习题六　树和二叉树

一、单项选择题

1．以下说法错误的是（　　）

A．树形结构的特点是一个结点可以有多个直接前趋

B．线性结构中的一个结点至多只有一个直接后继

C．树形结构可以表达(组织)更复杂的数据

D．树(及一切树形结构)是一种"分支层次"结构

E．任何只含一个结点的集合是一棵树

2．下列说法中正确的是（　　）

A．任何一棵二叉树中至少有一个结点的度为 2

B．任何一棵二叉树中每个结点的度都为 2

C．任何一棵二叉树中的度肯定等于 2

D．任何一棵二叉树中的度可以小于 2

3．讨论树、森林和二叉树的关系，目的是为了（　　）

A．借助二叉树上的运算方法去实现对树的一些运算

B．将树、森林按二叉树的存储方式进行存储

C．将树、森林转换成二叉树

D．体现一种技巧，没有什么实际意义

4．树最适合用来表示（　　）

A．有序数据元素　　B．无序数据元素

C．元素之间具有分支层次关系的数据　　D．元素之间无联系的数据

5．若一棵二叉树具有 10 个度为 2 的结点，5 个度为 1 的结点，则度为 0 的结点个数是（　）

A．9　　B．11　　C．15　　D．不确定

6．设森林 F 中有三棵树，第一，第二，第三棵树的结点个数分别为 M1，M2 和 M3。与森林 F 对应的二叉树根结点的右子树上的结点个数是（　　）。

A．M1　　B．M1+M2　　C．M3　　D．M2+M3

7．一棵完全二叉树上有 1001 个结点，其中叶子结点的个数是（　　）

A．250　B．500　C．254　D．505　E．以上答案都不对

8．设给定权值总数有 n 个，其哈夫曼树的结点总数为(　　)

A．不确定　　B．2n　　C．2n+1　　D．2n-1

9．二叉树的第 I 层上最多含有结点数为（　　）

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

10．一棵二叉树高度为 h,所有结点的度或为 0，或为 2，则这棵二叉树最少有(　　)结点

A．2h　　B．2h-1　　C．2h+1　　D．h+1

11．利用二叉链表存储树，则根结点的右指针是（　　）。

A．指向最左孩子　　B．指向最右孩子　　C．空　　D．非空

12．已知一棵二叉树的前序遍历结果为 ABCDEF,中序遍历结果为 CBAEDF,则后序遍历的结果为（　　）。

A．CBEFDA　　B．FEDCBA　　C．CBEDFA　　D．不定

13．已知某二叉树的后序遍历序列是 dabec，中序遍历序列是 debac，它的前序遍历是（　　）。

A．acbed　　B．decab　　C．deabc　　D．cedba

14．在二叉树结点的先序序列，中序序列和后序序列中，所有叶子结点的先后顺序（　　）

A．都不相同　　B．完全相同

C．先序和中序相同，而与后序不同　　D．中序和后序相同，而与先序不同

15．在完全二叉树中，若一个结点是叶结点，则它没（　　）。

A．左子结点　　B．右子结点

C．左子结点和右子结点　　D．左子结点，右子结点和兄弟结点

16．在下列情况中，可称为二叉树的是（　　）

A．每个结点至多有两棵子树的树

B. 哈夫曼树

C．每个结点至多有两棵子树的有序树

D. 每个结点只有一棵右子树

E．以上答案都不对

17. 一棵左右子树均不空的二叉树在先序线索化后，其中空的链域的个数是：（　　）。

A. 0　　B. 1　　C. 2　　D. 不确定

18. 引入二叉线索树的目的是（　　）

A．加快查找结点的前驱或后继的速度　　B．为了能在二叉树中方便的进行插入与删除

C．为了能方便的找到双亲　　D．使二叉树的遍历结果唯一

19．n 个结点的线索二叉树上含有的线索数为（　　）

A．2n　　B．n－1　　C．n＋1　　D．n

20．由 3 个结点可以构造出多少种不同的二叉树？（　　）

A．2　　B．3　　C．4　　D．5

21．下面几个符号串编码集合中，不是前缀编码的是（　　）。

A．{0,10,110,1111}　　B．{11,10,001,101,0001}

C．{00,010,0110,1000}　　D．{b,c,aa,ac,aba,abb,abc}

22. 一棵有 n 个结点的二叉树，按层次从上到下，同一层从左到右顺序存储在一维数组 A[1..n]中，则二叉树中第 i 个结点（i 从 1 开始用上述方法编号）的右孩子在数组 A 中的位置是（　　）

A．A[2i](2i<=n)　　B．A[2i+1](2i+1<=n)

C．A[i-2]　　D．条件不充分，无法确定

23、以下说法错误的是（　　）

A．哈夫曼树是带权路径长度最短的树，路径上权值较大的结点离根较近。

B．若一个二叉树的树叶是某子树的中序遍历序列中的第一个结点，则它必是该子树的后序遍历序列中的第一个结点。

C．已知二叉树的前序遍历和后序遍历序列并不能惟一地确定这棵树，因为不知道树的根结点是哪一个。

D．在前序遍历二叉树的序列中，任何结点的子树的所有结点都是直接跟在该结点的之后。

二、判断题（在各题后填写“√”或“×”）

1. 完全二叉树一定存在度为 1 的结点。（　　）

2．对于有 N 个结点的二叉树，其高度为 $\log_2 n$。（　　）

3. 二叉树的遍历只是为了在应用中找到一种线性次序。（　　）

4. 一棵一般树的结点的前序遍历和后序遍历分别与它相应二叉树的结点前序遍历和后序遍历是一致的。（　　）

5. 用一维数组存储二叉树时，总是以前序遍历顺序存储结点。(　　)
6．中序遍历一棵二叉排序树的结点就可得到排好序的结点序列。(　　)
7．完全二叉树中，若一个结点没有左孩子，则它必是树叶。(　　)
8. 二叉树只能用二叉链表表示。(　　)
9. 给定一棵树，可以找到唯一的一棵二叉树与之对应。(　　)
10. 用链表(llink-rlink)存储包含 n 个结点的二叉树，结点的 2n 个指针区域中有 n-1 个空指针。(　　)
11．树形结构中元素之间存在一个对多个的关系。(　　)
12．将一棵树转成二叉树，根结点没有左子树。(　　)
13．度为二的树就是二叉树。(　　)
14. 二叉树中序线索化后，不存在空指针域。(　　)
15．霍夫曼树的结点个数不能是偶数。(　　)
16．哈夫曼树是带权路径长度最短的树，路径上权值较大的结点离根较近。(　　)

三、填空题

1．在二叉树中，指针 p 所指结点为叶子结点的条件是_____。

2．深度为 k 的完全二叉树至少有_____个结点，至多有_____个结点。

3．高度为 8 的完全二叉树至少有_____个叶子结点。

4.具有 n 个结点的二叉树中,一共有_____个指针域,其中只有_____个用来指向结点的左右孩子，其余的_____个指针域为 NULL。

5．树的主要遍历方法有_____、_____、_____等三种。

6．一个深度为 k 的，具有最少结点数的完全二叉树按层次，(同层次从左到右）用自然数依此对结点编号，则编号最小的叶子的序号是_____；编号是 i 的结点所在的层次号是_____（根所在的层次号规定为 1 层)。

7．如果结点 A 有 3 个兄弟，而且 B 是 A 的双亲，则 B 的度是_____。

8．二叉树的先序序列和中序序列相同的条件是_____。

9．一个无序序列可以通过构造一棵_____树而变成一个有序序列，构造树的过程即为对无序序列进行排序的过程。

10．若一个二叉树的叶子结点是某子树的中序遍历序列中的最后一个结点，则它必是该子树的_____序列中的最后一个结点。

11．若以{4，5，6，7，8}作为叶子结点的权值构造哈夫曼树，则其带权路径长度是_____。

12．以下程序段采用先根遍历方法求二叉树的叶子数，请在横线处填充适当的语句。

```
Void countleaf(bitreptr t,int *count)/*根指针为 t，假定叶子数 count 的初值为
```

```
0*/
      {if(t!=NULL)
         {if((t->lchild==NULL)&&(t->rchild==NULL))_______;
          countleaf(t->lchild,&count);
          _______
         }
      }
```

13．以下程序是二叉链表树中序遍历的非递归算法，请填空使之完善。二叉树链表的结点类型的定义如下：

```
   typedef struct node   /*C 语言/
    {char data; struct  node *lchild,*rchild;}*bitree;
   void vst(bitree bt)                    /*bt 为根结点的指针*/
    { bitree  p; p=bt;  initstack(s);     /*初始化栈 s 为空栈*/
     while(p || !empty(s))               /*栈 s 不为空*/
      if(p) { push (s,p); (1)              ; }      /*P 入栈*/
      else { p=pop(s); printf(“%c”,p->data);(2)__________; }/*栈顶元素出栈*/
    }
```

14．二叉树存储结构同上题，以下程序为求二叉树深度的递归算法，请填空完善之。

```
     int depth(bitree bt)   /*bt 为根结点的指针*/
     {int hl,hr;
      if (bt==NULL) return((1)                );
      hl=depth(bt->lchild); hr=depth(bt->rchild);
      if((2)_____________)  (3)_____________;
      return(hr+1);
}
```

15．将二叉树 bt 中每一个结点的左右子树互换的 C 语言算法如下，其中 ADDQ(Q,bt),DELQ(Q),EMPTY(Q)分别为进队，出队和判别队列是否为空的函数，请填写算法中得空白处，完成其功能。

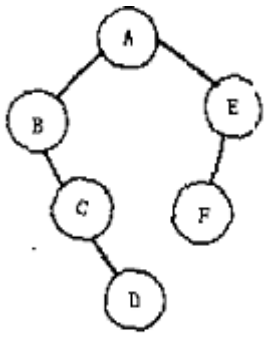
```
    typedef struct node
       {int data ; struct node *lchild, *rchild; }btnode;
    void  EXCHANGE(btnode *bt)
    {btnode *p, *q;
     if (bt){ADDQ(Q,bt);
             while(!EMPTY(Q))
              {p=DELQ(Q);  q=(1)_____; p->rchild=(2)_____; (3)____=q;
               if(p->lchild) (4)_____; if(p->rchild) (5)_____;
               }
    }   }//
```

四、应用题

1．树和二叉树之间有什么样的区别与联系？

2．分别画出具有 3 个结点的树和 3 个结点的二叉树的所有不同形态。

3．分别给出下图所示二叉树的先根、中根和后根序列。



4．一个深度为 L 的满 K 叉树有以下性质：第 L 层上的结点都是叶子结点，其余各层上每个结点都有 K 棵非空子树，如果按层次顺序从 1 开始对全部结点进行编号，求：

1）各层的结点的数目是多少？

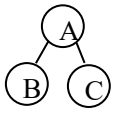
2）编号为 n 的结点的双亲结点（若存在）的编号是多少？

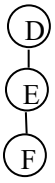
3）编号为 n 的结点的第 i 个孩子结点（若存在）的编号是多少？

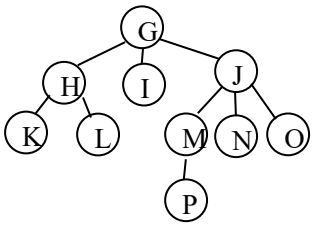
4）编号为 n 的结点有右兄弟的条件是什么？如果有，其右兄弟的编号是多少？

请给出计算和推导过程。

5．将下列由三棵树组成的森林转换为二叉树。（只要求给出转换结果）







6．设二叉树 BT 的存储结构如下:

	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

其中 BT 为树根结点的指针，其值为 6,Lchild,Rchild 分别为结点的左、右孩子指针域,data 为结点的数据域。试完成下列各题:

（1）画出二叉树 BT 的逻辑结构;

（2）写出按前序、中序、后序遍历该二叉树所得到的结点序列;

（3）画出二叉树的后序线索树。

7．设有正文 AADBAACACCDACACAAD,字符集为 A,B,C,D,设计一套二进制编码，使得上述正文的编码最短。

五、算法设计题

1．要求二叉树按二叉链表形式存储，

（1）写一个建立二叉树的算法。

（2）写一个判别给定的二叉树是否是完全二叉树的算法。

完全二叉树定义为：深度为K，具有N个结点的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。此题以此定义为准。

2．设一棵二叉树的结点结构为 (LLINK,INFO,RLINK),ROOT为指向该二叉树根结点的指针，p和q分别为指向该二叉树中任意两个结点的指针，试编写一算法ANCESTOR(ROOT,p,q,r)，该算法找到p和q的最近共同祖先结点r。

3．有一二叉链表，试编写按层次顺序遍历二叉树的算法。

4．已知二叉树按照二叉链表方式存储，利用栈的基本操作写出先序遍历非递归形式的算法。

5．对于二叉树的链接实现,完成非递归的中序遍历过程。

6．试写出复制一棵二叉树的算法。二叉树采用标准链接结构。。

7．请设计一个算法，要求该算法把二叉树的叶子结点按从左到右的顺序连成一个单链表，表头指针为head。 二叉树按二叉链表方式存储，链接时用叶子结点的右指针域来存放单链表指针。分析你的算法的时、空复杂度。

8．已知二叉树以二叉链表存储，编写算法完成：对于树中每一个元素值为x的结点，删去以它为根的子树，并释放相应的空间。

9．设一棵二叉树的根结点指针为T，C为计数变量，初值为0，试写出对此二叉树中结点计数的算法：BTLC（T，C）。

10．分别写出算法，实现在中序线索二叉树 T 中查找给定结点*p 在中序序列中的前驱与后继。在先序线索二叉树 T 中，查找给定结点*p 在先序序列中的后继。在后序线索二叉树 T 中，查找给定结点*p 在后序序列中的前驱。

第六章　树和二叉树

一、单项选择题

1.A
2.D
3．A
4．C
5．B
6．D
7．E
8．D
9．C
10．B
11．C
12．A
13．D
14．B
15．C
16．B
17．B
18．A
19．C
20．D
21．B
22．D
23．C

二、判断题（在各题后填写“√”或“×”）

1. 完全二叉树一定存在度为 1 的结点。×
2. 对于有 N 个结点的二叉树，其高度为 $\log_2 n$。×
3. 二叉树的遍历只是为了在应用中找到一种线性次序。√
4. 一棵一般树的结点的前序遍历和后序遍历分别与它相应二叉树的结点前序遍历和后序遍历是一致的。×
5. 用一维数组存储二叉树时，总是以前序遍历顺序存储结点。×
6．中序遍历一棵二叉排序树的结点就可得到排好序的结点序列　√
7．完全二叉树中，若一个结点没有左孩子，则它必是树叶。√
8. 二叉树只能用二叉链表表示。×
9. 给定一棵树，可以找到唯一的一棵二叉树与之对应。√
10. 用链表(llink-rlink)存储包含 n 个结点的二叉树，结点的 2n 个指针区域中有 n-1 个空指针。×
11．树形结构中元素之间存在一个对多个的关系。√
12．将一棵树转成二叉树，根结点没有左子树。×
13．度为二的树就是二叉树。×
14. 二叉树中序线索化后，不存在空指针域。×

15．霍夫曼树的结点个数不能是偶数。√
16．哈夫曼树是带权路径长度最短的树，路径上权值较大的结点离根较近。√

三、填空题

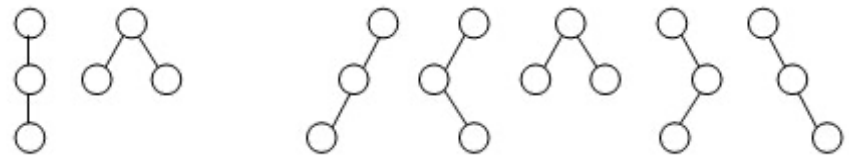
1．p->lchild==null && p->rchlid==null
2.(1)2k-1　(2)2k-1
3．64
4．2n　　n-1　　n+1
5．　先序遍历　　后序遍历　　中序遍历
6..(1)2k-2+1(第 k 层 1 个结点，总结点个数是 2H-1，其双亲是 2H-1/2=2k-2)(2)⌊log2i⌋+1
7．4
8．任何结点至多只有右子女的二叉树。
9．二叉排序树
10．前序
11．69
12．　*count++,　　countleaf(l->rchile,count)
13．(1) p=p->lchild　//　沿左子树向下　(2) p=p->rchild
14．(1)0　　(2) hl>hr　　(3)hr=hl
15．(1)p->rchild　(2)p->lchild　(3)p->lchild
　　(4)ADDQ(Q,p->lchild)　(5)ADDQ(Q,p->rchild)

四、应用题

1．树和二叉树逻辑上都是树形结构，树和二叉树的区别有三：一是二叉树的度至多为 2，树无此限制；二是二叉树有左右子树之分，即使在只有一个分枝的情况下，　也必须指出是左子树还是右子树，树无此限制；三是二叉树允许为空，树一般不允许为空（个别书上允许为空）。二叉树不是树的特例。

2.【解答】

具有 3 个结点的树　　具有 3 个结点的二叉树

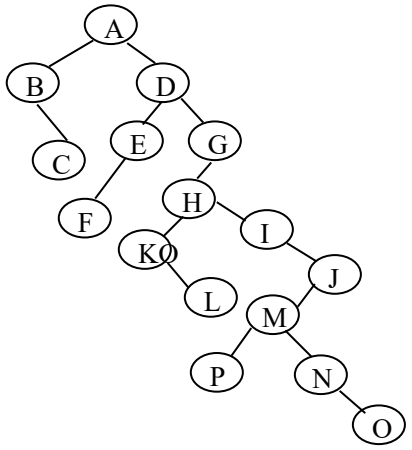


3．解答：先根序列：A B C D E F G H I J;
　　中根序列：B C D A F E H J I G;
　　后根序列：D C B F J I H G E A。

4．(1)k^{h-1}(h 为层数)
　(2) 因为该树每层上均有 K^{h-1} 个结点，从根开始编号为 1，则结点 i 的从右向左数第 2 个孩子的结点编号为 ki。设 n 为结点 i 的子女，则关系式(i-1)k+2<=n<=ik+1 成立，因 i 是整数，故结点 n 的双亲 i 的编号为⌊n-2)/k⌋+1。
　(3) 结点 n(n>1)的前一结点编号为 n-1（其最右边子女编号是(n-1)*k+1)，故结点 n 的第 i 个孩子的编号是(n-1)*k+1+i。

(4) 根据以上分析，结点 n 有右兄弟的条件是，它不是双亲的从右数的第一子女，即(n-1)%k!=0，其右兄弟编号是 n+1。

5.



6.（1）图略；

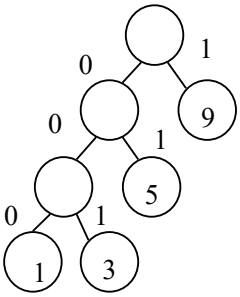
（2） 前序序列：A B C E D F H G I J

中序序列： E C B H F D J I G A

后序序列： E C H F J I G D B A

（3）图略。

7. 字符 A，B，C，D 出现的次数为 9,1,5,3。其哈夫曼编码如下 A:1，B:000，C:01，D:001



五、算法设计题

1.[题目分析]二叉树是递归定义的，以递归方式建立最简单。判定是否是完全二叉树，可以使用队列，在遍历中利用完全二叉树“若某结点无左子女就不应有右子女”的原则进行判断。

```
BiTree Creat()        //建立二叉树的二叉链表形式的存储结构
{ElemType x; BiTree bt;
  scanf(“%d”,&x);  //本题假定结点数据域为整型
  if(x==0) bt=null;
  else if(x>0)
        {bt=(BiNode *)malloc(sizeof(BiNode));
         bt->data=x; bt->lchild=creat(); bt->rchild=creat();
        }
       else error(“输入错误”);
```

```
  return(bt);
 }//结束 BiTree
int JudgeComplete(BiTree bt)  //判断二叉树是否是完全二叉树,如是，返回 1，否则，返回 0
      {int tag=0; BiTree p=bt, Q[]; // Q 是队列，元素是二叉树结点指针，容量足够大
       if(p==null) return (1);
       QueueInit(Q); QueueIn(Q,p);  //初始化队列，根结点指针入队
       while (!QueueEmpty(Q))
         {p=QueueOut(Q);                                    //出队
          if (p->lchild && !tag) QueueIn(Q,p->lchild);  //左子女入队
          else {if (p->lchild) return 0;                  //前边已有结点为空，本结点不空
                else tag=1;                                 //首次出现结点为空
          if (p->rchild && !tag) QueueIn(Q,p->rchild);  //右子女入队
          else if (p->rchild) return 0; else tag=1;
          } //while
       return 1;  } //JudgeComplete
```

[算法讨论]完全二叉树证明还有其它方法。判断时易犯的错误是证明其左子树和右子数都是完全二叉树，由此推出整棵二叉树必是完全二叉树的错误结论。

2.[题目分析]后序遍历最后访问根结点，即在递归算法中，根是压在栈底的。采用后序非递归算法，栈中存放二叉树结点的指针，当访问到某结点时，栈中所有元素均为该结点的祖先。本题要找 p 和 q 的最近共同祖先结点 r，不失一般性，设 p 在 q 的左边。后序遍历必然先遍历到结点 p，栈中元素均为 p 的祖先。将栈拷入另一辅助栈中。再继续遍历到结点 q 时，将栈中元素从栈顶开始逐个到辅助栈中去匹配，第一个匹配（即相等）的元素就是结点 p 和 q 的最近公共祖先。

```
typedef struct
 {BiTree t;int tag;//tag=0  表示结点的左子女已被访问，tag=1 表示结点的右子女已被访问
 }stack;
stack s[],s1[];//栈，容量够大
BiTree Ancestor(BiTree ROOT,p,q,r)//求二叉树上结点 p 和 q 的最近的共同祖先结点 r。
 {top=0; bt=ROOT;
  while(bt!=null ||top>0)
  {while(bt!=null && bt!=p && bt!=q)              //结点入栈
    {s[++top].t=bt; s[top].tag=0; bt=bt->lchild;} //沿左分枝向下
   if(bt==p)  //不失一般性，假定 p 在 q 的左侧,遇结点 p 时，栈中元素均为 p 的祖先结点
     {for(i=1;i<=top;i++) s1[i]=s[i]; top1=top; }//将栈 s 的元素转入辅助栈 s1 保存
   if(bt==q)  //找到 q 结点。
      for(i=top;i>0;i--)//；将栈中元素的树结点到 s1 去匹配
       {pp=s[i].t;
```

```
        for (j=top1;j>0;j--)
          if(s1[j].t==pp) {printf("p 和 q 的最近共同的祖先已找到");return (pp);}
        }
    while(top!=0 && s[top].tag==1) top--;       //退栈
    if (top!=0) {s[top].tag=1;bt=s[top].t->rchild;}  //沿右分枝向下遍历
   }//结束 while(bt!=null ||top>0)
   return(null);// q 、p 无公共祖先
  } //结束 Ancestor
```

3．解答：本算法要借用队列来完成，其基本思想是，只要队列不为空，就出队列，然后判断该结点是否有左孩子和右孩子，如有就依次输出左、右孩子的值，然后让左、右孩子进队列。

```
    void layorder (bitreptr T)
     {initqueue (q)                              /*队列初始化*/
      if(T!=NULL)
        {printf("%f", T->data);
         enqueue (q, T);                         /*入队列*/
         while (not emptyqueue (q) )             /*若队列非空*/
           {outqueue (q, p) ;                    /*出队*/
            if (p->lchild!=NULL)
               {printf("%f", p->lchild->data);
                enqueue (q, p->lchild);          /*入队列*/
               }
            if (p->rchild!=NULL)
              {printf("%", p->rchild->data);
               enqueue (q, p->rchild);           /*入队列*/
              }
           }
        }
 }
```

4.【解答】

```
    Void  PreOrder(BiTree  root)    /*先序遍历二叉树的非递归算法*/
    {
      InitStack(&S);
      p=root;
      while(p!=NULL || !IsEmpty(S) )
      { if(p!=NULL)
        {
          Visit(p->data);
          push(&S,p);
          p=p->Lchild;
        }
       else
        {
```

```
            Pop(&S,&p);
            p=p->RChild;
        }
      }
    }
```

5.. void InOrder(BiTree bt)

```
    {BiTree s[],p=bt;  //s 是元素为二叉树结点指针的栈，容量足够大
     int top=0;
     while(p || top>0)
        {while(p) {s[++top]=p; bt=p->lchild;}              //中序遍历左子树
         if(top>0){p=s[top--]; printf(p->data); p=p->rchild;} //退栈，访问，转右子树
        }   }
```

6. BiTree Copy(BiTree t)//复制二叉树 t

```
   {BiTree bt;
    if (t==null) bt=null;
    else{bt=(BiTree)malloc(sizeof(BiNode)); bt->data=t->data;
         bt->lchild=Copy(t->lchild);
         bt->rchild=Copy(t->rchild);
         }
    return(bt); }//结束 Copy
```

7.[题目分析]叶子结点只有在遍历中才能知道，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。

```
    LinkedList head,pre=null;  //全局变量
    LinkedList InOrder(BiTree bt)
      //中序遍历二叉树 bt，将叶子结点从左到右链成一个单链表，表头指针为 head
     {if(bt){InOrder(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
             InOrder(bt->rchild);                 //中序遍历左子树
             pre->rchild=null;                    //设置链表尾
            }
       return(head);  } //InOrder
```

时间复杂度为 O(n)，辅助变量使用 head 和 pre，栈空间复杂度 O(n)

8.[题目分析] 删除以元素值 x 为根的子树，只要能删除其左右子树，就可以释放值为 x 的根结点，因此宜采用后序遍历。删除值为 x 结点，意味着应将其父结点的左(右)子女指针置

空，用层次遍历易于找到某结点的父结点。本题要求删除树中每一个元素值为 x 的结点的子树，因此要遍历完整棵二叉树。

```
void  DeleteXTree(BiTree  bt)  //删除以 bt 为根的子树
 {DeleteXTree(bt->lchild); DeleteXTree(bt->rchild);//删除 bt 的左子树、右子树
  free(bt);  }// DeleteXTree                          //释放被删结点所占的存储空间
void  Search(B:Tree bt,ElemType x)
  //在二叉树上查找所有以 x 为元素值的结点，并删除以其为根的子树
 {BiTree Q[];//Q 是存放二叉树结点指针的队列，容量足够大
  if(bt)
   {if(bt->data==x) {DeleteXTree(bt); exit(0);}//若根结点的值为 x，则删除整棵树
    {QueueInit(Q); QueueIn(Q,bt);
     while(!QueueEmpty(Q))
       {p=QueueOut(Q);
        if(p->lchild)                // 若左子女非空
          if(p->lchild->data==x)     //左子女结点值为 x,应删除当前结点的左子树
            {DeleteXTree(p->lchild); p->lchild=null;} //父结点的左子女置空
          else Enqueue (Q,p->lchild);// 左子女入队列
        if(p->rchild)                 // 若右子女非空
          if(p->rchild->data==x)      //右子女结点值为 x,应删除当前结点的右子树
            {DeleteXTree(p->rchild);   p->rchild=null;} //父结点的右子女置空
          else Enqueue (Q,p->rchild);// 右子女入队列
       }//while
     }//if(bt)  }//search
```

9.
```
int BTLC(BiTree T,int *c)//对二叉树 T 的结点计数
 {if(T)
    {*c++;
     BTLC(T->lchild,&c); //统计左子树结点
     BTLC(T->rchild,&c); //统计右子树结点
     } }//结束 Count,调用时*c=0
```

10.

（1）找结点的中序前驱结点

```
BiTNode  *InPre (BiTNode  *p)
/*在中序线索二叉树中查找 p 的中序前驱结点，并用 pre 指针返回结果*/
{ if (p->Ltag= =1)  pre = p->LChild;   /*直接利用线索*/
 else
   {/*在 p 的左子树中查找“最右下端”结点*/
    for ( q=p->LChild; q->Rtag= =0; q=q->RChild);
     pre = q;
```

```
        }
     return (pre);
    }
```

（2）找结点的中序后继结点

```
    BiTNode  *InSucc (BiTNode  *p)
    /*在中序线索二叉树中查找 p 的中序后继结点，并用 succ 指针返回结果*/
    { if (p->Rtag= =1)  succ = p->RChild;   /*直接利用线索*/
     else
       {/*在 p 的右子树中查找“最左下端”结点*/
         for ( q=p->RChild; q->Ltag= =0; q=q->LChild);
          succ= q;
        }
     return (succ);
    }
```

(3) 找结点的先序后继结点

```
    BiTNode  *PreSucc (BiTNode  *p)
    /*在先序线索二叉树中查找 p 的先序后继结点，并用 succ 指针返回结果*/
    { if (p->Ltag= =0)  succ = p->LChild;
     else   succ= p->RChild;
     return (succ);
    }
```

(4) 找结点的后序前驱结点

```
    BiTNode  *SuccPre (BiTNode  *p)
    /*在后序线索二叉树中查找 p 的后序前驱结点，并用 pre 指针返回结果*/
    { if (p->Ltag= =1)  pre = p->LChild;
     else   pre= p->RChild;
     return (pre);
}
```