一、下面是有关二叉树的叙述，请判断正误（）

（ √ ）1. 若二叉树用二叉链表作存贮结构，则在 n 个结点的二叉树链表中只有 n—1 个非空指针域。

（ × ）2.二叉树中每个结点的两棵子树的高度差等于 1。

（ √ ）3.二叉树中每个结点的两棵子树是有序的。

（ × ）4.二叉树中每个结点有两棵非空子树或有两棵空子树。

（ × ）5.二叉树中每个结点的关键字值大于其左非空子树（若存在的话）所有结点的关键字值，且小于其右非空子树（若存在的话）所有结点的关键字值。 （应当是二叉排序树的特点）

（ × ）6.二叉树中所有结点个数是 2^{k-1}-1，其中 k 是树的深度。（应 2^{i}-1）

（ × ）7.二叉树中所有结点，如果不存在非空左子树，则不存在非空右子树。

（ × ）8.对于一棵非空二叉树，它的根结点作为第一层，则它的第 i 层上最多能有 2^{i}—1 个结点。（应 2^{i-1}）

（ √ ）9.用二叉链表法（link-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 个为空指针。

（正确。用二叉链表存储包含 n 个结点的二叉树，结点共有 2n 个链域。由于二叉树中，除根结点外，每一个结点有且仅有一个双亲，所以只有 n-1 个结点的链域存放指向非空子女结点的指针，还有 n+1 个空指针。）即有后继链接的指针仅 n-1 个。

（ √ ）10.具有 12 个结点的完全二叉树有 5 个度为 2 的结点。

最快方法：用叶子数＝[n/2]＝6，再求 $n_2=n_0-1=5$

二、填空（）

1. 由 3 个结点所构成的二叉树有 5 种形态。

2. 一棵深度为 6 的满二叉树有 $n_1+n_2=0+n_2=n_0-1=31$ 个分支结点和 $2^{6-1}=32$ 个叶子。

注：满二叉树没有度为 1 的结点，所以分支结点数就是二度结点数。

3. 一棵具有 2 5 7 个结点的完全二叉树，它的深度为 9 。

（ 注：用⌊ $\log_2(n)$ ⌋+1=⌊ 8.xx ⌋+1=9

4. 设一棵完全二叉树有 700 个结点，则共有 350 个叶子结点。

答：最快方法：用叶子数＝[n/2]＝350

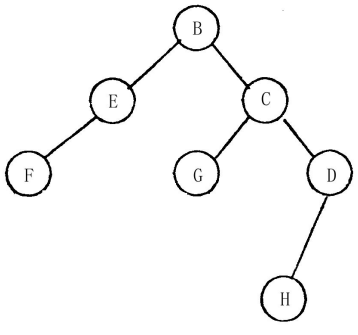
5. 设一棵完全二叉树具有 1000 个结点，则此完全二叉树有 500 个叶子结点，有 499 个度为 2 的结点，有 1 个结点只有非空左子树，有 0 个结点只有非空右子树。

答：最快方法：用叶子数＝[n/2]＝500 ，$n_2=n_0-1=499$。 另外，最后一结点为 2i 属于左叶子，右叶子是空的，所以有 1 个非空左子树。完全二叉树的特点决定不可能有左空右不空的情况，所以非空右子树数＝0.

6. 一棵含有 n 个结点的 k 叉树，可能达到的最大深度为 n ，最小深度为 2 。

答：当 k=1(单叉树)时应该最深，深度＝n（层）；当 k=n-1（n-1 叉树）时应该最浅，深度＝2（层），但不包括 n=0 或 1 时的特例情况。教材答案是“完全 k 叉树”，未定量。）

7. 二叉树的基本组成部分是：根（N）、左子树（L）和右子树（R）。因而二叉树的遍历次序有六种。最常用的是三种：前序法（即按 N L R 次序），后序法（即按 L R N 次序）和中序法（也称对称序法，即按 L N R 次序）。这三种方法相互之间有关联。若已知一棵二叉树的前序序列是 BEFCGDH，中序序列是 FEBGCHD，则它的后序序列必是 F E G H D C B 。



解：法 1：先由已知条件画图，再后序遍历得到结果；

法 2：不画图也能快速得出后序序列，只要找到根的位置特征。由前序先确定 root，由中序先确定左子树。例如，前序遍历 **BEFCGDH 中，根结点在最前面，是 B；则后序遍历中 B 一定在最后面。**

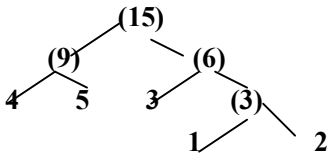
法 3：递归计算。如 B 在前序序列中第一，中序中在中间（可知左右子树上有哪些元素），则在后序中必为最后。如法对 B 的左右子树同样处理，则问题得解。

8. 中序遍历的递归算法平均空间复杂度为___O(n)___。

答：即递归最大嵌套层数，即栈的占用单元数。精确值应为树的深度 k+1，包括叶子的空域也递归了一次。

9. 用 5 个权值{3, 2, 4, 5, 1}构造的哈夫曼（Huffman）树的带权路径长度是___33___。

解：先构造哈夫曼树，得到各叶子的路径长度之后便可求出 WPL＝（4＋5＋3）×2＋（1＋2）×3=33



（注：两个合并值先后不同会导致编码不同，即哈夫曼编码不唯一）

（注：合并值应排在叶子值之后）

（注：原题为选择题： A．32　　B．33　　C．34　　D．15）

三、单项选择题（）

（ C ）1．　不含任何结点的空树_____。

（A）是一棵树;　　（B）是一棵二叉树;

（C）是一棵树也是一棵二叉树;　　（D）既不是树也不是二叉树

答：以前的标答是 B，因为那时树的定义是 n≥1

（ C ）2．二叉树是非线性数据结构，所以_____。

（A）它不能用顺序存储结构存储;　　（B）它不能用链式存储结构存储;

（C）顺序存储结构和链式存储结构都能存储;　　（D）顺序存储结构和链式存储结构都不能使用

（ C ）3.具有 n(n>0)个结点的完全二叉树的深度为_____。

（A）$\lceil \log_2(n) \rceil$　（B）$\lfloor \log_2(n) \rfloor$　（C）$\lfloor \log_2(n) \rfloor+1$　（D）$\lceil \log_2(n)+1 \rceil$

注 1：$\lceil x \rceil$表示不小于 x 的最小整数；$\lfloor x \rfloor$表示不大于 x 的最大整数，它们与[ ]含义不同！

注 2：选（A）是错误的。例如当 n 为 2 的整数幂时就会少算一层。似乎$\lfloor \log_2(n) +1 \rfloor$是对的？

（ A ）4．把一棵树转换为二叉树后，这棵二叉树的形态是_____。

（A）唯一的　　（B）有多种

（C）有多种，但根结点都没有左孩子　　（D）有多种，但根结点都没有右孩子

5. 从供选择的答案中，选出应填入下面叙述__？__内的最确切的解答，把相应编号写在答卷的对应栏内。

树是结点的有限集合，它_A_根结点，记为 T。其余的结点分成为 m（m≥0）个_B_的集合 T1，T2，…，Tm，每个集合又都是树，此时结点 T 称为 T_i 的父结点，T_i 称为 T 的子结点（1≤i≤m）。一个结点的子结点个数为该结点的_C_。

供选择的答案

A：　①有 0 个或 1 个　②有 0 个或多个　③有且只有 1 个　④有 1 个或 1 个以上

B：　①互不相交　② 允许相交　③ 允许叶结点相交　④ 允许树枝结点相交

C：　①权　② 维数　③ 次数（或度）　④ 序

答案：ABC＝1，1，3

6. 从供选择的答案中，选出应填入下面叙述___？___内的最确切的解答，把相应编号写在答卷的对应栏内。

二叉树__A__。在完全的二叉树中，若一个结点没有__B__，则它必定是叶结点。每棵树都能惟一地转换成与它对应的二叉树。由树转换成的二叉树里，一个结点 N 的左子女是 N 在原树里对应结点的__C__，而 N 的右子女是它在原树里对应结点的__D__。

供选择的答案

A： ①是特殊的树 ②不是树的特殊形式 ③是两棵树的总称 ④有是只有二个根结点的树形结构

B: ①左子结点 ② 右子结点 ③ 左子结点或者没有右子结点 ④ 兄弟

C～D： ①最左子结点 ② 最右子结点 ③ 最邻近的右兄弟 ④ 最邻近的左兄弟 ⑤ 最左的兄弟 ⑥ 最右的兄弟

答案：A=_____ B=_____ C=_____ D=_____

答案：ABCDE＝2，1，1，3

四、简答题（）

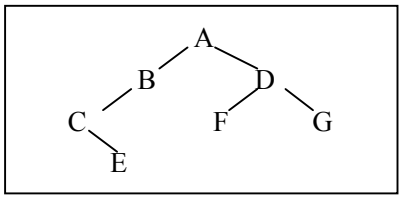
1. **一棵度为 2 的树与一棵二叉树有何区别？**

答：度为 2 的树从形式上看与二叉树很相似，但它的子树是无序的，而二叉树是有序的。即，在一般树中若某结点只有一个孩子，就无需区分其左右次序，而在二叉树中即使是一个孩子也有左右之分。

2.设如下图所示的二叉树 B 的存储结构为二叉链表，root 为根指针，结点结构为：（lchild,data,rchild）。其中 lchild，rchild 分别为指向左右孩子的指针，data 为字符型，root 为根指针，试回答下列问题：

1. 对下列二叉树 B，执行下列算法 traversal(root)，试指出其输出结果；
2. 假定二叉树 B 共有 n 个结点，试分析算法 traversal(root)的时间复杂度。（共 8 分）

二叉树 B




```
C 的结点类型定义如下：
struct node
{char data;
struct node *lchild, rchild;
};

C 算法如下：
void traversal(struct node *root)
{if (root)
  {printf("%c", root->data);
   traversal(root->lchild);
   printf("%c", root->data);
   traversal(root->rchild);
   }
}
```

解：这是“先根再左再根再右”，比前序遍历多打印各结点一次，输出结果为：A B C C E E B A D F F D G G

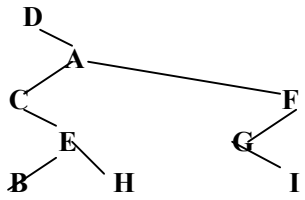
特点：①每个结点肯定都会被打印两次；②但出现的顺序不同，其规律是：凡是有左子树的结点，必间隔左子树的全部结点后再重复出现；如 A，B，D 等结点。反之马上就会重复出现。如 C，E，F，G 等结点。

时间复杂度以访问结点的次数为主，精确值为 2*n，时间渐近度为 O(n).

3.给定二叉树的两种遍历序列，分别是：

前序遍历序列：D，A，C，E，B，H，F，G，I； 中序遍历序列：D，C，B，E，H，A，G，I，F，试画出二叉树 B，并简述由任意二叉树 B 的前序遍历序列和中序遍历序列求二叉树 B 的思想方法。

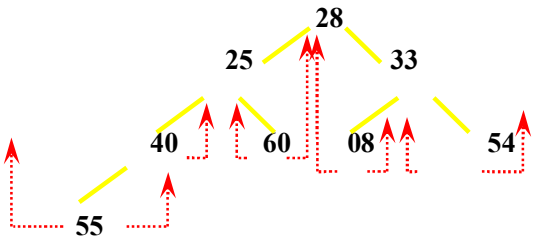
解：方法是：由前序先确定 root，由中序可确定 root 的左、右子树。然后由其左子树的元素集合和右子树的集合对应前序遍历序列中的元素集合，可继续确定 root 的左右孩子。将他们分别作为新的 root，不断递归，则所有元素都将被唯一确定，问题得解。

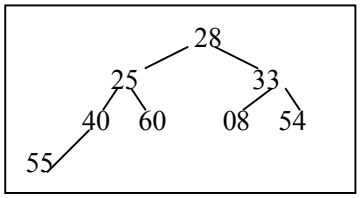


4.给定如图所示二叉树 T，请画出与其对应的中序线索二叉树。

解：要遵循中序遍历的轨迹来画出每个前驱和后继。

中序遍历序列：55 40 25 60 28 08 33 54





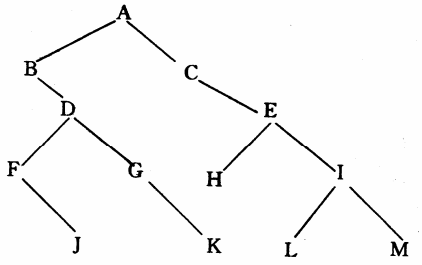
五、阅读分析题（）

1.试写出如图所示的二叉树分别按先序、中序、后序遍历时得到的结点序列。

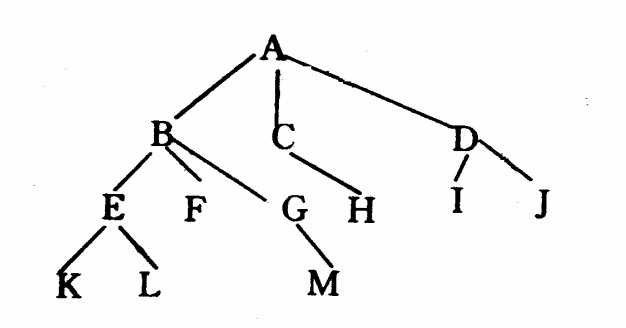
答：DLR：A B D F J G K C E H I L M

LDR: B F J D G K A C H E L I M

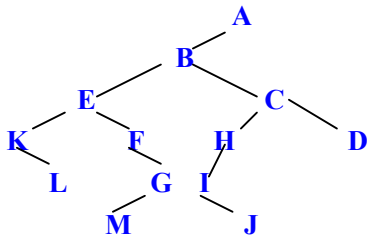
LRD：J F K G D B H L M I E C A



2. （P60 4-27）把如图所示的树转化成二叉树。



答：注意全部兄弟之间都要连线（包括度为 2 的兄弟）,并注意原有连线结点一律归入左子树，新添连线结点一律归入右子树。

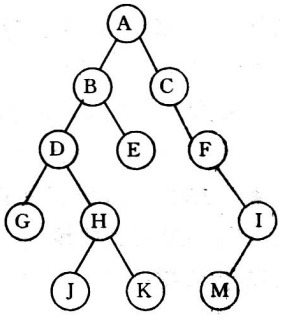


答：这是找结点后继的程序。
共有 3 处错误。
注：当 rtag＝1 时说明内装后继指针，可直接返回，第一句无错。
当 rtag＝0 时说明内装右孩子指针，但孩子未必是后继，需要计算。中序遍历应当先左再根再右，所以应当找左子树直到叶子处。**r=r->lchild; 直到 LTag=1 ；**

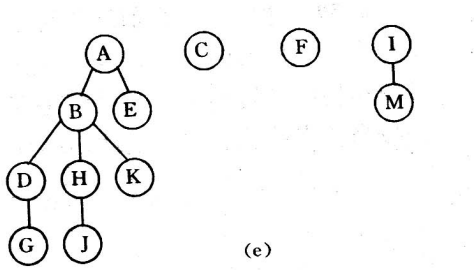
3

```
BiTree InSucc(BiTree q){
//已知 q 是指向中序线索二叉树上某个结点的指针，
//本函数返回指向*q 的后继的指针。
r=q->rchild;                    //应改为 r=q;
if(!r->rtag)
      while(!r->rtag)r=r->rchild;     //  应  改  为
while(!r->Ltag) r=r->Lchild;
      return r;   //应改为 return   r->rchild;
```

4.画出和下列二叉树相应的森林。



答：注意根右边的子树肯定是森林，
而孩子结点的右子树均为兄弟。



六、算法设计题（）

1.编写递归算法，计算二叉树中叶子结点的数目。

解：思路：输出叶子结点比较简单，用任何一种遍历递归算法，凡是左右指针均空者，则为叶子，将其打印出来。

法一：核心部分为：

```
DLR(liuyu *root)        /*中序遍历     递归函数*/
{if(root!=NULL)
  {if((root->lchild==NULL)&&(root->rchild==NULL)){sum++; printf("%d\n",root->data);}
    DLR(root->lchild);
    DLR(root->rchild); }
  return(0);
}
```

法二：

```
int LeafCount_BiTree(Bitree T)//求二叉树中叶子结点的数目
{
```

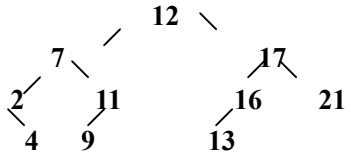
```
   if(!T) return 0; //空树没有叶子
   else if(!T->lchild&&!T->rchild) return 1; //叶子结点
   else return Leaf_Count(T->lchild)+Leaf_Count(T->rchild);//左子树的叶子数加
上右子树的叶子数
}//LeafCount_BiTree
```

注：上机时要先建树！例如实验二的方案一。

① 打印叶子结点值（并求总数）

思路：先建树，再从遍历过程中打印结点值并统计。

步骤 1　键盘输入序列 12，8，17，11，16，2，13，9，21，4，构成一棵二叉排序树。叶子结点值应该是 4，9, 13, 21，总数应该是 4.



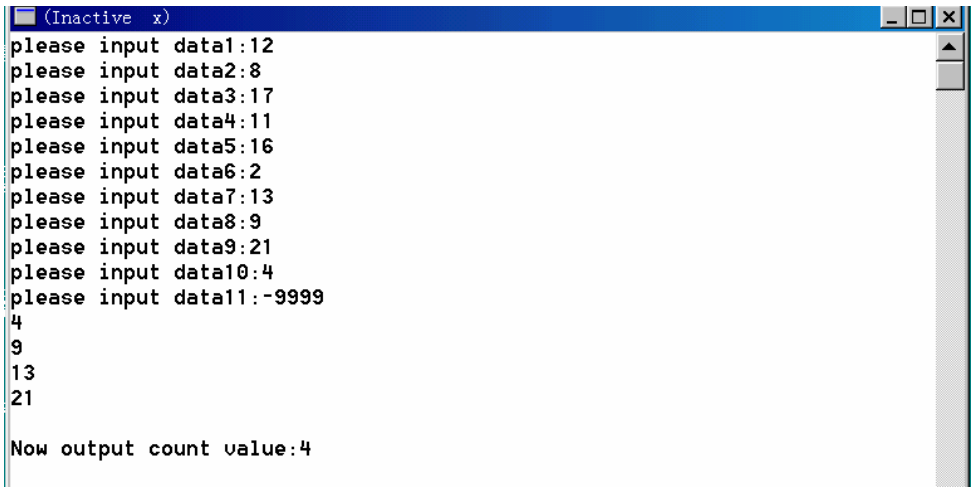
编程：　生成二叉树排序树之后，再中序遍历排序查找结点的完整程序如下：

说明部分为：

```
#include <stdio.h>
#include <stdlib.h>
typedef struct liuyu{int data;struct liuyu *lchild,*rchild;}test;
liuyu *root;
int sum=0;int m=sizeof(test);

void insert_data(int x)              /*如何生成二叉排序树？参见教材 P43C 程序*/
{ liuyu *p,*q,*s;
s=(test*)malloc(m);
s->data=x;
s->lchild=NULL;
s->rchild=NULL;

if(!root){root=s; return;}
p=root;
while(p)                             /*如何接入二叉排序树的适当位置*/
  {q=p;
if(p->data==x){printf("data already exist! \n");return;}
else if(x<p->data)p=p->lchild;    else p=p->rchild;
   }
if(x<q->data)q->lchild=s;
else q->rchild=s;
}

DLR(liuyu *root)        /*中序遍历     递归函数*/
{if(root!=NULL)
  {if((root->lchild==NULL)&&(root->rchild==NULL)){sum++; printf("%d\n",root->data);}
   DLR(root->lchild);
```

```
   DLR(root->rchild); }
 return(0);
}
 main()                    /*先生成二叉排序树，再调用中序遍历递归函数进行排序输出*/
{int i,x;
i=1;
root=NULL;                  /*千万别忘了赋初值给 root!*/
do{printf("please input data%d:",i);
i++;
scanf("%d",&x);             /*从键盘采集数据，以-9999 表示输入结束*/
if(x==-9999){
    DLR(root);
    printf("\nNow output count value:%d\n",sum);
    return(0);   }
    else insert_data(x);}              /*调用插入数据元素的函数*/
while(x!=-9999);
return(0);}
```

执行结果：



若一开始运行就输入-9999，则无叶子输出，sum=0。

2.写出求二叉树深度的算法，先定义二叉树的抽象数据类型。　　（）

编写递归算法，求二叉树中以元素值为 x 的结点为根的子树的深度。

答；设计思路：只查后继链表指针，若左或右孩子的左或右指针非空，则层次数加 1；否则函数返回。但注意，递归时应当从叶子开始向上计数，否则不易确定层数。

```
int depth(liuyu*root)       /*统计层数*/
{int d,p;                     /*注意每一层的局部变量 d,p 都是各自独立的*/
p=0;
if(root==NULL)return(p);        /*找到叶子之后才开始统计*/
else{
d=depth(root->lchild);
if(d>p) p=d;                  /*向上回朔时，要挑出左右子树中的相对大的那个深度值*/
d=depth(root->rchild);
if(d>p)p=d;
```

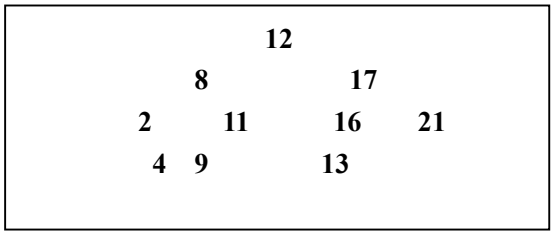
```
}
p=p+1;
return(p);
}
```

法二：

```
int Get_Sub_Depth(Bitree T,int x)//求二叉树中以值为 x 的结点为根的子树深度
{
   if(T->data==x)
   {
      printf("%d\n",Get_Depth(T)); //找到了值为 x 的结点,求其深度
      exit 1;
   }
   }
   else
   {
      if(T->lchild) Get_Sub_Depth(T->lchild,x);
      if(T->rchild) Get_Sub_Depth(T->rchild,x); //在左右子树中继续寻找
   }
}//Get_Sub_Depth
int Get_Depth(Bitree T)//求子树深度的递归算法
{
   if(!T) return 0;
   else
   {
      m=Get_Depth(T->lchild);
      n=Get_Depth(T->rchild);
      return (m>n?m:n)+1;
   }
}//Get_Depth
```

附：上机调试过程

步骤 1　键盘输入序列 12，8，17，11，16，2，13，9，21，4，构成一棵二叉排序树。层数应当为 4



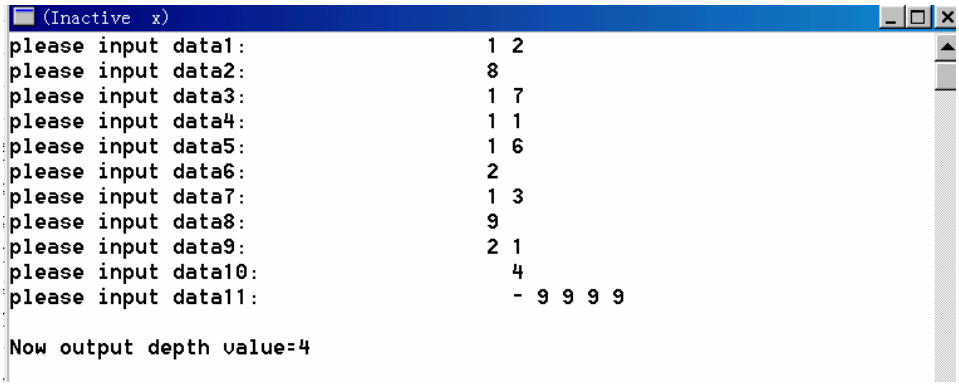
步骤 2：　执行求深度的函数，并打印统计出来的深度值。

完整程序如下：

```
#include <stdio.h>
#include <stdlib.h>
typedef struct liuyu{int data;struct liuyu *lchild,*rchild;}test;
liuyu *root;
int sum=0;int m=sizeof(test);
```

```
void insert_data(int x)              /*如何生成二叉排序树？参见教材 P43C 程序*/
{ liuyu *p,*q,*s;
s=(test*)malloc(m);
s->data=x;
s->lchild=NULL;
s->rchild=NULL;

if(!root){root=s; return;}
p=root;
while(p)                             /*如何接入二叉排序树的适当位置*/
  {q=p;
if(p->data==x){printf("data already exist! \n");return;}
else if(x<p->data)p=p->lchild;    else p=p->rchild;
   }
if(x<q->data)q->lchild=s;
else q->rchild=s;
}

int depth(liuyu*root)        /*统计层数*/
{int d,p;                            /*注意每一层的局部变量 d,p 都是各自独立的*/
p=0;
if(root==NULL)return(p);          /*找到叶子之后才开始统计*/
else{
d=depth(root->lchild);
if(d>p) p=d;                      /*向上回朔时，要挑出左右子树中的相对大的那个深度值*/
d=depth(root->rchild);
if(d>p)p=d;
}
p=p+1;
return(p);
}

void    main()                       /*先生成二叉排序树，再调用深度遍历递归函数进行统计并输出*/
{int i,x;
i=1;
root=NULL;                        /*千万别忘了赋初值给 root!*/
do{printf("please input data%d:",i);
i++;
scanf("%d",&x);                   /*从键盘采集数据，以-9999 表示输入结束*/
if(x==-9999){
        printf("\nNow output depth value=%d\n", depth (root)); return; }
    else insert_data(x);}                 /*调用插入数据元素的函数*/
while(x!=-9999);
return;}
```

执行结果：



3.编写按层次顺序（同一层自左至右）遍历二叉树的算法。

或：按层次输出二叉树中所有结点；

解：思路：既然要求从上到下，从左到右，则利用队列存放各子树结点的指针是个好办法。

这是一个循环算法，用 while 语句不断循环，直到队空之后自然退出该函数。

技巧之处：当根结点入队后，会自然使得左、右孩子结点入队，而左孩子出队时又会立即使得它的左右孩子结点入队，……以此产生了按层次输出的效果。

```
level(liuyu*T)
/* liuyu *T,*p,*q[100];     假设 max 已知*/
{int f,r;
f=0; r=0;                    /*置空队*/
r=(r+1)%max;
q[r]=T;                      /*根结点进队*/
while(f!=r)                 /*队列不空*/
{f=(f+1%max);
p=q[f];                      /*出队*/
printf("%d",p->data);                /*打印根结点*/
if(p->lchild){r=(r+1)%max; q[r]=p->lchild;}        /*若左子树不空，则左子树进队*/
if(p->rchild){r=(r+1)%max; q[r]=p->rchild;}        /*若右子树不空，则右子树进队*/
}
return(0);
}
```

法二：

```
void LayerOrder(Bitree T)//层序遍历二叉树
{
  InitQueue(Q); //建立工作队列

  EnQueue(Q,T);
  while(!QueueEmpty(Q))
  {
    DeQueue(Q,p);
    visit(p);
    if(p->lchild) EnQueue(Q,p->lchild);
    if(p->rchild) EnQueue(Q,p->rchild);
  }
```

}//LayerOrder

可以用前面的函数建树，然后调用这个函数来输出。

完整程序如下（已上机通过）

```
#include <stdio.h>
#include <stdlib.h>
#define max 50
typedef struct liuyu{int data;struct liuyu *lchild,*rchild;}test;
liuyu *root,*p,*q[max];
int sum=0;int m=sizeof(test);

void insert_data(int x)              /*如何生成二叉排序树？参见教材 P43C 程序*/
{ liuyu *p,*q,*s;
s=(test*)malloc(m);
s->data=x;
s->lchild=NULL;
s->rchild=NULL;

if(!root){root=s; return;}
p=root;
while(p)                             /*如何接入二叉排序树的适当位置*/
  {q=p;
if(p->data==x){printf("data already exist! \n");return;}
else if(x<p->data)p=p->lchild;    else p=p->rchild;
   }
if(x<q->data)q->lchild=s;
else q->rchild=s;
}

level(liuyu*T)
/* liuyu *T,*p,*q[100];     假设 max 已知*/
{int f,r;
f=0; r=0;                        /*置空队*/
r=(r+1)%max;
q[r]=T;                          /*根结点进队*/
while(f!=r)                 /*队列不空*/
{f=(f+1%max);
p=q[f];                          /*出队*/
printf("%d",p->data);                /*打印根结点*/
if(p->lchild){r=(r+1)%max; q[r]=p->lchild;}        /*若左子树不空，则左子树进队*/
if(p->rchild){r=(r+1)%max; q[r]=p->rchild;}        /*若右子树不空，则右子树进队*/
}
return(0);
}
```

```
void    main()                    /*先生成二叉排序树，再调用深度遍历递归函数进行统计并输出*/
{int i,x;
i=1;
root=NULL;                    /*千万别忘了赋初值给 root!*/
do{printf("please input data%d:",i);
i++;
scanf("%d",&x);                    /*从键盘采集数据，以-9999 表示输入结束*/
if(x==-9999){
        printf("\nNow output data value:\n", level(root)); return; }
    else insert_data(x);}                    /*调用插入数据元素的函数*/
while(x!=-9999);
return;}
```

4. 已知一棵具有 n 个结点的完全二叉树被顺序存储于一维数组 A 中，试编写一个算法打印出编号为 i 的结点的双亲和所有的孩子。

答：首先，**由于是完全二叉树，不必担心中途会出现孩子为 null 的情况。**

其次分析：结点 i 的左孩子为 2i，右孩子为 2i+1;直接打印即可。

```
Printf(“Left_child=”, %d, v[2*i].data; “Right_child=”, %d, v[2*i+1].data;);
但其双亲是 i/2，需先判断 i 为奇数还是偶数。若 i 为奇数，则应当先 i-- ，然后再除以 2。
If(i/2!=0)i--;
Printf(“Parents=”, %d, v[i/2].data;);
```

5.编写算法判别给定二叉树是否为完全二叉树。

答：

```
int IsFull_Bitree(Bitree T)//判断二叉树是否完全二叉树,是则返回 1,否则返回 0
{
  InitQueue(Q);
  flag=0;
  EnQueue(Q,T); //建立工作队列
  while(!QueueEmpty(Q))
  {
  {
    DeQueue(Q,p);
    if(!p) flag=1;
    else if(flag) return 0;
    else
    {
      EnQueue(Q,p->lchild);
      EnQueue(Q,p->rchild); //不管孩子是否为空,都入队列
    }
  }//while
  return 1;
}//IsFull_Bitree
```

分析:该问题可以通过层序遍历的方法来解决.与 6.47 相比,作了一个修改,不管当前结点是否有左右孩子,都入队列.这样当树为完全二叉树时,遍历时得到是一个连续的不包含空

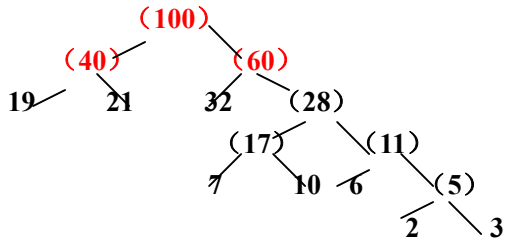
指针的序列.反之,则序列中会含有空指针.

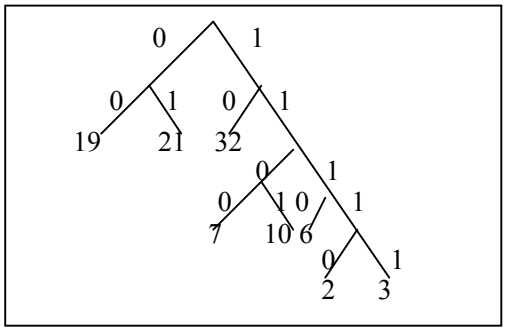
6. 假设用于通信的电文仅由 8 个字母组成，字母在电文中出现的频率分别为 0.07，0.19，0.02，0.06，0.32，0.03，0.21，0.10。试为这 8 个字母设计哈夫曼编码。使用 0～7 的二进制表示形式是另一种编码方案。对于上述实例，比较两种方案的优缺点。

解：方案 1；哈夫曼编码

先将概率放大 100 倍，以方便构造哈夫曼树。

w={7,19,2,6,32,3,21,10}，按哈夫曼规则：【[（2,3），6], (7,10)】，……19, 21, 32





方案比较：

字母编号	对应编码	出现频率
1	1100	0.07
2	00	0.19
3	11110	0.02
4	1110	0.06
5	10	0.32
6	11111	0.03

字母编号	对应编码	出现频率
1	000	0.07
2	001	0.19
3	010	0.02
4	011	0.06
5	100	0.32
6	101	0.03

方案 1 的 WPL＝2(0.19+0.32+0.21)+4(0.07+0.06+0.10)+5(0.02+0.03)=1.44+0.92+0.25=2.61

方案 2 的 WPL＝3(0.19+0.32+0.21+0.07+0.06+0.10+0.02+0.03)=3

结论：哈夫曼编码优于等长二进制编码