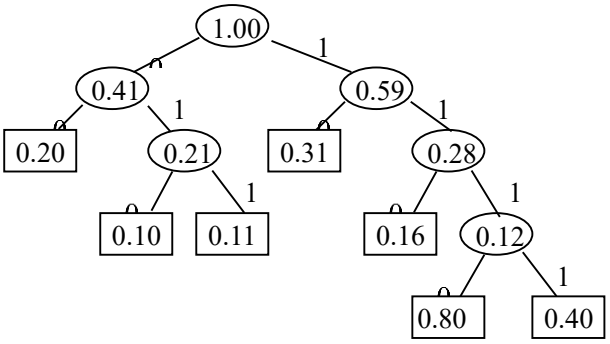
c：010

d：1110

e：011

f：00

g：1111



（2）对这 7 个字母进行等长编码，至少需要 3 位二进制数。

等长编码的平均长度：0.31*3+0.16*3+0.10*3+0.08*3+0.11*3+0.20*3+0.04*3=3

哈夫曼编码：0.31*2+0.16*3+0.10*3+0.08*4+0.11*3+0.20*2+0.04*4=2.54

哈夫曼编码比等长编码使电文总长压缩了：1 - 2.54/3=15.33%

2. 算法设计题

1．给定一棵用二叉链表表示的二叉树，其根指针为 root，试写出求二叉树结点的数目的算法。

【提示】采用递归算法实现。

$$\text{二叉树结点的数目}=\begin{cases}0 & \text{当二叉树为空}\\ \text{左子树结点数目}+\text{右子树结点数目}+1 & \text{当二叉树非空}\end{cases}$$

```
int count(BiTree root)
    {if(root==NULL)   return (0);
     else    return (count(root->lchild)+count(root->rchild)+1);
    }
```

2. 请设计一个算法，要求该算法把二叉树的叶结点按从左至右的顺序链成一个单链表。二叉树按 lchild-rchild 方式存储，链接时用叶结点的 rchild 域存放链指针。

【提示】这是一个非常典型的基于二叉树遍历算法，通过遍历找到各个叶子结点，因为不论前序遍历、中序遍历和后序遍历，访问叶子结点的相对顺序都是相同的，即都是从左至右。而题目要求是将二叉树中的叶子结点按从左至右顺序建立一个单链表，因此，可以采用三种遍历中的任意一种方法遍历。这里使用中序递归遍历。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，遍历到叶子结点时，就将它前驱的 rchild 指针指向它，

最后叶子结点的 rchild 为空。

```
LinkList head,pre=NULL;                                    /*全局变量*/
LinkList InOrder(BiTree T)
 /*中序遍历二叉树 T，将叶子结点从左到右链成一个单链表，表头指针为 head*/
  {if(T){InOrder(T->lchild);                               /*中序遍历左子树*/
        if(T->lchild==NULL && T->rchild==NULL)             /*当前是叶子结点*/
          if(pre==NULL)
              {head=T;
               pre=T;}                                     /*处理第一个叶子结点*/
          else {pre->rchild=T;
               pre=T;}                                     /*将叶子结点链入链表*/
   InOrder(T->rchild);                                     /*中序遍历右子树*/
   pre->rchild=NULL;                                       /*设置链表尾结点*/
   return(head);
  }
```

3．给定一棵用二叉链表表示的二叉树，其根指针为 root，试写出求二叉树的深度的算法。

【提示】采取递归算法。

```
int Height(BiTree root)
    {int hl,hr;
     if (root==NULL) return(0);
     else{hl=Height(root->lchild);
          hr=Height(root->rchild);
          if (hl>hr) return (hl+1);
          else return(hr+1);
          }
    }
```

4．给定一棵用二叉链表表示的二叉树，其根指针为 root，试求二叉树各结点的层数。

【提示】采用先序递归遍历算法实现。

$$\text{二叉树结点的层次数}=\begin{cases}1 & \text{当结点为根结点}\\ \text{其双亲结点的层次数}+1 & \text{当结点非根结点}\end{cases}$$

```
void fun (BiTree root,int n)
    {if(t==NULL)    return;
     else{printf("%d",n);
          fun(root->lchild,n+1);
          fun(root->rchild,n+1);
         }
    }
```

5．给定一棵用二叉链表表示的二叉树，其根指针为 root，试写出将二叉树中所有结点的左、右子树相互交换的算法。

【提示】设 root 为一棵用二叉链表存储的二叉树，则交换各结点的左右子树的运算可基于后序遍历实现：交换左子树上各结点的左右子树；交换左子树上各结点的左右子树；再交换根结点的左右子树。

```
void Exchange(BiTree root)
    {BiTNode *p;
     if(root)
        {Exchange(root->lchild);
         Exchange(root->rchild);
         p=root->lchild;
         root->lchild=root->rchild;
         root->rchild=p;
        }
     }
```

6．一棵具有 n 个结点的完全二叉树采用顺序结构存储，试设计非递归算法对其进行先序遍历。

【提示】二叉树的顺序存储是按完全二叉树的顺序存储格式按层次存储的，双亲结点与子女结点的下标间有确定的关系。对顺序存储结构的二叉树进行先序遍历，与二叉链表存放二叉树的遍历策略类似。但是在顺序存储结构下，判二叉树结点为空的条件为：结点下标大于 n，或结点值为 0（一般二叉树中的“虚结点”）。

```
void PreOrder(datatype data[n+1])                    /*0 号单元未用*/
    {int stack[n] ;
     int top;
     if(n<1)    return;
     t=1;
     top=0;
     while (t<=n||top>0)
       {while (t<=n&&data[t]!=0)
          {Visite(data[t]);
           stack[top]=t;    top++;
           t=t*2;}
           if(top<=0)    return;
           else {top--; t=stack[top];
                 t=t*2+1;}
       }
    }
```

7．二叉树中查找值为 x 的结点，试设计打印值为 x 的结点的所有祖先结点算法。

【提示】对二叉树进行先序非递归遍历，查找值为 x 的结点。进入子树时，将子树的根压栈；从子树返回时，将栈顶元素出栈。当找到值为 x 的结点时，栈中存放的就是其祖先结点，依次打印各结点的值。

```
void PrintNode(BiTree T, datatype x)
    {Init_Stack(S);                          /*初始化空栈*/
     p=T;
     while (p|| !Empty_Stack(S))             /*若 p 非空，或栈非空*/
        {while (p)
           {if(p->data==x)                   /*若当前结点值为 x，依次输出栈中元素的值*/
              {while(!Empty_Stack(S))
```

```
                {Pop(S,q);
                  printf(q->data);}
               return;
              }
            else{Push_Stack(S,p);}                    /*若当前结点值不是 x，压栈*/
                p=p->lchild;
              }
        if(!Empty_Stack(S))
            {Pop_Stack(S,r);                          /*当栈非空，栈顶元素出栈，进入右子树*/
             p=r->rchild; }
        else return;
      }
  }
```

8. 已知一棵二叉树的后序遍历序列和中序遍历序列，写出可以确定这棵二叉树的算法。

【提示】根据后序遍历和中序遍历的特点，采用递归算法实现。

```
void InPost (char in[ ], char post[ ], int il, int ir, int pl, int pr, BiTree t)
 /*数组 in 和数组 post 中存放着二叉树的中序遍历序列和后序遍历序列，il 和 ir 表示中序遍历序列的左右端
  */
 /*点，pl 和 pr 表示后序遍历序列的左右端点，t 表示二叉树的根*/
    {t=(BiTNode *)malloc(sizeof(BiTNode));
     t->data=post[pr];
     m=il;
     while (in[m]!=post [pr] )    m++;
     if(m== il)       t->lchild=NULL ;
      else    InPost ( in, post,il,m-1,pl,pl+m-1-il, t->lchild);
     if(m==ir)        t->rchild=NULL;
     else    InPost (in,post,m+1,ir,pr-ir+m,pr-1,t->rchild);
    }
```

9. 编写算法判断一棵二叉链表表示的二叉树是否是完全二叉树。

【提示】根据完全二叉树的定义可知，对完全二叉树按照从上到下，从左到右的次序遍历应满足：①若某结点没有左孩子，则一定无右孩子；②若某结点缺（左或右）孩子，则其所有后继一定无孩子。因此，可采用按层次遍历二叉树的方法依次对每个结点进行判断。这里增加一个标志以表示所有已扫描过的结点均有左、右孩子，将局部判断结果存入 CM 中，CM 表示整个二叉树是否是完全二叉树，B 为 1 表示到目前为止所有结点均有左右孩子。

```
int CompleteBT(BiTree T)
   {Init_Queue(Q);                                    /*初始化队列 Q*/
    B=1;CM=1;
    if(T!=NULL)
       {In_Queue(Q,T);
        while(!Empty_Queue(Q))                        /*当队列不为空时执行循环*/
          {p=Out_Queue(Q);
           if(p->lchild==NULL)
              {B=0;                                   /*B=0 表示缺少左、右孩子*/
```

```
            if(rchild!=NULL)
            CM=0;                                   /*CM=0 表示不是完全二叉树*/
           }
       else{CM=B;
            In_Queue(Q,p->lchild);                  /*左孩子入队列*/
            if(p->rchild==NULL)    B=0;
            else    In_Queue(Q,p->rchild);          /*右孩子入队列*/
          }
       }
    return(CM);
    }
                              }
```

10．有 n 个结点的完全二叉树存放在一维数组 A[1..n]中，试据此建立一棵用二叉链表表示的二叉树。

```
BiTree Create(dataype A[],int i)
 /*n 个结点的完全二叉树存于一维数组 A 中，据此建立以二叉链表表示的完全二叉树,初始调用时,i=1*/
  {BiTree T;
   if (i<=n)
      {T=(BiTree)malloc(sizeof(BiTNode));
       T->data=A[i];
       if (2*i>n) T->lchild=NULL;
       else T->lchild=Create(A,2*i);
       if (2*i+1>n) T->rchild=NULL;
       else T->rchild=Create(A,2*i+1);
      }
   return (T);
  }
```

11．编写算法判定两棵二叉树是否相似。所谓两棵二叉树 s 和 t 相似，即要么它们都为空或都只有一个结点，要么它们的左右子树都相似。

【提示】两棵空二叉树或仅有根结点的二叉树相似；对非空二叉树，可判左右子树是否相似，采用递归算法。

```
int Similar(BiTree s, BiTree t)
  {if(s==NULL && q==NULL) return (1);
   else if(!s&&t||s&&!t) return (0);
        else return(Similar(s->lchild,t->lchild)&&Similar(s->rchild,t->rchild));
  }
```

12．写出用逆转链方法对二叉树进行先序遍历的算法。

【提示】用逆转链的方法遍历二叉树，不需要附加的栈空间，而是在遍历过程中沿结点的左右子树下降时，临时改变结点 lchild 或者 rchild 的值，使之指向双亲，从而为以后的上升提供路径，上升时再将结点的 lchild 或者 rchild 恢复。

```
typedef struct tnode
  {datatype data;
   int tag;                 /*tag 的值初始为 0，进入左子树时置 1，从左子树回来时，再恢复为 0*/
```

```
    struct tnode    *lchild, *rchild;
  }Bnode, *Btree;
void PreOrder(Btree    bt)
  {Bnode *r,*p,*q;       /*p 指向当前结点，r 指向 p 的双亲，q 指向要访问的下一结点*/
   if(bt==NULL)     return;
   p=bt; r= NULL:
   while( p)
     {Visit(p);                                              /*访问 p 所指结点*/
      if(p->lchild)                                          /*下降进入左子树*/
          {p->tag=1; q=p->lchild;
           q=p->lchild=r; r=p; p=q; }
      else if(p->rchild)                                     /*下降进入右子树*/
              {q=p->rchild; p->rchild=r;
               r=p; p=q;}
           else {                                            /*上升*/
              while((r&&t->tag==0)||(r&&t->tag==1&&r->rchild==NULL))
                {if(r&&t->tag==0)
                   {q=r->rchild; r->rchild=p;
                    p=r; r=q;}                               /*从右子树回来*/
                 else {r->tag=0; q=r->lchild;
                       r->lchild=p; p=r; r=q;}               /*从左子树回来*/
                 if(r==NULL) return;
                 else {r->tag=0; q=r->rchild;
                       r->rchild= r->lchild; r->lchild=p; p=q;
                      }                                      /*从左子树回来，准备进入右子树*/
                }
              }
     }
  }
```

13．对以孩子－兄弟链表表示的树编写计算树的深度的算法。

【提示】采用递归算法实现。

$$\text{树的深度}=\begin{cases}0 & \text{若树为空}\\ \text{Max（第一棵子树的深度}+1\text{，兄弟子树的深度）} & \text{若树非空}\end{cases}$$

```
int high(CSTree T)
{if (T==NULL )return ( 0 );              /*若树为空，返回 0*/
 else {h1=high (t->lchild );               /*h1 为 T 的第一棵子树的深度*/
       h2=high(t->nextsibling );           /*h2 为 t 的兄弟子树的深度*/
       return(max(h1+1,h2));
      }
}
```

14．对以孩子链表表示的树编写计算树的深度的算法。

【提示】采用递归算法实现。

$$\text{树的深度}=\begin{cases}1 & \text{若根结点没有子树}\\ \max(\text{所有子树的深度})+1 & \text{若根结点有子树}\end{cases}$$

```
#define MAXNODE     树中结点的最大个数
int high(SNode t[MAXNODE], int j)
    {if(t[j].firstchild==NULL)    return(1);              /*若根结点没有子树*/
     else{p=t[i].firstchild;
          max=high(t,p->data);
          p=p->nextchild;
          while(p)
              {h=high(t,p->data);
               if(h>max) max=h;
               p=p->nextchild;
              }
          return(max+1);
         }
    }
```

15．对以双亲链表表示的树编写计算树的深度的算法。

【提示】从每一个结点开始，从下向上搜索至根结点，记录结点的层次数，其中的最大值就是树的深度。

```
int high(PNode t[ ], int n)                          /*求有 n 个结点的树 t 的深度*/
    {maxh=0;
     for (i=0 ;i<n ;i++)
         {if(t[i].parent==-1 ) h=1;
          else{s=i;
               h=1;
               while (t[s].parent!=-1 )
                  {s=t[s].parent ;
                   h++;}
                  }
          if(h>maxh)       maxh=h;
          }
     return(maxh);
    }
```